



PENERBIT ANDI

1010110110

Organisasi dan Arsitektur Komputer

Syahrul

AN
UR

Organisasi dan Arsitektur Komputer

Syahrul

Penerbit ANDI Yogyakarta



Organisasi dan Arsitektur Komputer

Oleh: Syahrul

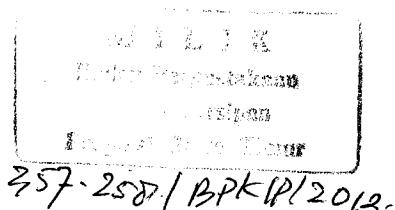
Hak Cipta © 2010 pada Penulis

Editor : Octaviani HS

Setting : Sri Sulistiyanı

Desain Cover : dan_dut

Korektor : Suci Nurasih / Aktor Sadewa



Hak Cipta dilindungi undang-undang.

Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apapun, baik secara elektronis maupun mekanis, termasuk memfotocopy, merekam atau dengan sistem penyimpanan lainnya, tanpa izin tertulis dari Penulis.

Penerbit: C.V ANDI OFFSET (Penerbit ANDI)

Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta
55281

Percetakan: ANDI OFFSET

Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta
55281

Perpustakaan Nasional: Katalog dalam Terbitan (KDT)

Syahrul

Organisasi dan Arsitektur Komputer/ Syahrul;

— Ed. 1. — Yogyakarta: ANDI,

19 18 17 16 15 14 13 12 11 10

xxii + 650 hlm.; 16 x 23 Cm.

10 9 8 7 6 5 4 3 2 1

ISBN: 978 – 979 – 29 – 1546 – 4

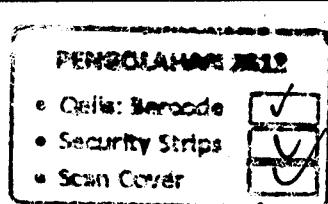
I. Judul

1. Computer Architectural

DDC'21 : 004.22

Buku ini kupersembahkan kepada:

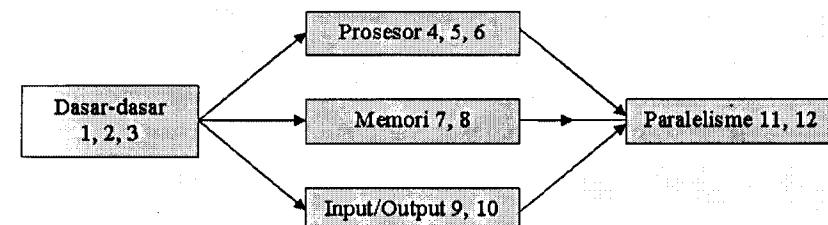
Istriku dan Anak-anakku yang Kucintai



KATA PENGANTAR

Buku ini bertujuan sebagai dasar untuk pembelajaran organisasi dan arsitektur komputer. Untuk belajar organisasi dan arsitektur komputer dipersyaratkan mahasiswa telah mengenal materi sistem digital dan pemrograman komputer. Dan juga sangat membantu jika mahasiswa telah mengerti tentang elektronika digital.

Buku ini disusun berdasarkan pengalaman penulis dalam mengajarkan materi kuliah *Organisasi dan Arsitektur Komputer* yang sebagian besar bersumber dari *textbook* dan buku-buku referensi komputer terkemuka lainnya yang banyak digunakan saat ini. Materi dalam buku ini disusun atas lima modul dengan struktur tiga-lapisan seperti yang dideskripsikan pada gambar berikut:



Tiga bab pertama membentuk lapisan atas yang menyediakan dasar-dasar organisasi dan arsitektur komputer. Setelah mempelajari bab tersebut, mahasiswa mempelajari salah satu bab dari pohon deskripsi. Lapisan tengah terdiri atas Bab 4, 5, 6, 7, 8, 9 dan 10 yang menjelaskan desain sub-sub sistem komputer. Lapisan terakhir terdiri atas Bab 11 dan 12 yang meliputi topik-topik khusus arsitektur komputer yaitu prinsip paralelisme dan pengantar multiprosesor yang merupakan komputer kinerja tinggi.

Bab 1 merupakan pengantar atau prinsip-prinsip dasar komputer modern. Bab ini membahas materi hardware dan software komputer dan menyediakan sekilas tentang konsep interupsi dan teknik-teknik I/O.

- Bab 2 memberikan gambaran tentang sejarah dan evolusi komputer dan memberikan perbedaan generasi-generasi komputer.
- Bab 3 menjelaskan atribut-atribut dasar sebuah komputer seperti set instruksi, modus-modus pengalamatan dan tipe data. Di sini juga diberikan perbedaan antara arsitektur mesin RISC dan CISC.
- Bab 4 menjelaskan algoritma aritmetika *fixed-point* dan *floating-point* serta dijelaskan pula penjumlahan serial dan paralel.
- Bab 5 difokuskan pada organisasi datapath dan desain ALU serta dijelaskan pula tentang mikro-operasi dan bahasa transfer register.
- Bab 6 menghadirkan teknik-teknik desain unit kontrol yang meliputi *hardwired control unit* dan *microprogrammed control unit*.
- Bab 7 difokuskan pada teknologi-teknologi memori dan desain memori utama.
- Bab 8 menjelaskan teknik-teknik berbagai peningkatan memori untuk tujuan penanganan kinerja dan masalah-masalah kapasitas yang meliputi memori cache, memori virtual dan interleaving memori.
- Bab 9 menjelaskan urutan komunikasi antara unit-unit internal. Berbagai *I/O controller*, standar-standar bus, mekanisme interupsi dan teknik-teknik I/O merupakan materi pokok dalam bab ini.
- Bab 10 menjelaskan berbagai macam peripheral komputer.
- Bab 11 memberikan penjelasan tentang *pipelining* yang merupakan teknik pemecahan tugas menjadi sub-sub tugas untuk dikerjakan secara simultan. Di sini juga dijelaskan tentang *vector computing*.
- Bab 12 memberikan sekilas pengantar tentang multiprosesor yang merupakan salah satu teknik peningkatan kinerja komputer menggunakan teknik paralelisme.

Saya sangat mengharapkan saran, koreksi dan komentar dari para pembaca yang budiman untuk dapat mengarahkan buku ini agar dapat lebih baik dan lebih sempurna.

Akhirnya saya mengucapkan terima kasih kepada semua pihak yang telah memberikan masukan dan saran yang sangat berarti dalam penerbitan buku ini. Berbagi ilmu pengetahuan merupakan amal jariah.

Mohon maaf jika terdapat banyak kekurangan dan kekhilafan. Hanya Allah yang Maha Mengetahui dan mempunyai ilmu yang sangat luas tak bertezi.

Syahrul

e-mail: syahrul_syl@yahoo.com

DAFTAR ISI

Kata Pengantar	v
Kata Pengantar	ix
BAB 1 ORGANISASI KOMPUTER MODERN	1
1.1 Manusia dan Komputer	2
1.2 Pengguna dan Komputer.....	3
1.2.1 Karakteristik Komputer	5
1.2.2 Komputer Digital	5
1.2.3 Hardware dan Software	7
1.2.4 Lapisan-Lapisan pada Komputer Modern	8
1.2.5 Software Aplikasi dan Software Sistem	10
1.3 Organisasi dan Arsitektur Komputer	10
1.3.1 Stored Program Concept	13
1.3.2 Memori Utama dan Memori Sekunder	14
1.3.3 Pengontrol Perangkat	15
1.3.4 Sinyal Antarmuka Perangkat	17
1.3.5 I/O Driver	18
1.4 Memori Utama	19
1.4.1 Komunikasi CPU – Memori	22
1.4.2 Adresibilitas Memori	24
1.5 Operasi CPU	25
1.5.1 Format Instruksi dan Siklus Instruksi	25
1.5.2 CPU State	27
1.5.3 Register-Register CPU	28
1.5.4 Clock	29
1.5.5 Operasi Makro dan Operasi Mikro	30
1.6 Konsep Interupsi	32
1.7 Teknik-Teknik I/O	36
1.8 Konsep Bus	39
1.8.1 Siklus Bus	44
1.9 Jenis-Jenis Komputer	45
1.10 Booting Sequence	47
1.11 Faktor-Faktor Kinerja Komputer	48

1.12	Pengukuran Kinerja Sistem	49
1.12.1	SPEC Rating	54
	Ringkasan	56
	Soal-soal Ulangan	59
	Soal-soal Latihan	62
BAB 2	EVOLUSI KOMPUTER	67
2.1	Proses Desain Komputer	67
2.2	Struktur Komputer	68
2.3	Fungsi Komputer	71
2.4	Dimensi-Dimensi Evolusi Komputer	72
2.4.1	Evolusi Mode-mode Penggunaan Komputer	73
2.4.2	Evolusi Arsitektur CPU Dasar	73
2.4.3	Evolusi Unit Kontrol dan Penerjemahan Instruksi	74
2.4.4	Evolusi Teknik-teknik Memori Utama	75
2.4.5	Evolusi Penanganan Siklus Instruksi	76
2.4.6	Evolusi Teknik-Teknik I/O	77
2.4.7	Evolusi Software Sistem	77
2.5	Sejarah Komputer	78
2.5.1	Komputer Generasi Pertama	79
2.5.2	Komputer Generasi Kedua	83
2.5.3	Komputer Generasi Ketiga	85
2.5.4	Komputer Generasi Keempat	87
2.5.5	Komputer Generasi Kelima	88
	Ringkasan	90
	Soal-soal Ulangan	91
	Soal-soal Latihan	92
BAB 3	ARSITEKTUR CPU DAN SET INSTRUKSI	93
3.1	CISC Banding RISC	93
3.1.1	Trend Teknologi CISC	94
3.1.2	Kelemahan CISC	95
3.1.3	Konsep RISC	95
3.1.4	Mikroprosesor RISC	96
3.2	Desain Set Instruksi	97
3.2.1	Bahasa Rakitan (Assembly)	98
3.2.2	CPU Berbasis Akumulator	100
3.2.3	CPU Berbasis Register	102
3.2.4	CPU Berbasis Stack	105

3.2.5	Panjang Instruksi	108
3.2.6	Format Instruksi	109
3.2.7	Lokasi Operand	111
3.2.8	Lokasi Hasil Pemrosesan	111
3.2.9	Variasi Panjang Instruksi	111
3.2.10	Urutan Data Little-Endian dan Big-Endian	112
3.2.11	Tipe Instruksi	114
3.2.12	Makro dan Subrutin	118
3.3	Mode Pengalamatan	120
3.3.1	Pengalamatan Immediate	121
3.3.2	Pengalamatan Langsung	122
3.3.3	Pengalamatan Tak Langsung	123
3.3.4	Pengalamatan Tak Langsung Register	124
3.3.5	Pengalamatan Register	124
3.3.6	Pengalamatan Indeks	125
3.3.7	Pengalamatan Relatif	126
3.3.8	Pengalamatan Base Register	127
3.3.9	Pengalamatan Stack	127
	Ringkasan	129
	Soal-soal Ulangan	130
	Soal-soal Latihan	132
BAB 4	ARITMETIKA KOMPUTER	133
4.1	Jenis-Jenis Aritmetika Komputer	134
4.1.1	Komponen-Komponen Hardware	135
4.2	Representasi Data	136
4.2.1	Data Karakter	136
4.2.2	Data Alamat	137
4.3	Data Biner	137
4.3.1	Bilangan Titik Tetap	138
4.3.2	Bilangan Titik-Mengambang	141
4.3.3	Normalisasi	142
4.3.4	Konversi Format Scientific ke Format Standar IEEE 754 ..	142
4.4	Aritmetika Bilangan Titik-Tetap	144
4.4.1	Perluasan Bit Tanda	144
4.4.2	Penjumlahan Integer	145
4.4.3	Penjumlahan Komplemen-2	154
4.4.4	Pengurangan Integer	156
4.4.5	Perkalian Bilangan Tidak Bertanda	158

4.4.6	Perkalian Bilangan Bertanda	162
4.4.7	Pembagian Integer	165
4.5	Aritmetika Bilangan Titik-Mengambang	170
4.5.1	Penjumlahan Bilangan Titik-Mengambang	170
4.5.2	Pengurangan Bilangan Titik-Mengambang	172
4.5.3	Perkalian Bilangan Titik-Mengambang	172
4.5.4	Pembagian Bilangan Titik-Mengambang	173
	Ringkasan	174
	Soal-soal Ulangan	175
	Soal-soal Latihan	176
BAB 5	DESAIN PROSESOR DAN DATAPATH	179
5.1	Fungsi Prosesor	179
5.2	Tujuan Desain Prosesor	182
5.3	Proses Desain Prosesor	184
5.3.1	Langkah-Langkah Desain Prosesor	185
5.3.2	Bahasa Transfer Register	186
5.4	Organisasi Datapath	188
5.4.1	Datapath Titik-Tetap	189
5.4.2	Desain Unit Aritmetika	192
5.4.3	Desain Unit Logika	194
5.4.4	Desain Penggeser (Shifter)	195
5.4.5	Desain ALU	197
5.5	Antarmuka Memori Utama	199
5.5.1	Urutan Pembacaan Memori	199
5.5.2	Urutan Penulisan Memori	200
5.6	Register File	200
5.7	Datapath pada Instruksi Sederhana	200
5.7.1	Instruksi Halt	202
5.7.2	Instruksi NOOP	203
5.7.3	Instruksi JUMP	203
5.7.4	Instruksi LOAD	204
5.7.5	Instruksi STORE	205
5.8	Datapath pada Unit Titik-Mengambang	207
	Ringkasan	209
	Soal-soal Ulangan	210
	Soal-soal Latihan	211

BAB 6	UNIT KONTROL	213
6.1	Fungsi Unit Kontrol	213
6.2	Urutan Reset	218
6.3	Pengenalan dan Pelayanan Interupsi	219
6.4	Penanganan Situasi Abnormal	221
6.5	Siklus Instruksi	221
6.5.1	Pengambilan Instruksi	221
6.5.2	Kalkulasi Alamat Operand	224
6.5.3	Pengambilan Operand	224
6.5.4	Operasi Eksekusi	224
6.5.5	Pemilihan Desain Unit Kontrol	225
6.6	Hardwired Control Unit	227
6.6.1	Keuntungan Hardwired Control Unit	233
6.6.2	Kekurangan Hardwired Control Unit	234
6.7	Microprogrammed Control Unit	234
6.7.1	Permasalahan pada Microprogrammed Control Unit	236
6.7.2	Pengurangan Panjang Word Mikroinstruksi	238
6.7.3	Horizontal dan Vertical microprogramming	239
6.7.4	Pencabangan Tidak Bersyarat	241
6.7.5	Pencabangan Bersyarat	242
6.7.6	Hardware, Software, Firmware	242
6.7.7	Keuntungan Microprogramming	243
6.7.8	Kekurangan Microprogramming	243
	Ringkasan	244
	Soal-soal Ulangan	245
	Soal-soal Latihan	246
BAB 7	DESAIN MEMORI UTAMA SEMIKONDUKTOR	249
7.1	Parameter-Parameter Memori	249
7.2	Operasi Memori Secara Umum	251
7.2.1	Masukan-Masukan Alamat	252
7.2.2	Masukan R/\bar{W}	254
7.2.3	Enable Memori	256
7.3	Klasifikasi Memori	256
7.3.1	Kemampuan Baca/Tulis	257
7.3.2	Metode Akses	257
7.3.3	Mode-Mode Penggunaan dan Fungsi	257
7.3.4	Teknologi Memori	262
7.4	Arsitektur ROM	263

7.4.1	Larik Register	263
7.4.2	Decoder Alamat	264
7.4.3	Penyangga Output	265
7.5	Pewaktuan ROM	265
7.6	Jenis-Jenis ROM	267
7.6.1	MROM	267
7.6.2	PROM	268
7.6.3	EPROM	268
7.6.4	EEPROM	268
7.6.5	Memori Flash	269
7.7	Aplikasi ROM	271
7.7.1	Firmware	271
7.7.2	Memori Bootstrap	272
7.7.3	Tabel-Tabel Data	272
7.7.4	Pengonversi Data	273
7.7.5	Pembangkit Fungsi	273
7.8	RAM Semikonduktor	274
7.9	Arsitektur Semikonduktor	274
7.9.1	Operasi Baca	277
7.9.2	Operasi Tulis	277
7.9.3	Pemilih Chip	277
7.9.4	Penyemat Input/Output Bersama	278
7.10	Static RAM (SRAM)	278
7.10.1	Pewaktuan SRAM	279
7.10.2	Siklus Baca	280
7.10.3	Siklus Tulis	282
7.10.4	IC RAM Aktual	283
7.10.5	Desain Modul-Modul SRAM	286
7.11	Dynamic RAM (DRAM)	287
7.11.1	Penyegaran Ulang DRAM	289
7.11.2	Penyegaran Ulang Baris	290
7.11.3	Multiplexing Alamat Baris dan Kolumn	291
7.12	Memperbesar Ukuran Word dan Kapasitas	293
7.12.1	Memperbesar Ukuran Word	294
7.12.2	Memperbesar Kapasitas	296
7.13	Alokasi Memori Utama	300
	Ringkasan	303
	Soal-soal Ulangan	304

	Soal-soal Latihan	306
	BAB 8 TEKNIK MANAJEMEN MEMORI	309
8.1	Hierarki Memori	310
8.2	Kekurangan Memori Utama	313
8.3	Prefetch Instruksi	314
8.4	Interleave Memori	316
8.5	Buffer Tulis	320
8.6	Memori Cache	321
8.6.1	Prinsip Cache	322
8.6.2	Hit dan Miss	325
8.6.3	Pemetaan Langsung	325
8.6.4	Pemetaan Asosiatif Penuh	331
8.6.5	Pemetaan Asosiatif Set	332
8.6.6	Penggantian Cache	340
8.6.7	Kebijakan Penulisan Cache	343
8.6.8	Pengintaian dan Invalidasi	345
8.6.9	Kinerja Cache dan Hit Rate	346
8.6.10	Cache Gabung dan Cache Pisah	349
8.6.11	Cache Multi Level	351
8.6.12	Cache Mikroprosesor 80486	353
8.6.13	Invalidasi Cache pada Pentium	356
8.7	Memori Virtual	357
8.7.1	Keuntungan Memori Virtual	359
8.7.2	Mekanisme Memori Virtual	359
8.7.3	Paging	361
8.7.4	Waktu Akses Efektif Paging	369
8.7.5	Penggunaan Cache, TLB dan Paging Bersama Sekaligus ..	371
8.7.6	Segmentasi	373
8.7.7	Skema Segmentasi pada Intel 80286	374
8.7.8	Mekanisme Memori Virtual Intel 80386	379
8.8	Memori Asosiatif	383
	Ringkasan	386
	Soal-soal Ulangan	389
	Soal-soal Latihan	391
	BAB 9 ORGANISASI INPUT/OUTPUT	395
9.1	Pengontrol I/O DAN I/O Driver	396
9.1.1	Pengontrol Perangkat	397

9.1.2	I/O Driver	403
9.1.3	Perangkat keras Pengontrol	404
9.2	Port I/O	406
9.2.1	Instruksi IN	407
9.2.2	Instruksi OUT	407
9.2.3	Memory Mapped I/O dan Direct I/O	408
9.3	Transfer Sinkron dan Asinkron	409
9.3.1	Transfer Sinkron	409
9.3.2	Transfer Asinkron dan Handshaking	412
9.4	Penanganan Interupsi pada PC	415
9.4.1	Konsep Interupsi	416
9.4.2	Jenis-Jenis Interupsi	416
9.4.3	Pengindraan Interupsi	419
9.4.4	Pelayanan Interupsi	420
9.4.5	Kembali dari ISR	421
9.4.6	Enable dan Disable Interupsi	421
9.4.7	Interupsi Tersarang	422
9.4.8	Prioritas Interupsi	423
9.4.9	Penghalangan Interupsi yang Dapat Dipilih	423
9.4.10	Interupsi yang Tak Dapat Dihalangi	424
9.5	Teknik-Teknik I/O pada PC	424
9.5.1	Programmed I/O Mode	426
9.5.2	Interrupt Mode	428
9.5.3	DMA Mode	429
9.6	Port Paralel	433
9.7	Port Serial	435
9.7.1	Komunikasi Asinkron dan Sinkron	436
9.8	Antarmuka I/O Serial Modern	440
9.8.1	Universal Serial Bus	440
9.8.2	Firewire (IEEE 1394)	442
9.9	Siklus Bus	443
9.10	Organisasi Bus	444
9.11	Teknik-Teknik Arbitrasi Bus	446
9.11.1	Metode Arbitrasi Bus	447
9.12	Jenis Bus Komputer	449
9.12.1	Konsep Bus Tunggal	450
9.12.2	Bus I/O dan Isolated Memory	454
	Ringkasan	460

Soal-soal Ulangan	462
Soal-soal Latihan	465
BAB 10 PERIPHERAL KOMPUTER	467
10.1 Keyboard	468
10.1.1 Mechanical Keyswitch Keyboard	469
10.1.2 Membrane Keyswitch Keyboard	469
10.1.3 Capacitive Keyswitch Keyboard	469
10.1.4 Sistem Tata Letak Keyboard	470
10.2 Mouse	471
10.3 Tampilan CRT	472
10.4 Tampilan LCD	474
10.4.1 Operasi Dasar LCD	474
10.4.2 Struktur LCD dan Proses Produksi	476
10.4.3 Jenis-Jenis Tampilan	477
10.4.4 Prinsip-Prinsip Tampilan Warna	477
10.4.5 Konfigurasi Tampilan	477
10.5 Printer	478
10.6 Disk Magnetik	479
10.6.1 Mekanisme Pembacaan dan Perekaman	480
10.6.2 Track dan Spot	485
10.6.3 Sektor	486
10.6.4 Parameter-Parameter Kinerja Disk	488
10.6.5 Floppy Disk Drive Versus Hard Disk Drive	490
10.7 RAID	491
10.7.1 RAID Level 0	492
10.7.2 RAID Level 1	492
10.7.3 RAID Level 2	493
10.7.4 RAID Level 3	494
10.7.5 RAID Level 4	495
10.7.6 RAID Level 5	497
10.7.7 RAID Level 6	498
10.8 Pita Magnetik	499
10.9 Disk Optik	500
10.9.1 Compact Disc	500
10.9.2 Digital Versatile Disc	503
10.10 Modem	503
10.11 Scanner	504
10.12 Plotter	504

Ringkasan	506
Soal-soal Ulangan	507
Soal-soal Latihan	509
BAB 11 PIPELINING DAN PEMROSESAN VEKTOR	511
11.1 Strategi Peningkatan Kinerja	512
11.2 Klasifikasi Paralelisme (Klasifikasi Flynn)	514
11.2.1 Prosesor Skalar, Vektor, Superskalar, dan Prosesor yang di-Pipeline	518
11.3 Pipelining.....	519
11.3.1 Pipeline Instruksi	520
11.3.2 Frekuensi Clock	524
11.3.3 Pipeline Bubble	525
11.3.4 Pipeline Hazard	525
11.4 Pipeline Aritmetika.....	535
11.4.1 Pipeline Aritmetika Titik-Tetap	535
11.4.2 Pipeline Aritmetika Titik-Mengambang	538
11.5 Pipeline Mesin RISC	538
11.6 Pemrosesan Vektor	539
11.6.1 Operasi-Operasi Vektor	540
11.6.2 Register Vektor	541
11.6.3 Register Skalar	541
11.6.4 Memory-memory Vector Processor	542
11.7 Prosesor Larik	544
Ringkasan	546
Soal-soal Ulangan	548
Soal-soal Latihan	549
BAB 12 PENGANTAR SISTEM MULTIPROSESOR	551
12.1 Klasifikasi Multiprosesor	551
12.2 Multiprosesor Simetris	553
12.3 Struktur Interkoneksi	553
12.3.1 Struktur Bus Bersama	554
12.3.2 Struktur Memori Multiport	554
12.3.3 Struktur Crossbar Swich	555
12.3.4 Struktur Jaringan Multistage	556
12.3.5 Struktur Jaringan Hypercube	558
12.4 Cluster	559
12.5 Masalah Cache Coherence	560

12.6 Fault Tolerance	561
Ringkasan	563
Soal-Soal Ulangan	564
Soal-Soal Latihan	565
DAFTAR PUSTAKA	567
Lampiran A Sirkuit Terintegrasi Digital	569
Lampiran B Jawaban Soal-Soal Ulangan	603
Lampiran C Jawaban Soal-Soal Latihan	613
TENTANG PENULIS	641

BAB 1

ORGANISASI KOMPUTER MODERN

Sasaran bab ini:

1. Stored Program Concept
2. Unit-Unit Fungsional Komputer
3. Tingkatan Software dan Hardware
4. Konsep Interupsi
5. Konsep Bus
6. Modus Transfer Data
7. Jenis-Jenis Komputer
8. Boot Sequence
9. Pengukuran Kinerja Komputer

Saat ini komputer digunakan pada hampir semua bidang kehidupan: perbankan, rumah sakit, sekolah, penerbitan, manufaktur, hiburan, toko, perpustakaan, pabrik, biro perjalanan, hotel, penelitian, produksi film, mainan (*game*), pertemuan dan konferensi, pelayanan keamanan, perekaman absensi, akuntansi, pemilihan umum, pengadilan, universitas, penjara dan sebagainya.

Pembeli komputer mempertimbangkan sejumlah faktor sebelum menetapkan pilihan seperti kinerja yang baik, harga murah, kemampuan pemrograman yang mudah, biaya perawatan murah, dan sebagainya. Hal inilah yang menjadi sukma dari tujuan arsitektur komputer dan insinyur perancang dalam menyediakan fitur-fitur tersebut.

Perancangan komputer merupakan sesuatu yang canggih dan kompleks. Hingga saat ini, berbagai konsep dan teknik telah diterapkan dalam perancangan dan pembuatan komputer. Faktor-faktor tersebut seperti kinerja, biaya, teknologi komponen, jenis-jenis penggunaan dan

kapasitasnya merupakan persyaratan penentuan pemilihan konsep dan teknik yang digunakan dalam perancangan suatu komputer. Studi tentang konsep dan teknik tersebut menjadi sasaran dari buku ini.

Bab ini merupakan fondasi bagi bab-bab selanjutnya yang memberikan garis-garis besar organisasi dasar sebuah komputer agar membuat pembaca lebih nyaman dalam belajar mengenai berbagai arsitektur dan masalah-masalah perancangan.

1.1 MANUSIA DAN KOMPUTER

Manusia hingga kini melakukan penghitungan dalam tiga cara yang berbeda:

1. Penghitungan manual secara penuh yaitu penghitungan tanpa adanya peralatan selain otak dan jari-jari
2. Penghitungan manual yaitu menggunakan peralatan sederhana seperti mistar geser, *abacus* dan sebagainya.
3. Penghitungan otomatis yaitu menggunakan mesin komputer.

Pada Tabel 1.1 dibandingkan tiga macam penghitungan berdasarkan parameter-parameter berikut:

1. Kecepatan penghitungan
2. Keandalan hasil
3. Kompleksitas masalah yang ditangani
4. Peningkatan keterlibatan upaya manusia
5. Konsistensi

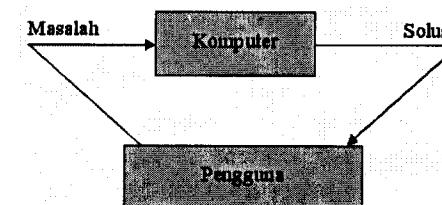
TABEL 1.1 Perbandingan jenis-jenis penghitungan

No.	Parameter	Penghitungan Manual	Penghitungan manual dengan alat sederhana	Penghitungan otomatis
1	Kecepatan	Rendah	Sedang	Tinggi
2	Keandalan	Rendah; berbeda setiap individu	Sedang	Tinggi
3	Kemungkinan kompleksitas masalah	Rendah	Sedang	Tinggi

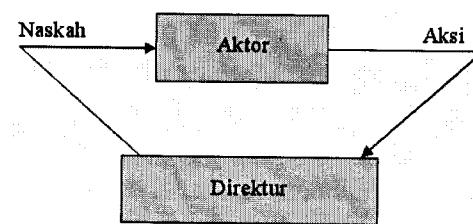
No.	Parameter	Penghitungan Manual	Penghitungan manual dengan alat sederhana	Penghitungan otomatis
4	Peningkatan upaya manusia	Sangat tinggi	Sedang	Sangat rendah
5	Konsistensi	Rendah karena kelelahan atau kebosanan	Sedang	Sangat tinggi
6	Pengaruh	Banyak masalah yang tak dapat diselesaikan dalam waktu yang layak	Sedikit lebih baik dari penghitungan manual	Banyak masalah yang dapat diselesaikan

1.2 PENGGUNA DAN KOMPUTER

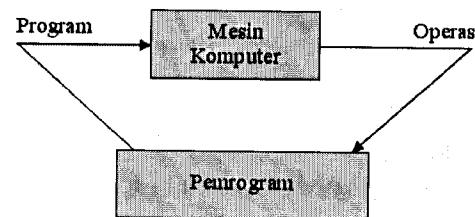
Komputer adalah alat elektronika yang dapat diprogram yang berfungsi untuk menyelesaikan berbagai macam permasalahan komputasi dan manipulasi data (Gambar 1.1). Kemampuan dasar komputer adalah melakukan kalkulasi atau komputasi aritmetika. Hubungan antara pengguna dan komputer adalah mirip dengan direktur sinema dan aktornya. Aktor 'melakukan *acting*' berdasarkan 'naskah' yang disiapkan oleh direktur (Gambar 1.2). Sedangkan komputer 'melakukan pemrosesan' berdasarkan 'program' yang disiapkan oleh pemrogram (Gambar 1.3).



Gambar 1.1 Pengguna dan komputer



Gambar 1.2 Peranan seorang aktor

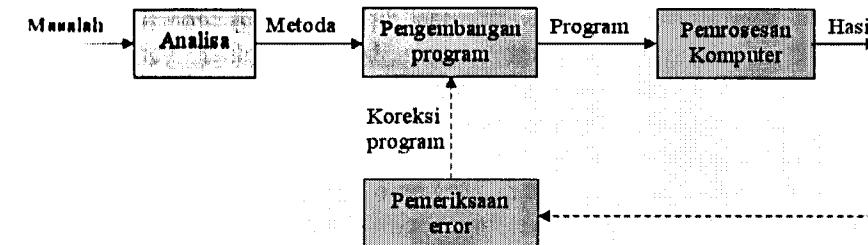


Gambar 1.3 Komputer dan pemrogram

Komputer digunakan untuk menyelesaikan masalah dalam beberapa bidang seperti sains, penelitian, perdagangan, administrasi, manufaktur dan sebagainya. Berikut sejumlah penggunaan komputer yang sederhana:

1. Perhitungan nilai rata-rata siswa dalam sebuah kelas
2. Penyiapan gaji bulanan untuk perusahaan
3. Perancangan gedung
4. Perhitungan pembayaran pajak oleh perusahaan bisnis
5. Penyiapan daftar bahan baku produksi
6. Penyiapan hasil ujian universitas

Proses penyelesaian suatu masalah dengan menggunakan komputer ditunjukkan pada Gambar 1.4. Dengan menganalisis suatu masalah, kita dapat memilih suatu metode (algoritma) yang akan diselasaikan. Kemudian, program dikembangkan untuk komputer pada setiap algoritma. Program ini dijalankan pada komputer dengan sampel data dan komputer memberikan hasilnya. Dengan melakukan pemeriksaan yang hati-hati pada hasil, kehadiran 'bug' (kutu, kesalahan pemrograman) dapat ditemukan. Koreksi yang tepat dilakukan terhadap program dan kesalahannya (*error*) dibetulkan. Jadi program yang telah teruji digunakan untuk menjalankan hasil pada komputer secara reguler menggunakan data yang tepat.



Gambar 1.4 Proses penyelesaian masalah

Keuntungan menggunakan komputer antara lain:

1. Pemrosesan (kalkulasi) cepat
2. Penyimpanan informasi yang besar
3. Membebaskan/mengurangi pekerjaan manual
4. Penyampaian pesan dan komunikasi

Kombinasi pemrosesan dan penyimpanan telah menghasilkan beberapa aplikasi baru seperti multimedia, internet, *desk top publishing*, dan sebagainya.

1.2.1 Karakteristik Komputer

Karakteristik utama komputer meliputi:

1. Komputasi yang sangat cepat
2. Bersifat konsisten yang tidak terpengaruh pada kelelahan, bosan, suka dan tidak suka, dan sebagainya.
3. Kapasitas penyimpanan yang besar (untuk data dan program)
4. Komputasi akurasi tinggi
5. Mesin serbaguna yang dapat diprogram untuk keperluan setiap pengguna

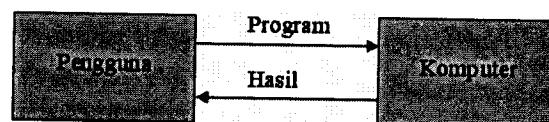
1.2.2 Komputer Digital

Pada umumnya komputer yang digunakan sekarang ini adalah komputer digital walaupun juga ada beberapa komputer analog. Komputer analog memantau/merasakan sinyal masukan yang mempunyai perubahan nilai yang kontinu. Hampir semuanya berupa variabel-variabel sinyal seperti tegangan, tekanan, temperatur, kecepatan dan sebagainya. Komputer

digital melakukan operasi pada informasi diskrit (digital) seperti bilangan. Komputer digital menggunakan sistem bilangan biner yang hanya mempunyai dua keadaan atau digit yaitu 0 dan 1 yang disebut bit (*binary digit*). Komputer digital menggunakan sirkuit digital di mana ada dua level berbeda baik sebagai masukan maupun sebagai keluaran. Kedua level tersebut dikenal sebagai logika 0 dan 1. Komputer modern adalah komputer digital yang mampu melakukan operasi-operasi aritmetika dan logika pada data dan memberikan suatu hasil (Gambar 1.5). Transistor merupakan komponen dasar pembangun komputer modern. Selanjutnya transistor digunakan untuk membuat gerbang logika (*logic gate*). Dan akhirnya logic gate membentuk komponen-komponen pembangun komputer seperti ALU (*arithmetic and logic unit*), *flip-flop*, *register*, *multiplexer*, *decoder*, *encoder*, memori, dan sebagainya untuk membentuk sebuah sistem komputer.



Gambar 1.5 Komputer sebagai mesin elektronika



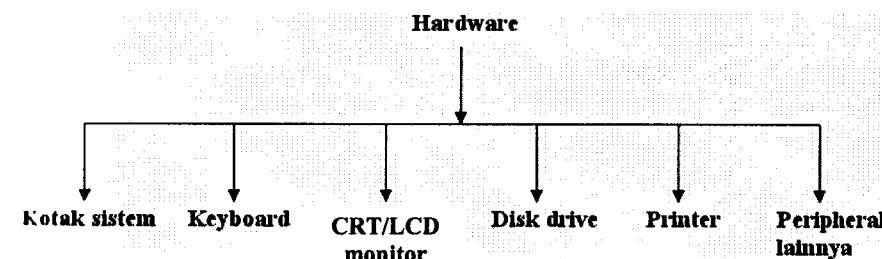
Gambar 1.6 Penyelesaian masalah dengan komputer

Untuk menyelesaikan suatu masalah, pengguna memasukkan program ke komputer (Gambar 1.6). Program mempunyai urutan instruksi yang menentukan jenis operasi yang dikerjakan. Jadi program memberikan langkah demi langkah untuk penyelesaian masalah. Komputer dapat melakukan berbagai instruksi seperti ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPARE, MOVE, INPUT, OUTPUT dan sebagainya. Semua daftar instruksi komputer disebut set instruksi. Sebuah program menggunakan gabungan instruksi sesuai dengan metode (algoritma) untuk menyelesaikan masalah. Program dapat didefinisikan sebagai sebuah urutan instruksi dengan data yang sesuai/tepat untuk penyelesaian suatu masalah. Komputer menganalisis setiap instruksi dan melakukan tindakan pada data.

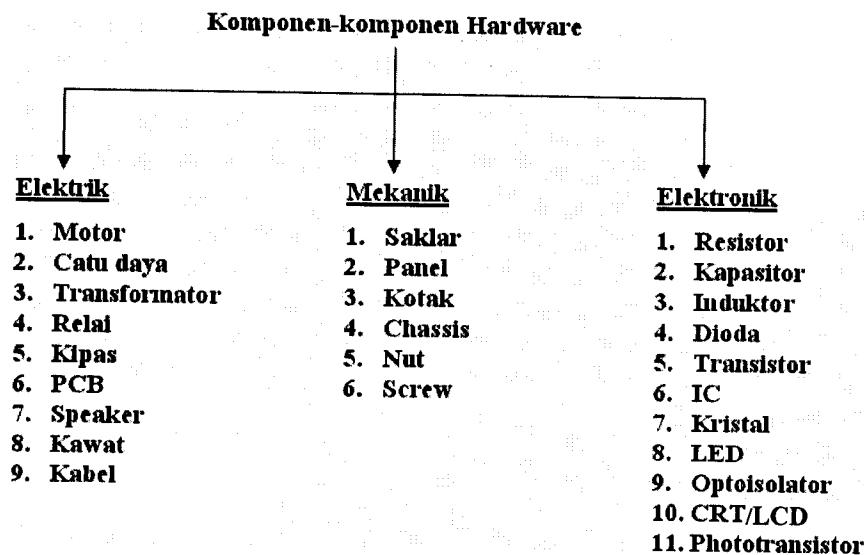
Program komputer untuk mesin elektronika harus terdiri atas 1 dan 0 saja. Operasi komputer ditentukan oleh instruksi sebagai informasi biner. *Operand* (data) diberikan dalam bentuk bilangan biner. Program yang demikian disebut program bahasa mesin. Walaupun membosankan melakukan penulisan program dalam bahasa mesin, semua program dahulu kala ditulis hanya dalam bahasa mesin karena tidak ada pilihan lain. Komputer dengan model yang berbeda mempunyai bahasa mesin yang berbeda pula dan karena itu program bahasa mesin untuk suatu komputer tak dapat dilanjutkan pada komputer yang lain dengan set instruksi yang berbeda. Seorang pemrogram komputer modern yang tidak dapat memprogram dalam bahasa mesin, maka dia mempunyai alternatif yang mudah seperti bahasa tingkat tinggi.

1.2.3 Hardware dan Software

Komputer digital adalah sebuah mesin elektronika yang dibangun dengan kecerdasan internal untuk mengeksekusi instruksi. Istilah *hardware* umumnya merujuk pada sirkuit-sirkuit elektronika yang terdapat di dalam mesin komputer. Modul utama hardware ditunjukkan pada Gambar 1.7. Secara praktis, istilah hardware digunakan untuk semua komponen fisik di dalam sebuah komputer termasuk mekanika, rakitan komponen-komponen listrik dan elektronika. Gambar 1.8 menunjukkan bagian-bagian hardware dalam sebuah komputer.

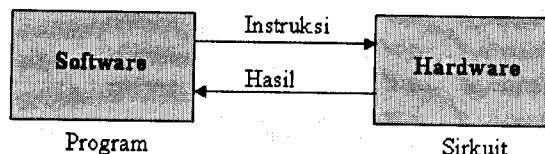


Gambar 1.7 Modul-modul hardware pada umumnya



Gambar 1.8 Komponen-komponen hardware komputer

Semua program adalah *software*. Software dikembangkan untuk menyelesaikan suatu masalah dan mengontrol hardware ketika program dieksekusi. Dengan kata lain, hardware mengikuti software (Gambar 1.9). Hardware dapat dilihat secara visual, sedangkan software merupakan suatu rancangan aksi logika yang tidak nampak secara visual.

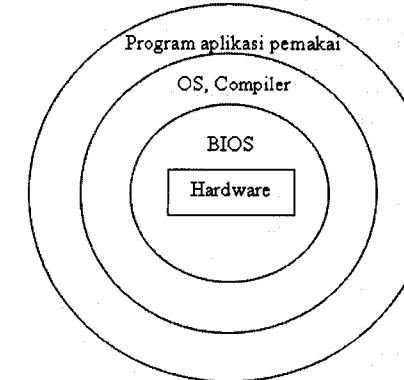


Gambar 1.9 Antarmuka hardware-software

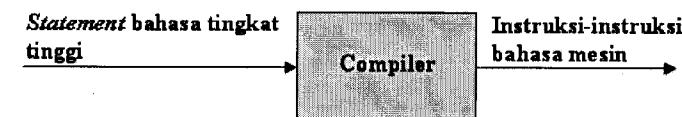
1.2.4 Lapisan-Lapisan pada Komputer Modern

Komputer modern bukan hanya pada mesin elektronikanya. Dia adalah sebuah sistem yang terdiri atas banyak lapisan. Lapisan paling dalam adalah unit hardware yang dikelilingi oleh lapisan-lapisan software lainnya, seperti yang ditunjukkan pada Gambar 1.10. Seorang pemrogram menulis sebuah program aplikasi dalam bahasa tingkat tinggi dengan menggunakan bilang-

an desimal dan pernyataan-pernyataan/*statement* dalam Bahasa Inggris. *Compiler* (kompiler) adalah sebuah penerjemah bahasa yang mengubah program bahasa tingkat tinggi menjadi program bahasa mesin yang ekivalen dan yang terdiri atas instruksi-instruksi dan bilangan biner (Gambar 1.11).



Gambar 1.10 Lapisan-lapisan sistem komputer



Gambar 1.11 Penggunaan kompiler

Program bahasa tingkat tinggi terdiri atas pernyataan-pernyataan, sedangkan program bahasa mesin mempunyai instruksi-instruksi. Sistem operasi adalah kumpulan dari program-program yang menyediakan berbagai fungsi dengan sasaran pada penawaran efisiensi dan kemudahan pada pengguna dan pemrogram. Berikut adalah fungsi-fungsi penting sistem operasi:

1. Penanganan pengguna komputer untuk permintaan sejumlah pelayanan
2. Penjadwalan program
3. Pengaturan operasi-operasi I/O
4. Pengaturan unit-unit hardware

BIOS (*Basic Input-Output control System*) adalah kumpulan *I/O driver* (program untuk pelaksanaan operasi-operasi I/O) untuk berbagai *device* (perangkat) peripheral dalam komputer. Program ini dapat dipanggil oleh program yang lain kapan saja suatu operasi I/O harus dikerjakan.

1.2.5 Software Aplikasi dan Software Sistem

Software komputer dikelompokkan ke dalam dua macam yaitu: software aplikasi dan software sistem. Program aplikasi adalah sebuah program yang menyelesaikan permasalahan pengguna. Beberapa contoh yang khas adalah: program penggajian, program kontrol inventaris, perhitungan pajak, penjadwal ruang kelas, software manajemen perpustakaan, software pemesanan (reservasi) kereta, *billing software* dan program-program mainan/*game*. Program tersebut merupakan program aplikasi karena tujuan/sasarannya adalah penanganan aplikasi spesifik (keperluan pengguna). Program sistem adalah sebuah program yang membantu utilisasi efisien sistem oleh program-program yang lain dan para pengguna. Umumnya dikembangkan untuk suatu jenis komputer dan tidak berkenaan dengan aplikasi spesifik atau aplikasi pengguna. Sistem operasi dan kompiler merupakan contoh software sistem.

1.3 ORGANISASI DAN ARSITEKTUR KOMPUTER

Arsitektur komputer adalah sebuah sains (ilmu) untuk tujuan perancangan sistem komputer. Tujuan seorang arsitek komputer adalah merancang sebuah sistem dengan kinerja yang tinggi dengan biaya yang layak, memenuhi semua persyaratan-persyaratan lainnya. "Arsitektur Komputer" memberikan berbagai atribut pada sistem komputer yang dibutuhkan oleh seorang pemrogram bahasa mesin atau seorang perancang software sistem untuk mengembangkan suatu program. Model konseptual arsitektur komputer memberikan informasi berikut:

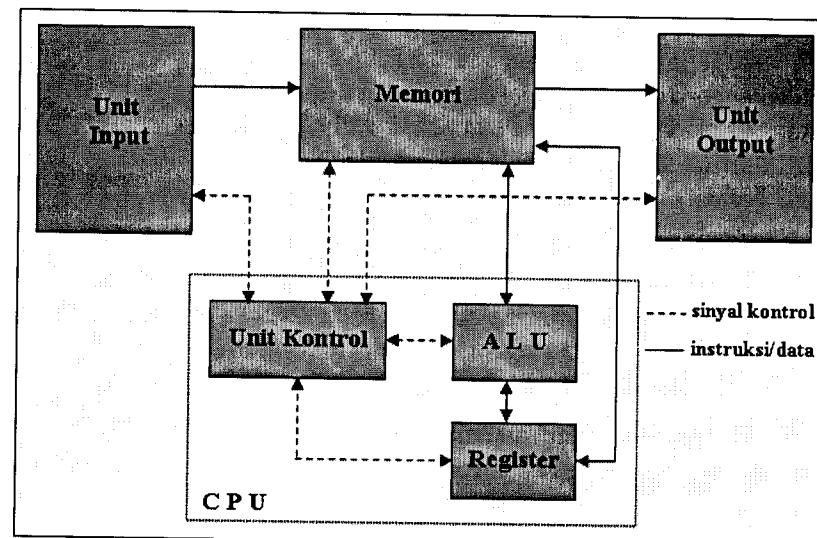
1. Set instruksi
2. Format instruksi
3. Kode operasi
4. Jenis-jenis operand
5. Mode-mode pengalamatan operand
6. Register
7. *Main memory space utilization (memory map)*

8. Alokasi ruang I/O (*I/O map*)
9. Pengerjaan/penetapan interupsi dan prioritas
10. Pengerjaan kanal-kanal DMA dan prioritas
11. Teknik-teknik I/O yang digunakan untuk berbagai perangkat
12. Format-format perintah pengontrol I/O
13. Format-format status pengontrol I/O

Organisasi komputer memberikan gambar yang lebih dalam mengenai struktur fungsional dan interkoneksi logika antara unit-unit (blok fungsional). Biasanya termasuk rincian atau detail hardware yang dapat diketahui oleh pemrogram, seperti sinyal-sinyal kontrol, antarmuka komputer dan peripheral serta teknologi memori yang digunakan.

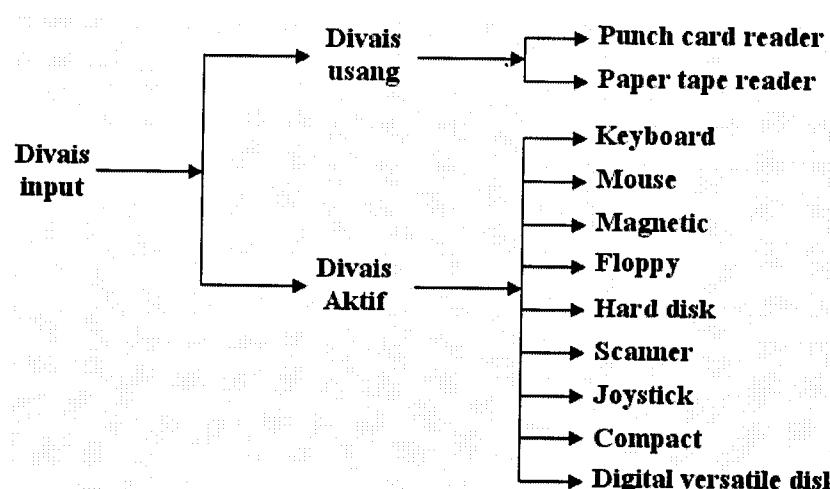
Dua komputer dengan arsitektur yang sama dapat saja mempunyai organisasi yang berbeda, demikian pula sebaliknya. Di dalam perancangan sebuah komputer, yang pertama ditetapkan adalah arsitekturnya, baru kemudian memutuskan organisasinya.

Komputer modern adalah sistem komputer yang terdiri atas hardware dan software. Hardware mempunyai lima macam unit fungsional: memori, ALU (*Arithmetic and Logic Unit*), register, unit kontrol, unit I/O (*Input/output*). Program dan data dimasukkan ke dalam komputer melalui unit input. Memori menyimpan program dan data. Unit kontrol membaca dan menganalisis instruksi satu per satu dan memberikan sinyal kontrol ke seluruh unit untuk melakukan berbagai macam operasi. ALU adalah bagian/unit mesin yang mampu melakukan operasi aritmetika dan logika. Instruksi yang diberikan yang merupakan kumpulan operasi ditunjukkan atau dipandu oleh sinyal kendali yang diterima dari unit kontrol. Hasil dari instruksi disimpan di memori dan dapat dibawa ke unit output.

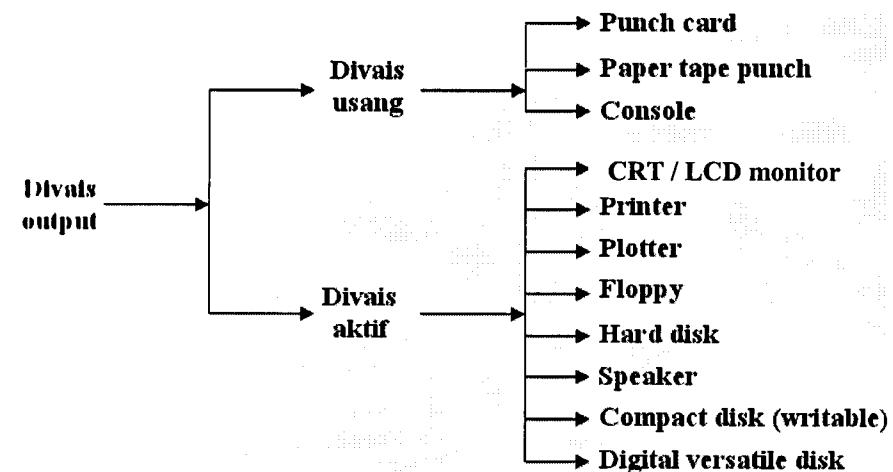


Gambar 1.12 Unit fungsional komputer

Gambar 1.13 dan Gambar 1.14 memperlihatkan beberapa contoh unit masukan dan keluaran. Beberapa di antaranya berfungsi ganda yaitu sebagai masukan dan keluaran.



Gambar 1.13 Peranti-peranti masukan



Gambar 1.14 Peranti-peranti keluaran

1.3.1 Stored Program Concept

Semua komputer modern menggunakan *stored program concept* yang awalnya disusun oleh tim desain komputer ISA yang dipimpin oleh John Von Neumann. Oleh karena itu, biasa disebut konsep Von Neumann.

Stored Program Concept:

"Program bahasa mesin disimpan di dalam komputer beserta data relevan lainnya, dan secara intrinsik komputer mampu memanipulasi program dan data tersebut, misalnya mengambil (load) data/program dari disk ke memori, memindahkannya dari satu lokasi memori ke lokasi memori lainnya, dan menyimpannya kembali ke disk."

Stored program concept pada hakikatnya mengandung:

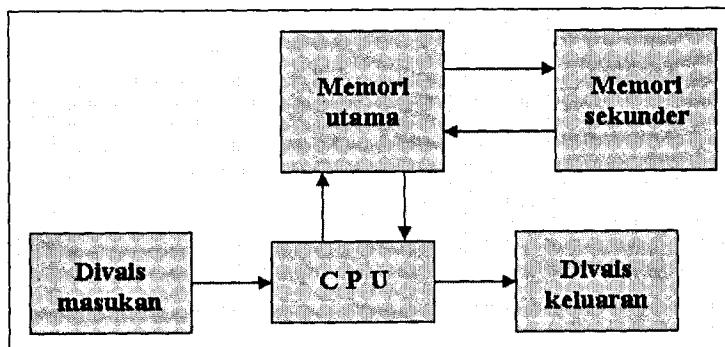
1. Mempunyai 5 unit hardware: memori, ALU, register, unit kontrol, unit I/O.
2. Program dan data disimpan bersama di dalam memori.
3. Pertama kali program berada di memori, kemudian komputer dapat mengeksekusinya secara otomatis tanpa intervensi manual.
4. Unit kontrol mengambil dan mengeksekusi instruksi satu per satu secara sekuensial/berurut. Eksekusi sekuensial tersebut dapat dimodifikasi oleh jenis instruksi tertentu.

5. Suatu instruksi dapat memodifikasi isi dari suatu lokasi memori. Karena itu sebuah program dapat memodifikasi dirinya sendiri.

1.3.2 Memori Utama dan Memori Sekunder

Memori utama adalah wadah penyimpanan utama untuk data, instruksi maupun hasil pengolahan CPU (*central processing unit*). Semua instruksi yang akan dieksekusi oleh CPU diambil langsung melalui memori ini. Oleh karena itu, program harus diletakkan dulu di dalam memori utama.

Memori pembantu (*auxiliary memory*) atau disebut juga memori sekunder merupakan wadah penyimpanan eksternal yang berada di luar inti sistem komputer dan dapat menyimpan data dan program dalam jumlah yang besar. CPU tidak dapat mengambil langsung instruksi dari sebuah program yang berada di dalam memori sekunder. Program yang tersimpan di dalam memori tersebut, apabila hendak dijalankan, harus dibawa terlebih dahulu (*load*) ke dalam memori utama (lihat Gambar 1.15).

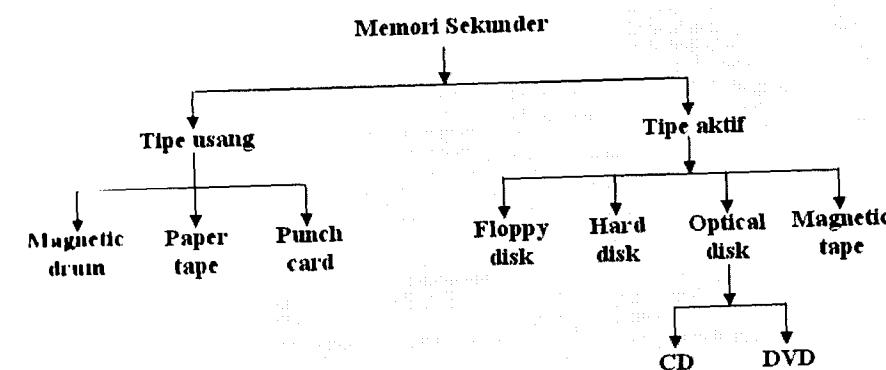


Gambar 1.15 Memori pembantu/sekunder

Memori sekunder relatif lebih murah dibandingkan dengan memori utama sehingga biasanya penggunaan memori utama terbatas kapasitasnya, sementara memori sekunder umumnya kapasitasnya jauh di atas kapasitas memori utama.

Kini memori utama tersedia dalam teknologi semikonduktor, sedangkan memori sekunder menggunakan teknologi magnetik dan optik. Pada Gambar 1.16 dapat dilihat jenis memori sekunder baik yang telah usang

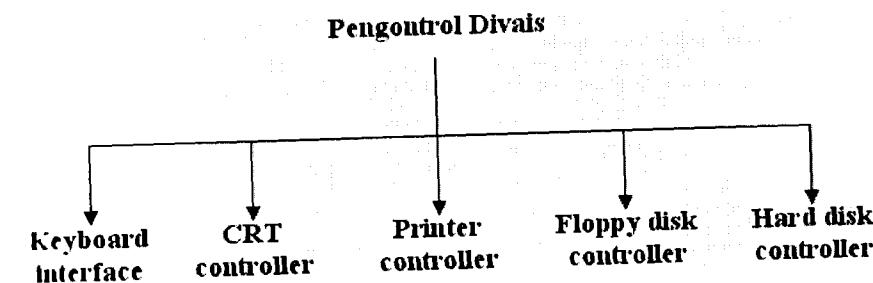
maupun yang masih aktif digunakan saat ini. Memori-memori tersebut dihubungkan ke komputer sebagai peranti I/O.



Gambar 1.16 Jenis-jenis memori sekunder/pembantu

1.3.3 Pengontrol Perangkat

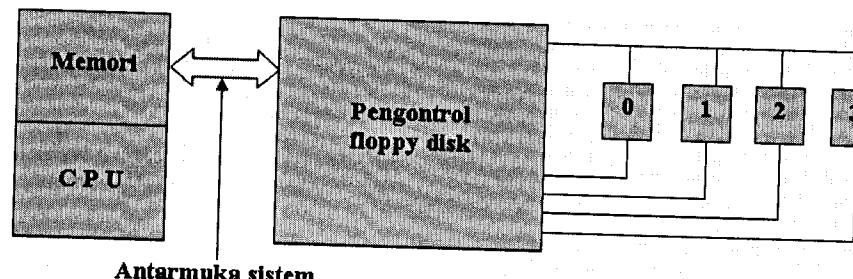
Sebuah perangkat peripheral terhubung (*link*) dengan inti sistem (CPU dan memori) oleh suatu pengontrol perangkat/komponen, biasa juga disebut pengontrol I/O (*I/O controller*). Pada Gambar 1.17. ditunjukkan beberapa pengontrol perangkat.



Gambar 1.17 Beberapa pengontrol perangkat yang lazim

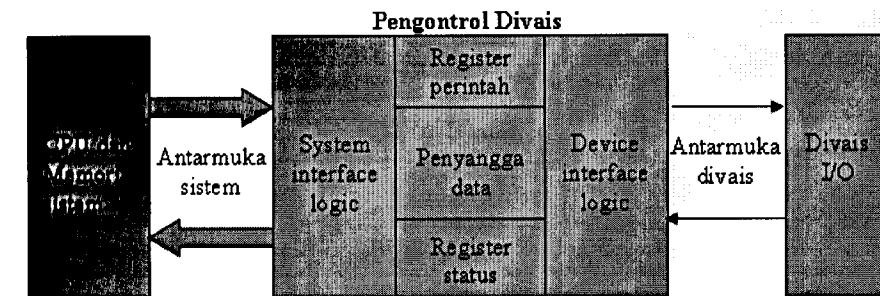
Fungsi utama pengontrol perangkat adalah mentransfer informasi (program dan data) antara inti sistem dengan perangkat. Pengontrol perangkat dapat menangani lebih dari satu perangkat dalam jenis yang

sama. Misalnya sebuah pengontrol floppy disk umumnya menangani empat floppy disk drive seperti yang ditunjukkan pada Gambar 1.18. Secara fisik keberadaan pengontrol perangkat dapat dibedakan dalam tiga macam yaitu (1) sebagai unit yang terpisah, (2) terintegrasi dengan perangkat dan (3) terintegrasi dengan CPU.

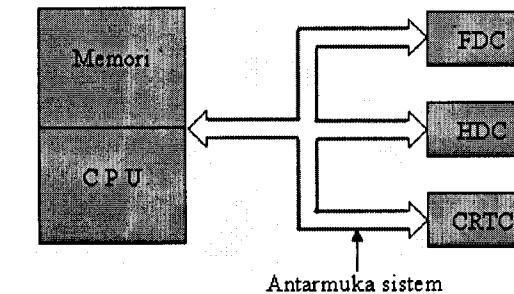


Gambar 1.18 Pengontrol perangkat dan perangkat jamak

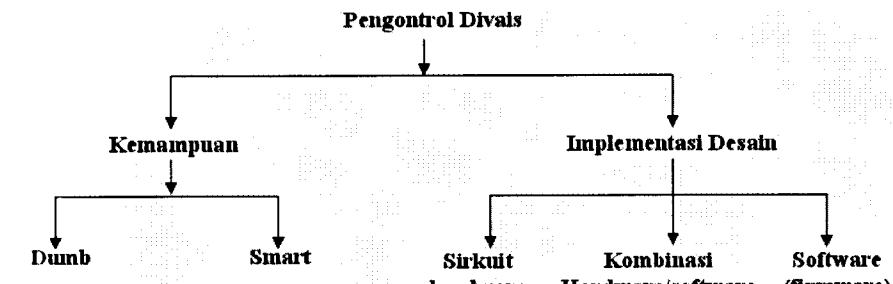
Pengontrol perangkat yang dasar mempunyai lima bagian seperti yang ditunjukkan Gambar 1.19. Pengontrol perangkat berkomunikasi dengan perangkat melalui antarmuka perangkat/komponen yang membawa sinyal antara pengontrol perangkat dengan perangkat. Semua pengontrol perangkat berkomunikasi dengan CPU atau memori melalui antarmuka sistem seperti yang ditunjukkan pada Gambar 1.20. Walaupun beberapa pengontrol perangkat terhubung ke antarmuka sistem, namun secara logika hanya ada satu yang terhubung ke antarmuka sistem pada saat melakukan komunikasi dengan CPU/memori. Pengontrol lainnya tetap tidak melakukan interferensi/gangguan ketika ada sebuah pengontrol yang sedang melakukan komunikasi dengan CPU/memori. Umumnya setiap pengontrol perangkat diberikan nomor sebagai alamat untuk keperluan identifikasi. Register perintah (*command register*) menyimpan perintah yang diberikan oleh software (CPU). Penyangga data menyimpan sementara data yang akan ditransfer pada perangkat. Register status menyimpan status dari perangkat dan pengontrol. Pada Gambar 1.21 ditunjukkan jenis-jenis pengontrol perangkat.



Gambar 1.19 Antarmuka perangkat dan antarmuka sistem



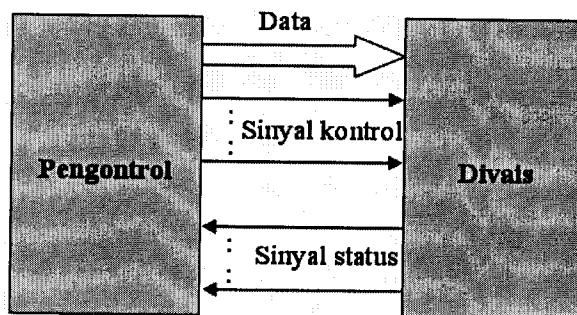
Gambar 1.20 Antarmuka sistem yang lazim



Gambar 1.21 Jenis-jenis pengontrol perangkat

1.3.4 Sinyal Antarmuka Perangkat

Ada tiga macam sinyal antara perangkat dengan pengontrol perangkat (Gambar 1.22): sinyal kontrol, sinyal status, dan sinyal data.



Gambar 1.22 Sinyal-sinyal antarmuka perangkat

1. Sinyal kontrol diberikan oleh pengontrol perangkat kepada perangkat yang meminta perangkat agar melakukan suatu tugas tertentu. Misalnya:
 - a) Sinyal kontrol *RESET* yang berfungsi untuk me-reset, atau clear kondisi internal pada perangkat tersebut.
 - b) Sinyal kontrol *STEP* yang diberikan pada disk drive untuk melakukan pergerakan *head baca/tulis* untuk pindah ke *track* berikutnya.
2. Sinyal status adalah sinyal hasil respons balik dari perangkat I/O yang dikirim ke pengontrol perangkat yang melaporkan kondisi atau status internal tertentu yang dialami atau yang terjadi pada perangkat I/O. Misalnya:
 - a) Sinyal status *ERROR* yang melaporkan bahwa telah terjadi error (kesalahan) di dalam perangkat I/O.
 - b) Sinyal status *PAPER EMPTY* oleh printer, yang melaporkan ke pengontrol printer bahwa tidak ada kertas di dalam printer.
3. Sinyal data dapat dikirim secara serial melalui sebuah konduktor bit per bit atau secara paralel melalui 8 buah konduktor yang membawa 8 bit atau 1 byte data sekaligus. Misalnya:
 - a) Data serial dari mouse
 - b) Data paralel dari printer

1.3.5 I/O Driver

I/O driver untuk suatu perangkat mempunyai rutin dalam melakukan berbagai operasi, misalnya membaca, menulis dan sebagainya. Setiap rutin

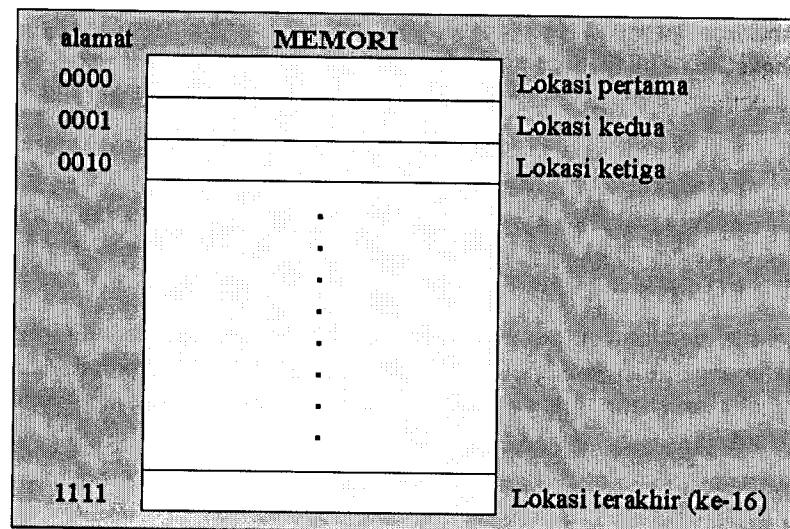
memberikan perintah yang sesuai ke pengontrol I/O agar pengontrol perangkat memberikan sinyal kontrol yang diperlukan ke perangkat. Untuk mendukung perangkat dalam suatu sistem, ada tiga hal yang diperlukan:

1. Pengontrol perangkat yang secara logika mengantarmuka perangkat ke inti sistem.
2. Kabel antarmuka perangkat yang secara fisik menghubungkan perangkat ke pengontrol perangkat.
3. I/O driver

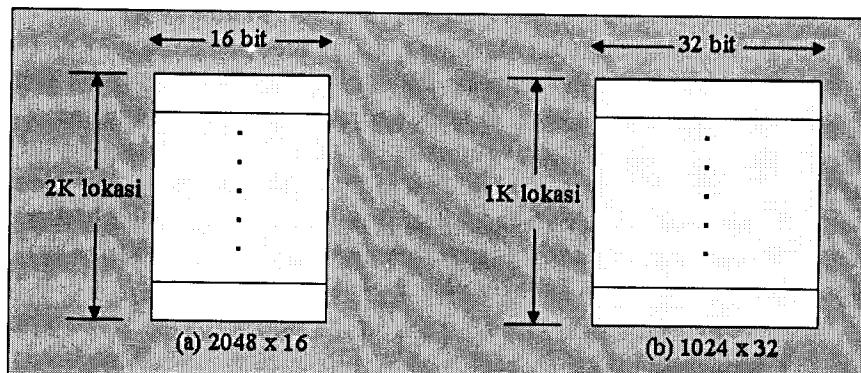
Arsitektur pengontrol perangkat harus dikenalkan kepada pemrogram untuk mengembangkan I/O driver. Sistem operasi dan program lainnya menggunakan rutin pengemudi perangkat untuk melakukan operasi input/output.

1.4 MEMORI UTAMA

Memori menyimpan informasi, data dan hasil dari program yang dieksekusi oleh CPU. Memori ini disebut memori program karena CPU mengambil instruksi hanya dari memori tersebut. Memori utama secara fungsional mengorganisir lokasi. Setiap lokasi menyimpan sejumlah bit tertentu yang tetap. Istilah “*word length*” dari suatu memori menunjukkan jumlah bit pada setiap lokasi. Kapasitas total dari suatu memori adalah jumlah lokasi dikalikan dengan word length. Setiap lokasi di dalam memori mempunyai sebuah alamat yang unik dan dinyatakan dalam notasi biner atau heksadesimal (lihat Gambar 1.23). Dua buah memori yang berbeda dengan kapasitas yang sama dapat memiliki organisasi yang berbeda, seperti yang diilustrasikan pada Gambar 1.24.



Gambar 1.23 Lokasi memori utama



Gambar 1.24 Organisasi dan kapasitas memori

Contoh 1.1

Sebuah komputer mempunyai memori utama dengan jumlah lokasi 1024 dan masing-masing lokasi 32 bit. Hitung kapasitas total memori.

$$\text{Word length} = 32 \text{ bit} = 4 \text{ byte}$$

$$\text{Jumlah lokasi} = 1024 = 1\text{K}$$

$$\text{Kapasitas total memori} = 1\text{K} \times 4 \text{ byte} = 4 \text{ KB}$$

Waktu yang dibutuhkan oleh memori untuk membaca (atau menulis) pada sebuah lokasi memori disebut waktu akses. Umumnya, memori utama akan mempunyai waktu akses yang sama pausnya). Komputer modern menggunakan memori semikonduktor yang mempunyai waktu akses sekitar 10 ns. Komputer terdahulu menggunakan memori magnetik dengan waktu akses sekitar 1 ms. Settling time untuk memori semikonduktor dan memori magnetik masing-masing berada dalam orde 10 dan 200 ns. Waktu siklus adalah waktu minimum yang intervalnya mulai dari awal satu akses ke awal akses berikutnya. Jadi merupakan penjumlahan waktu akses dan waktu settling.

Memori yang mempunyai waktu akses yang sama untuk semua lokasi di dalam memori tersebut dinamakan memori akses acak (*random access memory*). Memori semikonduktor adalah contoh memori akses acak. Walaupun memori magnetik sudah usang, namun merupakan memori yang bersifat *non-volatile* yang penting. Memori RAM adalah memori yang bersifat *volatile*, artinya jika tidak mendapat catu daya atau catu dayanya mati maka isinya akan hilang.

Contoh 1.2

Sebuah memori utama mempunyai waktu akses 45 ns. Di samping itu juga mempunyai time gap 5 ns yang diperlukan untuk penyelesaian satu akses ke awal akses berikutnya. Hitung bandwidth memori tersebut.

$$\text{Access time} = 45 \text{ ns}; \text{settling time} = 5 \text{ ns}$$

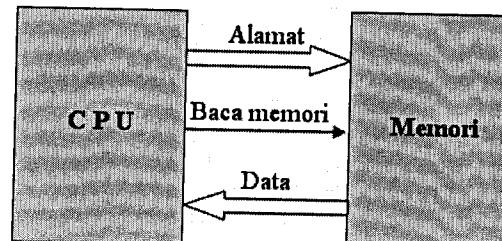
$$\text{Cycle time} = \text{access time} + \text{settling time} = 45 \text{ ns} + 5 \text{ ns} = 50 \text{ ns}$$

$$\text{Bandwidth} = 1/\text{cycle time} = 1/50 \text{ ns} = 20 \text{ MHz}.$$

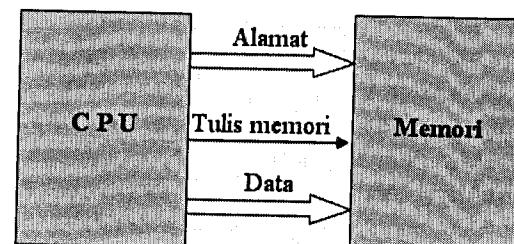
Pada komputer modern, biasanya sebagian kecil ruang memori utama dialokasikan untuk ROM (*Read Only Memory*). CPU dapat membaca ROM tetapi tidak dapat menulisnya. Umumnya diperlukan sejumlah program kontrol dan BIOS (*Basic Input Output System*) yang disimpan secara permanen dalam ROM oleh pabrik. Dalam sistem yang khusus misalnya *embedded system* diperlukan sistem operasi sendiri yang secara permanen disimpan dalam ROM. CPU tidak dapat membedakan antara ROM dan *Read Write Memory* (RWM). Hanya software yang mengetahui peta memori dalam penentuan rentang alamat untuk ROM dan RWM. Pada praktisnya RWM disebut RAM walaupun secara teknis tidak benar.

1.4.1 Komunikasi CPU – Memori

CPU melakukan pengalaman atau mengalami memori baik saat operasi pembacaan memori maupun saat operasi penulisan memori. Selama operasi pembacaan memori, pertama CPU mengirimkan alamat lokasi dan kemudian mengirimkan sinyal baca. Pada saat menerima sinyal baca, memori mulai melakukan pembacaan dari lokasi yang ditunjuk oleh alamat. Setelah melalui waktu akses, isi dari lokasi memori diletakkan oleh memori pada saluran data (Gambar 1.25). Selama operasi penulisan memori, CPU terlebih dahulu mengirimkan alamat lokasi dan kemudian mengirim data yang akan ditulisi dan sinyal tulis memori (Gambar 1.26). Pada saat menerima sinyal tulis memori, memori mulai melakukan penulisan pada lokasi sesuai alamat yang ditetapkan. Sampai waktu akses selesai, memori sibuk melakukan operasi penulisan. CPU menggunakan dua buah register untuk berkomunikasi dengan memori (Gambar 1.27). Selama operasi baca/tulis, CPU meletakkan alamat memori pada register alamat memori (*memory address register*, MAR). Register Penyangga Memori (*memory buffer register*, MBR) digunakan untuk menyimpan data dari CPU selama operasi tulis dan data dari memori selama operasi baca.



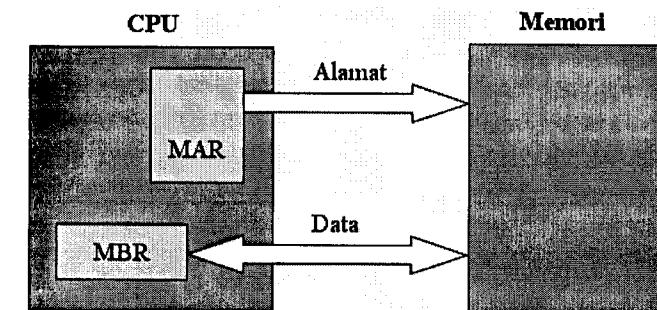
Gambar 1.25 Operasi pembacaan memori



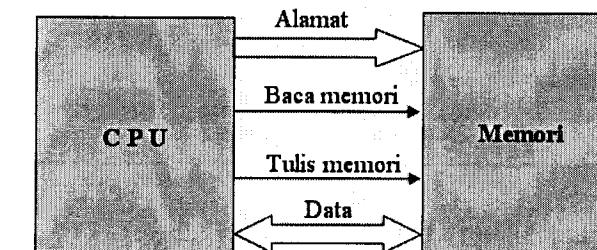
Gambar 1.26 Operasi penulisan memori

Ada dua jenis antarmuka CPU – Memori yang diimplementasikan dalam sistem komputer:

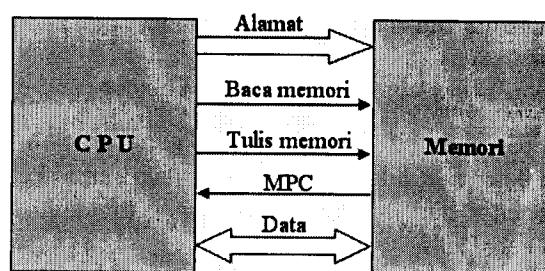
1. Antarmuka sinkron (*synchronous interface*). Pada antarmuka sinkron (Gambar 1.28), waktu yang digunakan oleh memori untuk operasi baca/tulis adalah tetap dan selalu sama. Karena itu, tidak ada feedback dari memori ke CPU yang menunjukkan selesainya operasi baca/tulis.
2. Antarmuka asinkron (*asynchronous interface*). Pada antarmuka asinkron (Gambar 1.29), memori memberitahukan selesainya operasi penulisan dengan mengirimkan sinyal status, *MEMORY FUNCTION COMPLETE* (MFC). Sinyal ini juga disebut *MEMORY OPERATION COMPLETE*.



Gambar 1.27 Register dalam CPU untuk pengaksesan memori



Gambar 1.28 Antarmuka memori sinkron



Gambar 1.29 Antarmuka memori asinkron

1.4.2 Adresibilitas Memori

Jumlah bit alamat memori menentukan jumlah maksimum lokasi memori yang dapat diakses oleh CPU. Anggap CPU mempunyai n bit alamat, maka lokasi memori maksimum yang dapat diakses oleh CPU adalah 2^n lokasi. Ini disebut *CPU's memory addressability* (kemampuan CPU mengalami memori).

Contoh 1.3

Sebuah CPU mempunyai saluran alamat 12 bit untuk mengalami memori. (a) Berapa besar adresibilitas memori dari CPU tersebut. (b) Jika memori mempunyai kapasitas total 16 KB, berapakah word length dari memori.

$$\text{Jumlah bit alamat} = 12$$

$$\text{Adresibilitas Memori} = 2^{12} = 4096 \text{ lokasi} = 4 \text{ K lokasi}$$

$$\text{Kapasitas memori} = 16 \text{ KB}$$

$$\begin{aligned} \text{Word length} &= \text{kapasitas memori/jumlah lokasi memori} = 16 \text{ KB}/4\text{K} \\ &= 4 \text{ B} \end{aligned}$$

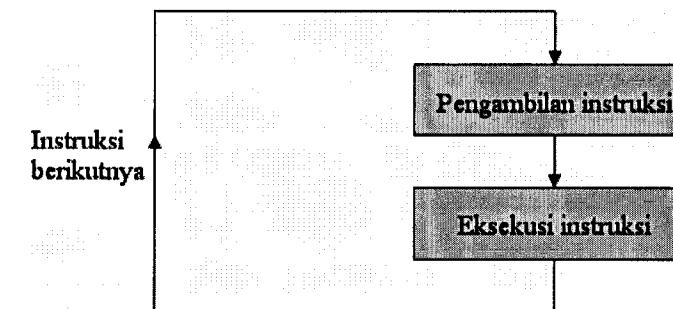
Pada Tabel 1.2 ditunjukkan adresibilitas memori beberapa CPU populer.

TABEL 1.2 Adresibilitas memori beberapa CPU populer

CPU	Jumlah bit alamat	Adresibilitas Memori
IBM System 360/40	24	16 mega
Intel 8080	16	64 kilo
Intel 8088	20	1 mega
Pentium	32	4 giga
Unknown	40	1 tera

1.5 OPERASI CPU

Fungsi CPU adalah melakukan eksekusi program yang tersimpan di memori. Untuk melakukan hal ini, CPU mengambil sebuah instruksi pada satu waktu, mengeksekusinya dan baru kemudian mengambil instruksi berikutnya lagi. Pekerjaan ini dilakukan secara berulang dan dikenal dengan sebutan *siklus instruksi*. Seperti yang terlihat pada Gambar 1.30, siklus instruksi terdiri atas dua fase: yaitu fase pengambilan (*fetch phase*) dan fase eksekusi. Pada fase pengambilan, sebuah instruksi diambil dari memori. Pada fase eksekusi, instruksi dianalisis atau didekode kemudian dilakukan operasi yang relevan.



Gambar 1.30 Fase siklus instruksi

1.5.1 Format Instruksi dan Siklus Instruksi

Format umum sebuah instruksi diperlihatkan pada Gambar 1.31. Medan (*field*) kode operasi atau dikenal dengan field opcode menunjukkan operasi yang dikerjakan dan field operand menunjukkan data.

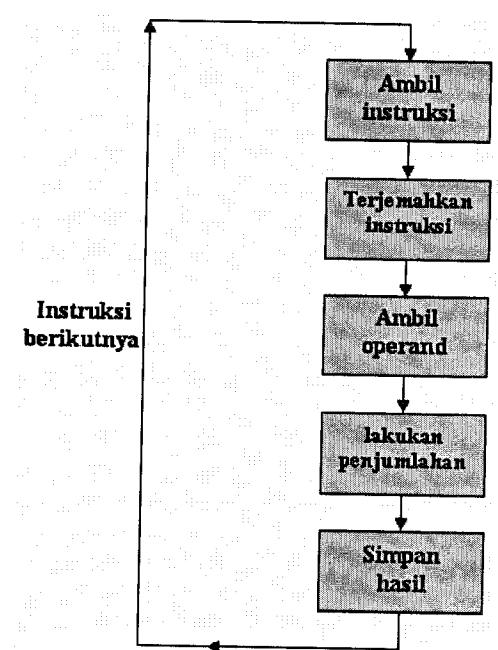


Gambar 1.31 Format instruksi

Umumnya field operand memberikan alamat lokasi memori di mana operand (data) tersebut disimpan. Misalnya instruksi ADD yang mempunyai format seperti yang diperlihatkan Gambar 1.32. Pola bit pada field opcode menunjukkan instruksi ADD. Dua field berikutnya menunjukkan lokasi di mana dua operand (data) tersedia atau tersimpan. Pada Gambar 1.33 diperlihatkan langkah-langkah yang terjadi di dalam siklus instruksi dan Tabel 1.3 menunjukkan arti setiap langkah.



Gambar 1.32 Format instruksi ADD



Gambar 1.33 Langkah-langkah siklus instruksi

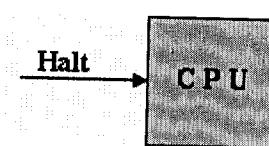
TABEL 1.3 Langkah-langkah siklus instruksi dan aksinya

No. langkah	Langkah	Respons yang dilakukan	Keterangan
1	Ambil instruksi	Unit kontrol; tindakan eksternal	Ambil instruksi berikutnya dari memori utama
2	Penerjemahan Instruksi	Unit kontrol; tindakan internal	Analisis pola opcode pada instruksi dan identifikasi operasi yang tepat sesuai yang ditetapkan
3	Ambil operand	Unit kontrol; eksternal (memori) atau tindakan internal bergantung pada lokasi operand	Ambil operand, satu per satu, dari memori utama atau register CPU dan suplaikan ke ALU
4	Eksekusi (ADD)	ALU; tindakan internal	Mengerjakan operasi aritmetika atau logika sesuai yang ditetapkan
5	Simpan hasil	Unit kontrol; tindakan eksternal atau internal	Simpan hasil di memori atau register

1.5.2 CPU State

CPU mempunyai dua keadaan operasi utama yaitu *running state* dan *halt state*. Pada *running state*, CPU melakukan siklus instruksi. Pada *halt state*, CPU tidak melakukan siklus instruksi. Ketika komputer dihidupkan, maka CPU segera melakukan *running state*. CPU beralih dari *running state* ke *halt state* apabila ada aksi berikut:

1. **Software halt.** Sebuah instruksi halt diambil dan akan dieksekusi oleh CPU, karena itu CPU dalam keadaan berhenti.
2. **Halt Input.** CPU menerima masukan sinyal halt dan berhenti. Sinyal ini dapat berasal dari penekanan tombol *HALT* oleh operator atau saklar *PAUSE* pada panel depan atau oleh sirkuit eksternal ke CPU.



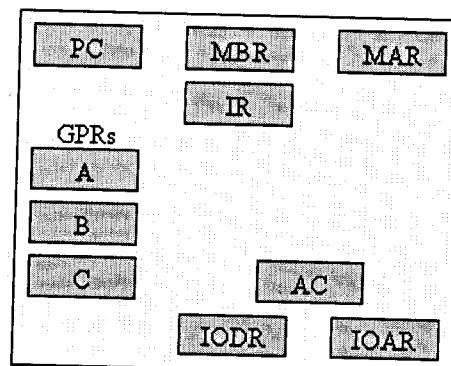
Gambar 1.34 Sinyal masukan halt

3. **Auto Shutdown.** Selama *instruction cycle*, CPU mengalami situasi abnormal yang serius dan karena itu terjadi shutdown yang menghentikan *instruction cycle*. (Shutdown ini berbeda dengan *system shutdown*.) Kemampuan ini mungkin hanya terdapat pada *powerful CPU's*. Jika kemampuan "*shutdown*" tidak tersedia dalam CPU, maka bisa terjadi "*hang*", "*loop*" atau menjadi tidak beres pada situasi abnormal yang tidak terprediksi.

1.5.3 Register-Register CPU

CPU mempunyai register utama seperti berikut (lihat Gambar 1.35):

1. *Accumulator (AC)*
2. *Program Counter (PC)*
3. *Memory Address Register (MAR)*
4. *Memory Buffer Register (MBR)*
5. *Instruction Register (IR)*
6. *General Purpose Register (GPR)*
7. *I/O Data Register (IODR)*
8. *I/O Address Register (IOAR)*

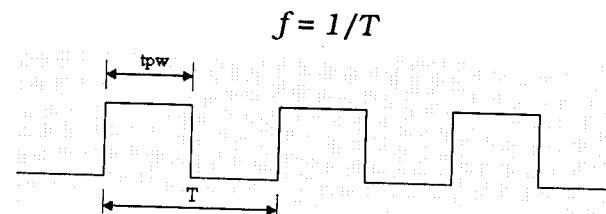


Gambar 1.35 Register utama CPU

1. Register akumulator (*accumulator*) menyimpan hasil operasi sebelumnya yang ada dalam ALU, juga digunakan sebagai register masukan ke penjumlahah.
2. *Program Counter (instruction address counter)* berisi alamat lokasi memori di mana instruksi berikut harus diambil. Segera setelah pengambilan instruksi telah lengkap, isi PC (*program counter*) dinaikkan untuk menunjuk ke alamat instruksi berikutnya.
3. *Instruction register* menyimpan instruksi yang baru saja diambil dari memori untuk dilakukan pendekodean/penerjemahan instruksi.
4. MAR berisi alamat lokasi memori selama memori dalam operasi baca/tulis.
5. MBR berisi data yang dibaca dari memori (selama pembacaan) atau data yang akan ditulis ke dalam memori (selama penulisan).
6. GPR (*general purpose register*) digunakan untuk keperluan umum misalnya menyimpan *operand*, alamat, dan seterusnya. Selain itu CPU juga mempunyai beberapa register kerja yang disebut *scratch pad memory* (memori untuk keperluan corat-coret). Register ini digunakan untuk menjaga/menyimpan hasil-hasil sementara dalam satu siklus instruksi untuk instruksi yang kompleks seperti PERKALIAN, PEMBAGIAN, dan seterusnya.

1.5.4 Clock

Unit clock membangkitkan dan mensuplai pulsa clock secara berurutan dan kontinu. Sinyal clock mempunyai bentuk gelombang yang periodik. Sinyal clock digunakan sebagai referensi waktuan oleh unit kontrol. Jumlah gelombang periodik yang berulang dalam satuan waktu disebut frekuensi (*f*). Satuan frekuensi ditetapkan dalam *cycle per second* (*cps* = siklus per detik) atau Hz. Frekuensi clock mengindikasikan kecepatan operasi internal dari prosesor. Interval waktu antara sinyal periodik dengan sinyal periodik berikutnya disebut periode waktu (*T*). Hubungan antara frekuensi dan periode adalah



Gambar 1.36 Bentuk gelombang periodik

Lebar pulsa (*pulse width*, t_{pw}) menunjukkan durasi pulsa clock. Istilah lain yang berhubungan dengan lebar pulsa dikenal dengan istilah *duty cycle*, di mana *duty cycle* adalah rasio (dinyatakan dalam persen) lebar pulsa terhadap periode. Ilustrasi mengenai periode waktu dan lebar pulsa dapat dilihat pada Gambar 1.36.

$$\text{Duty cycle} = \left(\frac{t_{pw}}{T} \right) \times 100\%$$

Contoh 1.4

Suatu sinyal clock mempunyai frekuensi 10 MHz dengan duty cycle 50%. Hitung periode waktu dan lebar pulsa clock tersebut.

$$\text{Periode waktu, } T = 1/f = 1/(10 \times 10^6) = 100 \text{ ns}$$

$$\text{Duty cycle} = \left(\frac{t_{pw}}{T} \right) \times 100\%$$

$$\text{Lebar pulsa, } t_{pw} = 0,5 T = 50 \text{ ns}$$

1.5.5 Operasi Makro dan Operasi Mikro

Setiap instruksi menetapkan operasi-makro yang akan dikerjakan oleh CPU. Opcode merupakan makro-operasi yang eksak. Set instruksi menyimpan semua operasi-makro untuk sebuah komputer. Sebuah program bahasa mesin juga disebut *macro-program*. Untuk mengeksekusi suatu instruksi maka CPU harus melakukan beberapa operasi-mikro. Operasi-mikro merupakan operasi dasar di dalam komputer. Berikut beberapa contoh operasi-mikro:

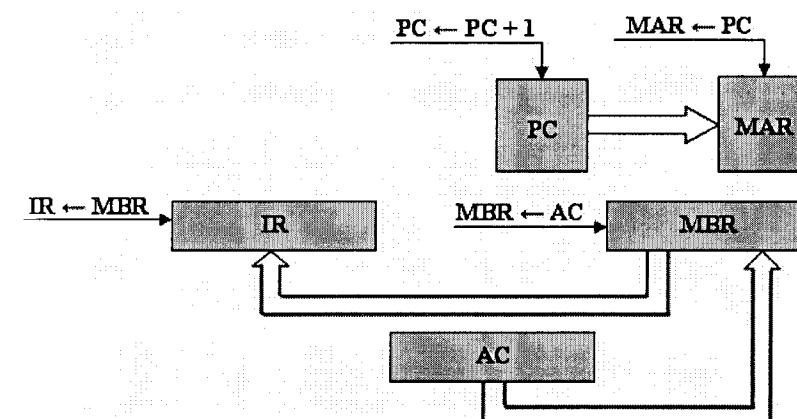
- membersihkan register
- menambah isi pencacah
- menjumlah dua buah input
- mentransfer isi register ke register lain
- men-setting sebuah flip flop

- meng-komplemen-kan isi register
- melakukan pergeseran isi register
- membaca memori dan menulis memori

Setiap operasi-mikro akan dikerjakan bila sinyal kontrol yang sesuai diberikan oleh unit kontrol. Tabel 1.4 memberikan beberapa contoh operasi-mikro. Gambar 1.37 mengilustrasikan beberapa titik kontrol.

TABEL 1.4 Contoh operasi-mikro

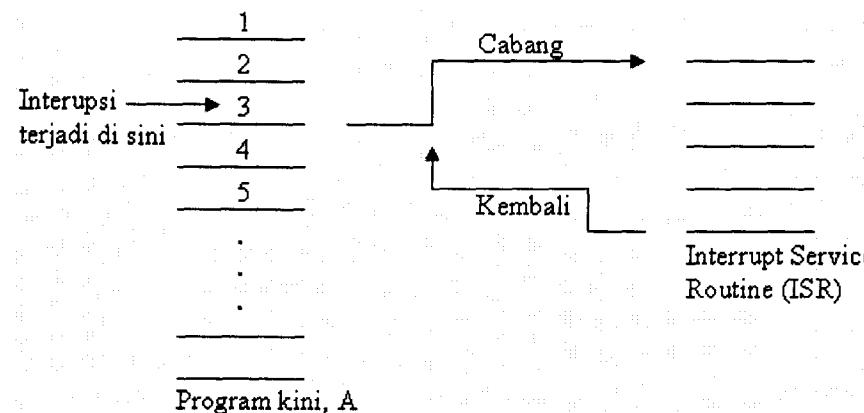
No	Sinyal kontrol	Operasi-mikro	Keterangan
1	$\text{MAR} \leftarrow \text{PC}$	Isi PC disalin (transfer) ke MAR	Operasi-mikro pertama pada pengambilan instruksi
2	$\text{PC} \leftarrow \text{PC} + 1$	Isi PC dinaikkan	PC selalu menunjuk ke alamat instruksi berikutnya
3	$\text{IR} \leftarrow \text{MBR}$	Isi MBR disalin ke IR	Operasi-mikro terakhir pada pengambilan instruksi
4	$\text{MBR} \leftarrow \text{AC}$	Isi akumulator disalin ke MBR	Operasi-mikro pertama pada penyimpanan hasil



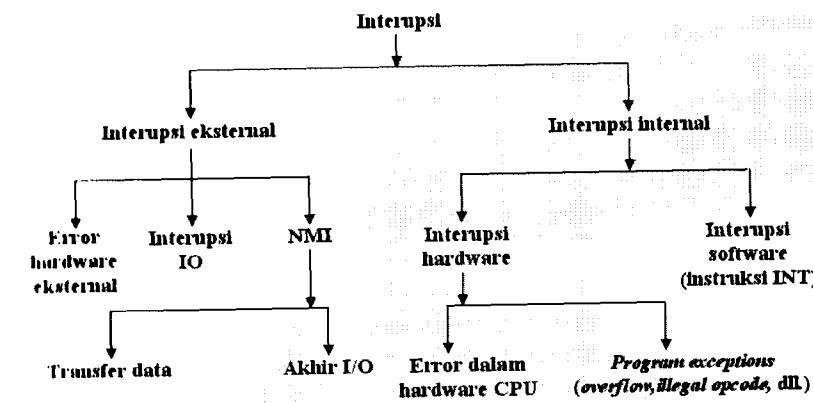
Gambar 1.37 Contoh titik-titik kontrol

1.6 KONSEP INTERUPSI

Interupsi adalah suatu kejadian dalam sistem komputer yang meminta pelayanan penting kepada CPU ketika CPU sedang melakukan pemrosesan. Dalam merespons interupsi, CPU menunda eksekusi program saat itu dan melakukan pencabangan ke rutin layanan interupsi (*interrupt service routine*, ISR). ISR adalah sebuah program yang melayani interupsi dan dilakukan dengan tepat. Setelah eksekusi ISR, CPU kembali lagi ke program yang menginterupsi. Ini artinya CPU akan melanjutkan program yang tersisa ketika sebelum terjadi pencabangan. Untuk itu, kondisi (status) CPU sebelum melayani ISR harus diingat (disimpan). Ketika akan kembali dari ISR, status yang tersimpan diambil kembali ke CPU. Gambar 1.38 mengilustrasikan konsep interupsi.

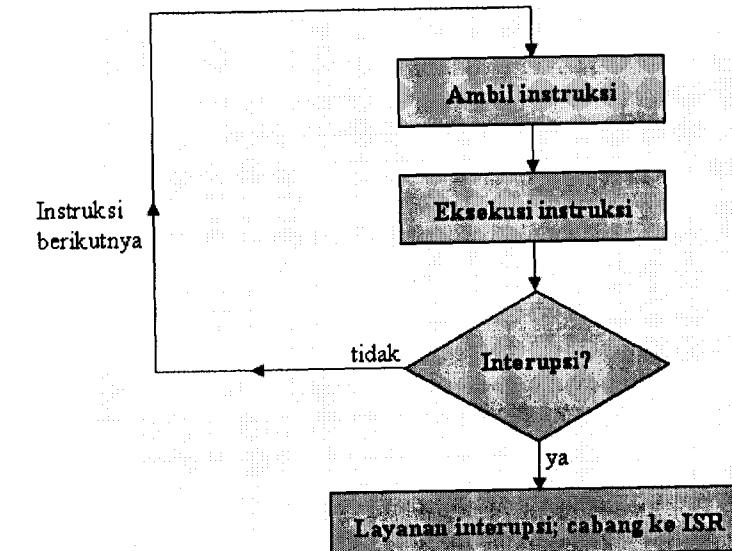


Gambar 1.38 Konsep interupsi



Gambar 1.39 Jenis-jenis interupsi

Ada beberapa macam interupsi seperti yang dapat dilihat pada Gambar 1.39. Tabel 1.5 memberikan penjelasan penyebab interupsi. Interupsi dapat terjadi kapan saja. Tetapi, umumnya CPU dapat mengenalinya hanya ketika siklus instruksi telah lengkap/selesai dan akan memulai siklus instruksi lainnya. Gambar 1.40 memperlihatkan siklus instruksi yang disertai dengan pengenalan atau pengecekan interupsi.



Gambar 1.40 Siklus instruksi dan interupsi

TABEL 1.5 Penyebab interupsi yang lazim dan tindakan yang dilakukan

No	Jenis Interupsi	Penyebab Interupsi	Tindakan ISR
1	Penyelesaian I/O	Pengontrol perangkat melaporkan bahwa interupsi telah diselesaikan oleh operasi I/O yaitu, komando berakhir	Kontrol diberikan ke I/O driver sehingga dia dapat melakukan verifikasi status penyelesaian (sukses atau tidak sukses)
2	Transfer data	Pengontrol perangkat melaporkan bahwa dia mempunyai data byte yang siap (untuk operasi input) atau dia memerlukan data byte (untuk operasi output)	Kontrol diberikan ke I/O driver di mana dilakukan transfer data oleh instruksi IN atau OUT.
3	Overflow	Hasil suatu operasi melebihi kapasitas akumulator	Kontrol diberikan ke sistem operasi yang dapat membatal-kan program atau menarik perhatian pengguna

Nested Interrupt: Jika CPU diinterupsi ketika sedang mengeksekusi ISR, disebut *interrupt nesting*. Akibatnya CPU mencabang ke ISR yang baru. Kalau ISR yang baru telah diselesaikan maka CPU kembali ke ISR yang menginterupsi (sebelumnya, lama).

Interrupt priority: Ketika yang menginterupsi lebih dari satu pada saat yang sama, maka pada saat itu yang akan dilayani satu saja. Urutan pelayanan interupsi disebut prioritas interupsi. Jadi dengan adanya penginterupsi yang mempunyai prioritas lebih tinggi, mengakibatkan terjadi penundaan terhadap penginterupsi yang prioritasnya lebih rendah.

Interrupt masking: Ada saat di mana CPU secara kontinu menjalankan program tanpa interupsi. Untuk hal tersebut, maka pengenalan interupsi CPU dapat dicegah oleh penghalang CPU dan ini disebut dengan penghalangan interupsi (*interrupt masking*). Bila interupsi dihalangi, interupsi tidak ditarik kembali, namun ditunda. Bila penghalang dilepaskan/hilang, interupsi mengambil tempat. Penghalangan interupsi pada level CPU

dilakukan dengan cara me-reset bendera IE (*Interrupt Enable flag*) pada CPU. Jika IE flag logika 1, CPU merasakan adanya interupsi dan kemudian melayaninya. Jika IE flag logika 0, CPU mengabaikan interupsi. Dua buah Instruksi yaitu EI (*Enable Interrupt*) dan DI (*Disable Interrupt*) mengontrol flag tersebut. Jika EI men-set bendera IE (IE flag) maka interupsi akan *enable* atau tidak terhalang. Jika DI me-reset bendera IE maka interupsi akan *disable* atau terhalang.

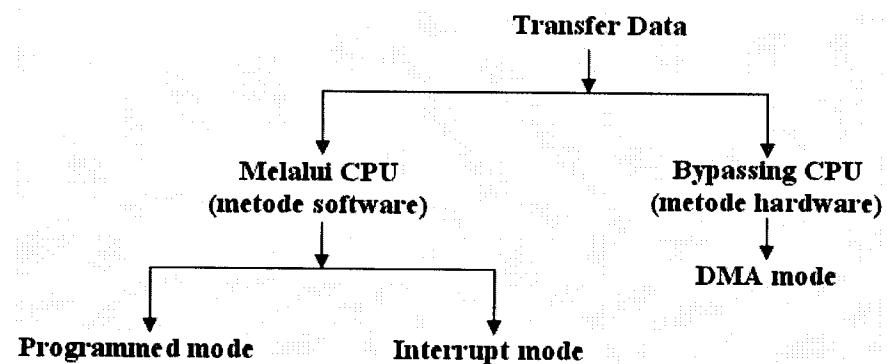
Non-Maskable Interrupt (NMI): Interupsi tertentu harus dilayani tanpa ada penundaan. Dengan kata lain, dapat menyebabkan kerusakan pada program atau data atau hasil. Interupsi ini disebut *non-maskable interrupt* dan CPU tidak dapat menghalanginya. Pada Tabel 1.6 diberikan beberapa penyebab NMI. Ketika CPU melayani NMI maka akan terlepas dari kondisi IE flag ('1' atau '0').

TABEL 1.6 Penyebab NMI dan tindakan yang dilakukan

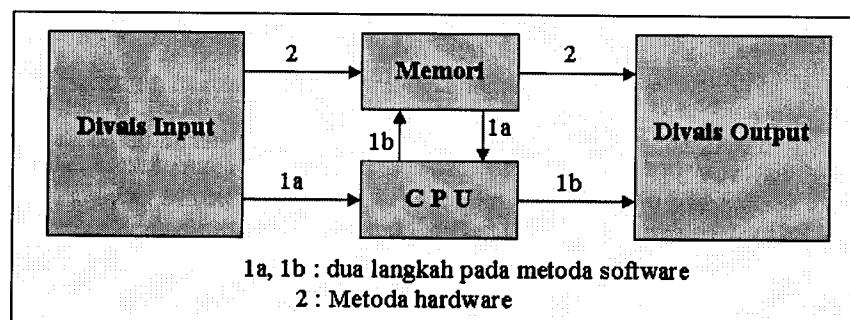
No	Penyebab NMI	Definisi	Tindakan ISR
1	<i>Power fail</i>	Peringatan lanjut bahwa catu daya AC mungkin mati	Kontrol dapat diberikan ke rutin <i>SAVE</i> yang menyimpan status CPU. Jika terdapat daya cadangan (UPS dsb.), transfer sumber daya dilaksanakan.
2	<i>Memory parity error</i>	Saat dibaca dari memori, dia mendeteksi bahwa sejumlah bit data telah mengalami kegagalan	Kontrol diberikan ke rutin <i>ERROR RECOVERY</i> yang dapat membatalkan program atau menanyakan respons operator.
3	<i>Bus cycle malfunction</i>	Siklus bus sekarang telah terjadi beberapa malfungsi	Kontrol diberikan ke rutin <i>MACHINE CHECK</i> yang mengambil keputusan apakah mencoba lagi atau membatalkan dengan suatu pesan.

1.7 TEKNIK-TEKNIK I/O

Perangkat I/O membantu kita berinteraksi dengan komputer agar dapat memberi/menerima data, program dan hasil ke/dari komputer. Ketika dilakukan operasi input, kita memindahkan informasi dari perangkat masukan ke memori (atau CPU). Demikian halnya pada operasi output, informasi dipindahkan dari memori (atau CPU) ke perangkat keluaran. Rutin I/O (program) bertugas menjaga/mengawasi operasi I/O. I/O driver adalah kumpulan dari rutin I/O untuk berbagai operasi pada perangkat I/O yang spesifik. I/O driver mengawasi pemberian perintah/komando ke pengontrol perangkat, melakukan verifikasi status dari pengontrol perangkat dan perangkat serta menangani operasi I/O. I/O driver adalah sebuah program sistem. I/O driver dari semua perangkat I/O dikenal sebagai BIOS.



Gambar 1.41(a) Metode transfer data

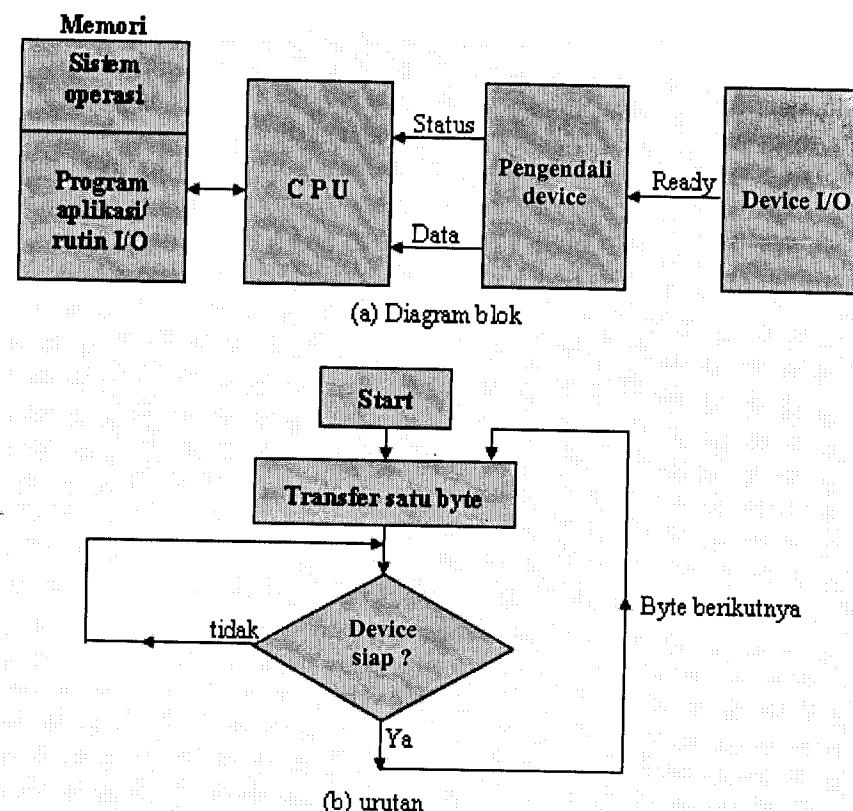


Gambar 1.41(b) Prinsip-prinsip transfer data

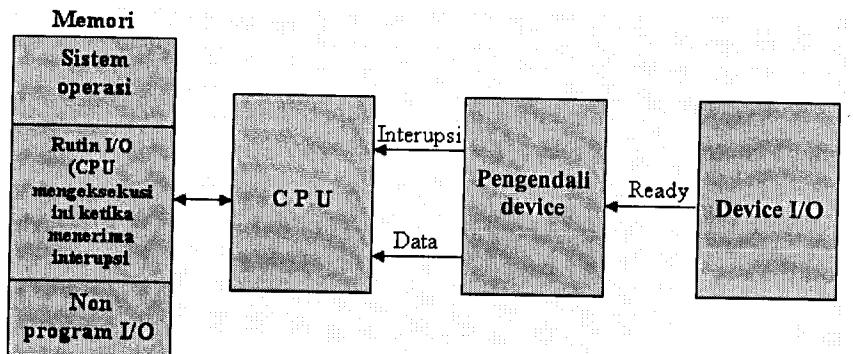
Rutin I/O dapat mengikuti tiga metode yang berbeda untuk pelaksanaan transfer data seperti yang ditunjukkan pada Gambar 1.41(a). Pada metode software, rutin I/O secara fisik mentransfer setiap penggalan data (byte) melalui CPU dalam dua langkah seperti yang ditunjukkan pada Gambar 1.41(b). Instruksi IN atau OUT digunakan untuk mentransfer data antara CPU dan perangkat. Karena sumber dan tujuan data yang sangat besar ada di dalam memori, instruksi IN atau OUT digunakan bersama dengan instruksi STORE atau LOAD untuk mentransfer setiap byte (word) data.

I/O Port: I/O Port adalah unit yang dapat dialami program di mana data dapat disuplai ke CPU atau dapat menerima data yang diberikan oleh CPU. *Input port* mensuplai data bila teralamati, sedangkan *output port* menerima data bila teralamati. Instruksi IN digunakan untuk menerima data dari input port. Instruksi OUT digunakan untuk mensuplai data ke output port. Setiap port memiliki sebuah alamat unik seperti halnya dengan lokasi memori yang juga mempunyai alamat unik. Secara fisik keberadaan sebuah port dapat dalam berbagai bentuk: sebagai unit tersendiri, bersama dengan port lainnya, atau bagian dari sebuah komponen fungsi khusus seperti pengontrol interupsi atau pengontrol perangkat, dan sebagainya. Secara fungsional, sebuah port berfungsi sebagai sebuah jendela untuk komunikasi dengan CPU oleh antarmuka port melalui bus data. Misalnya, posisi (ON/OFF) dari delapan buah saklar yang dapat dirasakan melalui delapan bit input port. Instruksi IN dapat men-transfer status ini ke sebuah register CPU. Demikian pula delapan buah LED dapat dihubungkan ke output port melalui antarmuka. Instruksi OUT dapat mensuplai pola yang diinginkan dengan mengirimkan isi register CPU ke output port. Konsep penggunaan instruksi IN dan OUT untuk berkomunikasi dengan I/O port dikenal dengan "Direct I/O" atau "I/O mapped I/O". Beberapa CPU memperlakukan I/O port seperti halnya lokasi memori. Pada sistem ini, instruksi IN dan OUT tidak digunakan. Konsep ini disebut "Memory mapped I/O".

Pada *program mode (polling)*, rutin I/O mencurahkan sepenuhnya pada transfer data dari permulaan sampai akhir (byte pertama ke byte terakhir) seperti yang ditunjukkan pada Gambar 1.42. Setelah melakukan transfer satu byte, I/O driver harus menunggu hingga perangkat siap untuk mentransfer byte berikutnya. Jadi, waktu CPU terbuang selama menunggu. Pada *interrupt mode* (Gambar 1.43), CPU men-switch antara rutin I/O dan program lain di antara transfer byte berurutan.

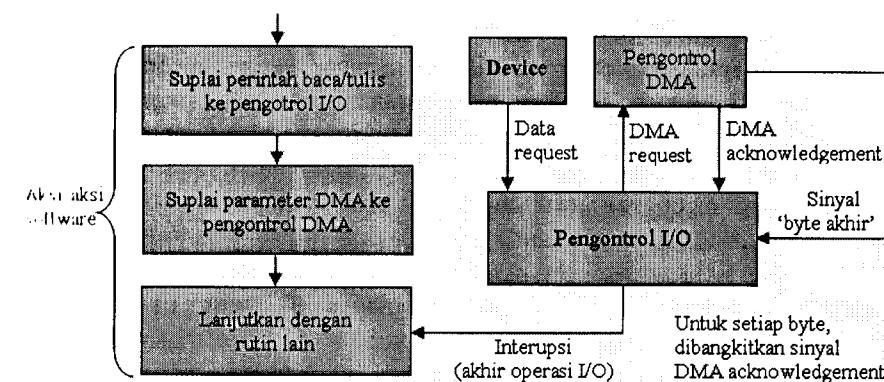


Gambar 1.42 Transfer data pada program mode



Gambar 1.43 Transfer data pada interrupt mode

Pada metode hardware, rutin I/O tidak terlibat dalam transfer data secara aktual. *Dedicated hardware unit* seperti *data channel* atau pengontrol DMA (*Direct Memory Access*) mengawasi pengontrolan transfer data antara memori dan pengontrol I/O (Gambar 1.44). Rutin I/O mengawali suatu urutan transfer data dengan memberikan komando pada pengontrol I/O dan parameter-parameter DMA ke pengontrol DMA. Parameter-parameter DMA termasuk alamat awal memori, jumlah byte dan arah transfer data dengan memori. Transfer data antara memori dan pengontrol I/O dilakukan tanpa pengetahuan CPU. Pada akhir, interupsi dari pengontrol I/O melaporkan selesainya operasi I/O pada I/O driver.



Gambar 1.44 Transfer data DMA

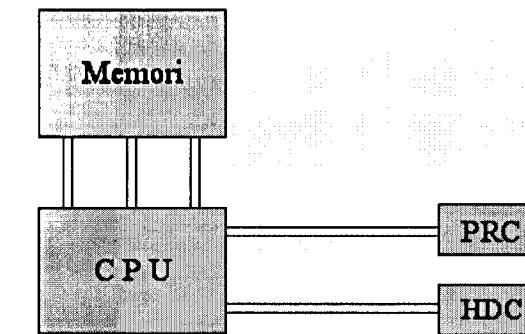
Transfer data dari perangkat kecepatan tinggi seperti hard disk dan floppy disk kesulitan dengan program mode atau interrupt mode karena transfer rate pada mode ini lebih lambat daripada data yang datang dari perangkat tersebut. Karena itu, DMA mode lebih sesuai untuk perangkat kecepatan tinggi. Perangkat kecepatan rendah dapat ditangani oleh program mode atau interrupt mode. Interrupt mode cocok (ideal) untuk perangkat kecepatan rendah agar waktu CPU tidak terbuang di dalam menunggu kesiapan perangkat di antara transfer byte berurutan.

1.8 KONSEP BUS

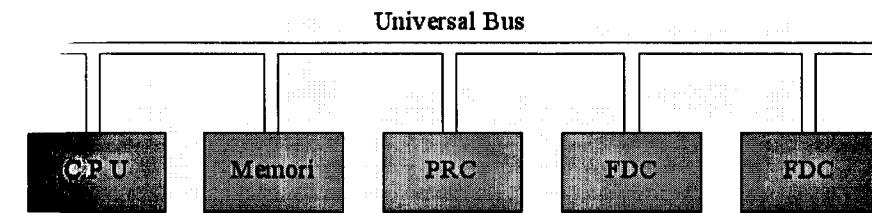
Hubungan bus/jalur diperlukan di dalam komputer untuk membawa berbagai jenis informasi di antara subsistem (CPU, memori, dan pengontrol I/O), antara lain:

1. Instruksi dari memori ke CPU
2. Data dari memori ke CPU
3. Data dari CPU ke memori
4. Alamat memori dari CPU ke memori
5. Alamat port dari CPU ke pengontrol I/O
6. Perintah dari CPU ke pengontrol I/O
7. Status dari pengontrol I/O ke CPU.

Pada komputer mainframe, ada jalur terpisah dari setiap register sumber ke setiap register tujuan (Gambar 1.45) untuk menghubungkan sumber dan tujuan. Setiap titik sumber telah dihubungkan langsung ke setiap titik tujuan. Akibatnya, biaya hardware menjadi sangat besar karena banyaknya perkawatan dan sirkuit driver/receiver. Pada minicomputer dan microcomputer, konsep bus digunakan untuk melakukan interkoneksi sinyal antara subsistem di dalam komputer. Bus merupakan jalur yang digunakan secara bersama untuk sejumlah sumber dan tujuan (Gambar 1.46). Sebuah kawat tunggal digunakan untuk membawa sebuah sinyal ke beberapa unit. Namun demikian, pada satu saat hanya ada dua unit yang terhubung secara logika; satu sumber dan satu tujuan. Misalnya CPU mengirimkan data delapan bit ke pengontrol floppy disk. Delapan bit dikirim pada delapan saluran. Setiap saluran secara keseluruhan dikontrol bersama dan semuanya menerima pola data. Jadi, setiap bit data ada sebuah bus dan karena itu terdapat delapan bus untuk data. Dengan kata lain lebar bus datanya adalah 8-bit. Walaupun semua pengontrol menerima data yang sama, hanya ada satu pengontrol secara logika terhubung ke CPU. Keuntungan utama metoda bus adalah mengurangi biaya perkawatan dan yang berhubungan dengan sirkuit driver/receiver. Kekurangannya adalah kecepatannya rendah karena bus digunakan bersama. Pada satu waktu, hanya ada dua unit yang dapat berkomunikasi, unit lain yang ingin melakukan komunikasi harus menunggu.

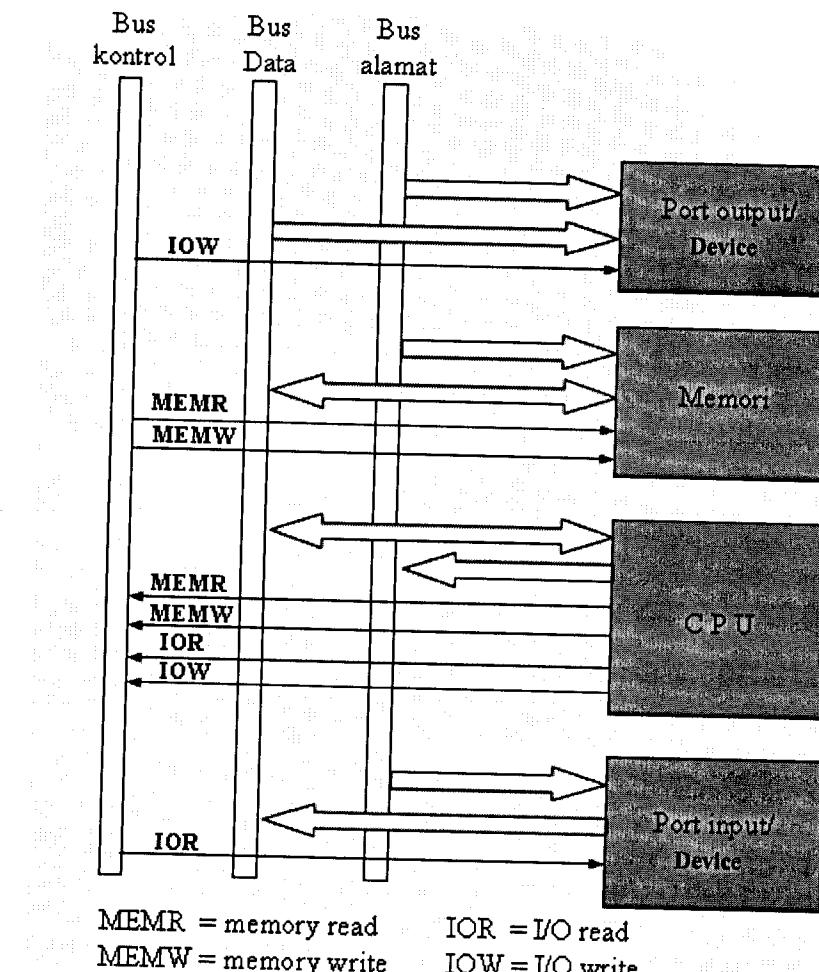


Gambar 1.45 Komunikasi non-bus

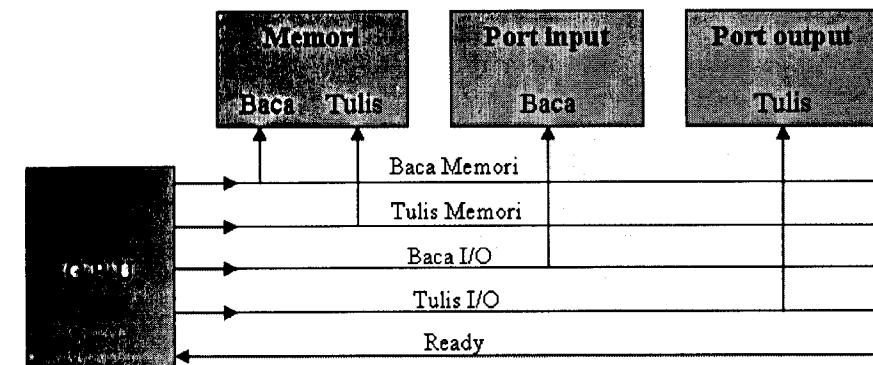


Gambar 1.46 Konsep bus

Pada Gambar 1.47 ditunjukkan struktur bus. Ada tiga macam bus yaitu bus data, bus alamat, dan bus kontrol. Data bus berfungsi untuk menyalurkan data antara CPU dan memori serta unit I/O. Bus alamat digunakan oleh CPU untuk mengirimkan alamat ke memori atau pengontrol I/O. Bus kontrol berfungsi untuk menyalurkan sinyal kontrol ke berbagai unit agar tidak terjadi kekacauan koordinasi dalam sistem komputer, misalnya control bus untuk membaca memori (dari CPU), menulis memori (dari CPU), membaca I/O (dari CPU), menulis I/O (dari CPU), *clock*, *reset* dan *ready*. Pada Gambar 1.48 dan Tabel 1.7 ditunjukkan sejumlah sinyal kontrol.



Gambar 1.47 Struktur bus sederhana



Gambar 1.48 Sinyal-sinyal bus kontrol

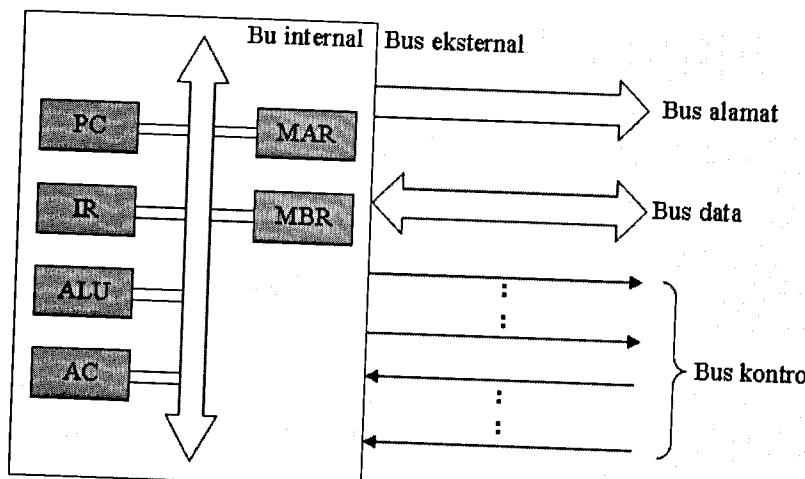
TABEL 1.7 Sinyal-sinyal kontrol

No	Sinyal kontrol	Penjelasan
1	<i>Memory read</i>	Sinyal keluaran dari CPU; mengindikasikan bahwa memori yang teralamati harus meletakkan datanya pada bus data.
2	<i>Memory write</i>	Sinyal keluaran; mengindikasikan bahwa memori yang teralamati harus mengambil data dari bus data.
3	<i>I/O read</i>	Sinyal keluaran dari CPU; mengindikasikan bahwa port masukan (input port) yang terpilih harus meletakkan datanya pada bus data.
4	<i>I/O write</i>	Sinyal keluaran dari CPU; mengindikasikan bahwa port keluaran (output port) yang terpilih harus mengambil data dari bus data.
5	<i>Ready</i>	Sinyal masukan ke CPU; mengindikasikan bahwa subsistem yang teralamati (memori atau I/O port) telah selesai melakukan operasi baca/tulis.

Mikroprosesor mempunyai dua struktur bus yang berbeda (lihat Gambar 1.49):

1. Bus internal, menghubungkan (*link*) komponen-komponen internal dalam mikroprosesor.

2. Bus eksternal, menghubungkan mikroprosesor dengan unit-unit eksternal seperti memori, pengontrol I/O, dan serpih (*chip*) pendukung lainnya.



Gambar 1.49 Bus internal dan bus eksternal pada mikroprosesor

1.8.1 Siklus Bus

Pada komputer yang berdasarkan bus, komunikasi antara CPU dan subsistem lainnya menggunakan bus. Urutan kejadian yang dikerjakan bus untuk mentransfer satu byte (word) melalui bus data disebut siklus bus. Ada empat macam siklus bus yang utama:

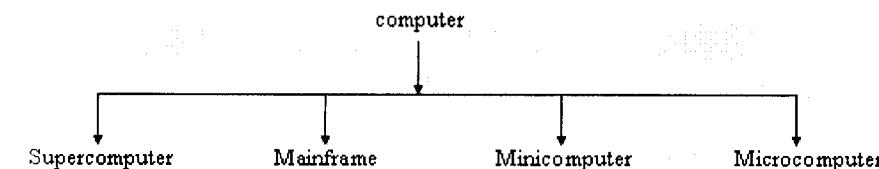
1. *Memory read bus cycle*: CPU membaca data dari lokasi memori
2. *Memory write bus cycle*: CPU menulis data ke lokasi memori.
3. *I/O read bus cycle*: CPU membaca (menerima) data dari port input
4. *I/O write bus cycle*: CPU menulis (mengirim) data ke port output.

Sebuah siklus instruksi bisa saja melibatkan lebih dari satu macam siklus bus. Pengambilan (*fetching*) suatu instruksi melibatkan/membutuhkan *memory read bus cycle*. Penyimpanan (*storing*) hasil di dalam memori membutuhkan *memory write bus cycle*. Pengambilan operand melibatkan *memory read bus cycle*. Eksekusi Instruksi IN atau OUT melibatkan *I/O read bus cycle* atau *I/O write bus cycle*.

CPU mulai suatu siklus bus dengan mengirimkan alamat (alamat memori atau alamat port) pada bus alamat. Semua subsistem yang terhubung ke bus melakukan decode (penerjemahan kode) untuk mengetahui apakah dia teralamati. Hanya subsistem yang teralamati yang terhubung secara logika ke bus dan yang lainnya tidak mengganggu. CPU yang menunjukkan/mengindikasikan jenis siklus bus dengan mengirimkan sinyal kontrol yang tepat (membaca memori, menulis memori, membaca I/O, menulis I/O) pada bus kontrol. Pada siklus bus yang terpilih, CPU mengirimkan satu sinyal kontrol tertentu yang menunjukkan jenis siklus bus yang terpilih tersebut. Selama terjadi siklus bus output, CPU meletakkan data pada bus data dan data diambil oleh subsistem yang teralamati. Selama terjadi siklus bus input, subsistem yang teralamati meletakkan data pada bus data dan CPU mengambil (membaca) data tersebut.

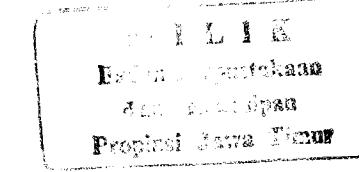
1.9 JENIS-JENIS KOMPUTER

Komputer digital digunakan untuk berbagai aplikasi. Berdasarkan kinerja, ukuran, biaya dan kapasitas, komputer digital dapat diklasifikasikan menjadi empat jenis (Gambar 1.50): *mainframe* atau *maxicomputer*, *minicomputer*, *microcomputer*, dan *supercomputer*.



Gambar 1.50 Jenis-jenis komputer

Karena perkembangan teknologi konstan dan standar yang ada tidak seragam antara pabrik komputer yang berbeda, sejumlah definisi untuk jenis-jenis komputer tidak mudah untuk diterima oleh setiap orang. Seringkali penemuan teknologi baru memberikan pilihan yang baru dalam harga yang murah. Pada Tabel 1.8 dapat dilihat contoh dari klasifikasi komputer. Sistem yang besar (*mainframe*) secara fisik didistribusikan/disimpan ke dalam satu atau dua buah lemari. Komputer ini adalah salah satu komputer yang mahal yang hanya dapat dibeli oleh organisasi atau institusi besar. Minicomputer dikembangkan dengan tujuan memenuhi



komputer murah agar dapat terjangkau oleh organisasi kecil. Beberapa fitur hardware yang ada pada mainframe tidak terdapat pada hardware minicomputer agar biaya atau harga dapat lebih murah. Beberapa fitur yang ada pada hardware komputer mainframe dapat dikerjakan oleh software dalam minicomputer. Sebagai hasilnya, kinerja minicomputer lebih rendah dari mainframe. Penemuan mikroprosesor (CPU tunggal) melahirkan microcomputer yang beberapa kali lebih murah dari minicomputer, tetapi pada saat yang sama menawarkan semua yang terdapat pada minicomputer dengan kecepatan yang rendah. Hal tersebut yang secara praktis mematikan pangsa pasar minicomputer.

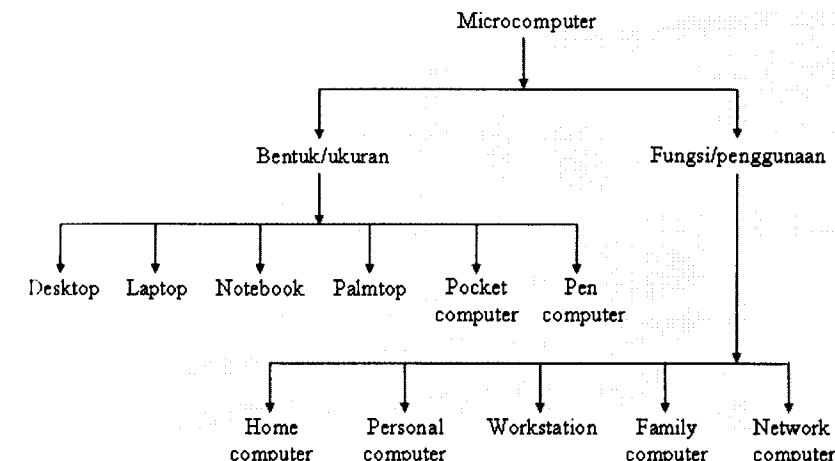
TABEL 1.8 Klasifikasi komputer

Jenis komputer	Komputer khusus	Fitur khas	Wordlength khas
Mainframe	B5000, IBM S/360	Ukuran besar, biaya tinggi	32 bit dan lebih
Minicomputer	PDP-8, HP2100A, TDC-312	Ukuran sedang, biaya rendah	8–16 bit
Microcomputer	IBM PC, TRS 80	Ukuran sangat kecil, biaya sangat rendah	4–32 bit
Supercomputer	CRAY-1, Cyber 203/205	Kecepatan sangat tinggi	64 bit dan lebih

Saat ini **microcomputer** tersedia dalam cakupan yang luas. **Microcomputer** diklasifikasikan dalam dua cara:

1. Berdasarkan penggunaan fungsional atau aplikasi
2. Berdasarkan bentuk atau ukuran

Pada Gambar 1.51 ditunjukkan model-model kedua klasifikasi tersebut.



Gambar 1.51 Klasifikasi mikrokomputer

1.10 BOOTING SEQUENCE

Komputer modern selalu di bawah kontrol sistem operasi. Perangkat lunak lain menyimpan kontrol bila sistem operasi menetapkan CPU untuk hal tersebut. Bagaimana sistem operasi mendapatkan kontrol setelah switching pada komputer? Urutan aksi berikut dilakukan di dalam komputer dengan segera setelah switching hidup (*on*):

1. Daya hidup me-reset CPU (mikroprosesor)
2. CPU mengambil instruksi pertama dari suatu lokasi tertentu dalam memori utama. Alamat ini dikenal dengan *"reset vector"* untuk prosesor. Biasanya alamat ini disimpan di ROM.
3. ROM berisi program pendek yang disebut *"boot strap loader"* yang membaca *boot record* (track 0, sector 1) dari disk dan menyimpannya dalam memori utama (area memori baca/tulis).
4. Selanjutnya *boot strap loader* mengontrol boot record di mana program ini (*boot program*) yang me-load sistem operasi dari disk ke memori utama dalam area baca/tulis dan mengontrol sistem operasi.
5. Sistem operasi berakhir.

Proses pembacaan (*loading*) sistem operasi (setelah daya hidup) ke memori disebut *booting*.

1.11 FAKTOR-FAKTOR KINERJA KOMPUTER

Pada bagian ini kita membahas hal penting yaitu penilaian kinerja komputer. Secara khusus kita fokuskan pembahasan pada pengukuran kinerja yang digunakan untuk menilai komputer. Misalnya, seorang pengguna komputer mengukur kinerjanya berdasarkan waktu yang digunakan untuk mengeksekusi suatu pekerjaan (program). Pada sisi lain, seorang insinyur laboratorium mengukur kinerja sistemnya berdasarkan jumlah total pekerjaan yang diselesaikan dalam suatu waktu. Jadi kalau pengguna mengukur kinerja berdasarkan waktu eksekusi program, maka insinyur laboratorium mempertimbangkan *throughput* lebih penting dalam pengukuran kinerja. Sebuah metrik untuk penilaian kinerja komputer membantu membandingkan desain-desain alternatif.

Kinerja komputer diukur dari jumlah waktu yang digunakan komputer dalam mengeksekusi sebuah program. Boleh jadi diasumsikan bahwa kecepatan mesin penjumlahan merupakan indikator dari performa, tetapi sebenarnya bukan. Ini karena siklus instruksi pada instruksi ADD tidak hanya melibatkan operasi penjumlahan, tetapi juga operasi lain seperti pengambilan instruksi (*instruction fetch*), penerjemahan instruksi (*instruction decode*), pengambilan operand (*operand fetch*), penyimpanan hasil, dan sebagainya. Faktor-faktor yang berkontribusi dalam kecepatan operasi tersebut adalah:

- Pengambilan instruksi (*instruction fetch*): waktu akses memori
- Penerjemahan instruksi (*instruction decode*): kecepatan unit kontrol
- Kalkulasi alamat operand (*operand address calculation*): (1) Waktu akses GPR/waktu akses memori (2) waktu tambahan pengalaman
- Pengambilan operand (*operand fetch*): waktu akses memori/waktu akses GPR
- Eksekusi: waktu tambahan
- Penyimpanan hasil (*store result*): waktu akses memori utama/waktu akses GPR.

Dua komputer yang bekerja pada kecepatan clock (*clock rate*) yang sama, waktu eksekusi untuk instruksi ADD dapat saja berbeda jika kedua komputer tersebut mempunyai organisasi internal yang berbeda. Karena itu, jenis instruksi yang berbeda dan jumlah instruksi yang dieksekusi oleh

CPU ketika menjalankan sebuah program menentukan waktu yang digunakan oleh komputer untuk sebuah program.

1.12 PENGUKURAN KINERJA SISTEM

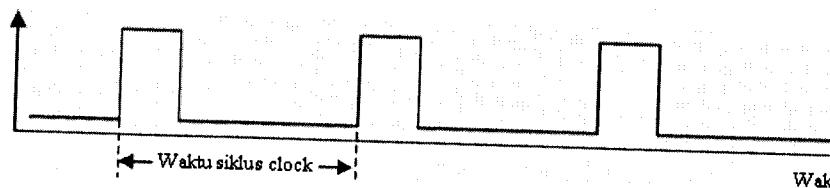
Dahulu istilah MIPS (*Millions of Instruction executed Per Second*) telah sering digunakan untuk menunjukkan kecepatan sebuah komputer. Karena waktu siklus instruksi adalah berbeda untuk instruksi yang berbeda, maka nilai MIPS akan berbeda untuk campuran/gabungan instruksi berbeda di dalam program. Karena itu, nilai MIPS dari suatu komputer hanya memberikan suatu ide kasar mengenai kinerja sistem.

Program Benchmark adalah sebuah program evaluasi yang dikembangkan secara khusus yang dapat digunakan untuk membandingkan kinerja dari komputer yang berbeda. Waktu yang dibutuhkan oleh sebuah sistem komputer untuk mengeksekusi program benchmark digunakan untuk mengukur kinerja sistem komputer tersebut. Hal ini digunakan oleh pembeli, pengembang dan tim pemasaran untuk pengambilan keputusan. Ada dua aspek penting yang perlu dipertimbangkan di sini:

1. Jenis instruksi yang berbeda mempunyai waktu eksekusi berbeda.
2. Jumlah kejadian suatu jenis instruksi bervariasi pada jenis yang berbeda pada program.

Kemampuan dari program benchmark dalam memberikan informasi akurat yang dapat diterima mengenai kinerja sistem dari sebuah komputer bergantung pada gabungan instruksi yang digunakan pada program benchmark. Karena program-program aplikasi dari jenis yang berbeda (saintifik, komersial, dan sebagainya) mempunyai gabungan instruksi berbeda, maka sangat sulit untuk mengembangkan sebuah program benchmark yang akan benar-benar memberikan stimulasi pada semua *real life program*. Karena itu dilakukan pemisahan program benchmark yang dikembangkan dengan gabungan instruksi yang berbeda untuk setiap jenis aplikasi: saintifik, komersial, pendidikan, dan sebagainya. Sementara, tidak ada benchmark yang secara penuh dapat merepresentasikan kinerja sistem keseluruhan; hasil dari suatu kelompok di dalam memilih program benchmark secara hati-hati dapat memberikan informasi yang dapat berharga mengenai kinerja sebenarnya.

Analisis kinerja akan membantu menjawab pertanyaan-pertanyaan seperti seberapa cepat sebuah program dapat dieksekusi menggunakan komputer tertentu? Untuk menjawab pertanyaan tersebut, kita perlu menentukan waktu yang digunakan oleh suatu komputer untuk mengeksekusi suatu pekerjaan yang diberikan. Kita menentukan waktu siklus clock sebagai waktu antara dua tebing naik (atau turun) berurutan dari suatu sinyal clock periodik (lihat Gambar 1.52). Siklus clock membolehkan penghitungan komputasi-komputasi unit, sebab penyimpanan hasil-hasil komputasi disinkronisasikan dengan tebing-tebing clock naik (atau turun). Waktu yang diperlukan untuk mengeksekusi suatu pekerjaan oleh sebuah komputer sering dinyatakan dengan siklus-siklus clock.



Gambar 1.52 Sinyal clock

Jika jumlah siklus clock CPU untuk pengeksekusian suatu pekerjaan dinyatakan dengan J , waktu siklus clock dinyatakan dengan T , dan frekuensi clock dengan f . Maka waktu yang digunakan CPU untuk mengeksekusi suatu pekerjaan (*waktu CPU*) dapat dinyatakan dengan:

$$\text{Waktu CPU} = J \times T = J/f$$

Menghitung jumlah instruksi yang dieksekusi dalam suatu program lebih mudah dibandingkan jika menghitung jumlah siklus-siklus clock CPU yang diperlukan untuk pengeksekusian program tersebut. Karena itu rata-rata jumlah siklus clock per instruksi (*clock per instruction*, CPI) digunakan sebagai alternatif dalam mengukur kinerja. Persamaan berikut menunjukkan bagaimana menghitung CPI.

$$CPI = \frac{\text{siklus - siklus clock CPU pada program}}{\text{jumlah instruksi}}$$

$$\text{Waktu CPU} = \text{jumlah instruksi} \times CPI \times T = \text{jumlah instruksi} \times CPI / f$$

Kita ketahui bahwa set instruksi suatu mesin terdiri atas beberapa kelompok instruksi: instruksi ALU (instruksi-instruksi aritmetika dan logika serta instruksi *assignment* sederhana), load, store, branch dan sebagainya. Jika CPI setiap kelompok instruksi diketahui, maka CPI secara keseluruhan dapat dihitung dengan:

$$CPI = \frac{\sum_{i=1}^n CPI_i \times I_i}{\text{jumlah instruksi}}$$

di mana I_i adalah jumlah instruksi jenis i yang dieksekusi dalam program dan CPI_i adalah rata-rata jumlah siklus clock yang diperlukan untuk mengeksekusi instruksi tersebut.

Contoh 1.5

Anggap penghitungan CPI keseluruhan (total) untuk suatu mesin. Pada data berikut diberikan pengukuran kinerja yang direkam ketika mengeksekusi sekumpulan program-program benchmark. Anggap frekuensi clock (*clock rate*) CPU adalah 200 MHz.

Kelompok instruksi	Persentase kemunculan	Jumlah siklus clock per instruksi
ALU	38	1
Load & store	15	3
Branch	42	4
Lainnya	5	5

Anggap eksekusi 100 instruksi, CPI keseluruhan dapat dihitung:

$$CPI_a = \frac{\sum_{i=1}^n CPI_i \times I_i}{\text{jumlah instruksi}} = \frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100} = 2.76$$

Catatan bahwa CPI mencerminkan organisasi dan arsitektur set instruksi prosesor, sedangkan jumlah instruksi mencerminkan arsitektur set instruksi dan teknologi kompiler yang digunakan. Hal ini menunjukkan derajat saling ketergantungan antara dua parameter kinerja. Karena itu, sangat penting bahwa baik CPI maupun jumlah instruksi merupakan pertimbangan dalam penilaian kebaikan suatu komputer atau ekivalen dengan membandingkan kinerja dua mesin.

Pengukuran kinerja lainnya yang telah mendapat banyak perhatian belakangan ini adalah MIPS (laju eksekusi per unit waktu) yang ditentukan dengan:

$$MIPS = \frac{\text{jumlah instruksi}}{\text{waktu eksekusi} \times 10^6} = \frac{\text{clock rate}}{CPI \times 10^6}$$

Contoh 1.6

Anggap bahwa kumpulan program-program benchmark yang sama pada contoh di atas digunakan untuk mengeksekusi pada mesin lain, katakanlah mesin B, hasil pengukuran yang dilakukan adalah:

Kelompok instruksi	Persentase kemunculan	Jumlah siklus clock per instruksi
ALU	35	1
Load & store	30	2
Branch	15	3
Lainnya	20	5

Berapa MIPS rating untuk mesin pada contoh sebelumnya (mesin A) dan mesin B jika clock rate 200 MHz.

$$CPI_a = \frac{\sum_{i=1}^n CPI_i \times I_i}{\text{jumlah instruksi}} = \frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100} = 2.76$$

$$MIPS_a = \frac{\text{clock rate}}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.76 \times 10^6} = 70,46$$

$$CPI_b = \frac{\sum_{i=1}^n CPI_i \times I_i}{\text{jumlah instruksi}} = \frac{35 \times 1 + 30 \times 2 + 15 \times 3 + 20 \times 5}{100} = 2.4$$

$$MIPS_b = \frac{\text{clock rate}}{CPI_b \times 10^6} = \frac{200 \times 10^6}{2.4 \times 10^6} = 83,33$$

jadi $MIPS_b > MIPS_a$

Penting dicatat di sini bahwa walaupun MIPS telah digunakan sebagai pengukur kinerja mesin, penggunaannya untuk membandingkan mesin-mesin yang mempunyai set instruksi yang berbeda harus dilakukan dengan berhati-hati. Hal ini karena MIPS tidak memasukkan waktu eksekusi. Anggap misalnya, pengukuran berikut dilakukan pada dua mesin berbeda menjalankan kumpulan program-program benchmark.

Kelompok instruksi	Jumlah instruksi (dalam jutaan)	Jumlah siklus clock per instruksi
Mesin A		
ALU	8	1
Load & store	4	3
Branch	2	4
Lainnya	4	3
Mesin B		
ALU	10	1
Load & store	8	2
Branch	2	4
Lainnya	4	3

$$CPI_a = \frac{\sum_{i=1}^n CPI_i \times I_i}{\text{jumlah instruksi}} = \frac{(8 \times 1 + 4 \times 3 + 2 \times 4 + 4 \times 3) \times 10^6}{(8 + 4 + 2 + 4) \times 10^6} \approx 2.2$$

$$MIPS_a = \frac{\text{clock rate}}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.2 \times 10^6} \approx 90.9$$

$$\text{Waktu CPU}_a = \frac{\text{jumlah instruksi} \times CPI_a}{\text{clock rate}} = \frac{18 \times 10^6 \times 2.2}{200 \times 10^6} = 0.198 \text{ detik}$$

$$CPI_b = \frac{\sum_{i=1}^n CPI_i}{jumlah instruksi} = \frac{(10 \times 1 + 8 \times 2 + 2 \times 4 + 4 \times 4) \times 10^6}{(10 + 8 + 2 + 4) \times 10^6} = 2.1$$

$$MIPS_b = \frac{clock rate}{CPI_b \times 10^6} = \frac{200 \times 10^6}{2.1 \times 10^6} = 95.2$$

$$Waktu CPU_b = \frac{jumlah instruksi \times CPI_b}{clock rate} = \frac{20 \times 10^6 \times 2.1}{200 \times 10^6} = 0,252, \text{ detik}$$

$$MIPS_b > MIPS_a \text{ dan } Waktu CPU_b > Waktu CPU_a$$

Contoh di atas menunjukkan bahwa walaupun mesin B mempunyai MIPS yang lebih tinggi daripada mesin A, dia membutuhkan waktu CPU yang lebih lama untuk mengeksekusi kumpulan program-program benchmark yang sama.

Million floating-point instructions per second, MFLOPS (rate/jumlah eksekusi instruksi floating-point per unit waktu) juga telah digunakan sebagai suatu pengukur kinerja mesin yang didefinisikan sebagai:

$$MFLOPS = \frac{jumlah operasi floating - point dalam sebuah program}{waktu eksekusi \times 10^6}$$

Jika MIPS mengukur rate instruksi rata-rata, maka MFLOPS hanya menentukan untuk subset instruksi-instruksi floating-point. Sebuah argumen untuk melawan MFLOPS adalah fakta bahwa kumpulan operasi-operasi floating-point tidak konsisten terhadap mesin-mesin dan karena itu operasi-operasi floating-point yang sebenarnya akan bervariasi dari satu mesin ke mesin lainnya. Argumen lainnya adalah fakta bahwa kinerja suatu mesin untuk suatu program yang diukur dengan MFLOPS tidak dapat disamakan untuk menyediakan suatu metrik tunggal kinerja untuk mesin tersebut.

1.12.1 SPEC Rating

SPEC (*System Performance Evaluation Corporation*) adalah sebuah organisasi non-profit yang diperuntukkan bagi evaluasi kinerja. Korporasi ini

memilih program aplikasi khusus untuk area aplikasi berbeda dan mempublikasikan hasil kinerja untuk komputer komersial penting. Kinerja dari salah satu komputer yang ada secara komersial diambil sebagai komputer referensi. Kinerja komputer lain diukur sebagai pengukuran relatif terhadap komputer standar. Berikut ada beberapa standar yang dilikeluarkan oleh SPEC:

- SPEC 1995, SPEC 2000, Java benchmark: JB2000 dan JVM98
- Web server benchmark: SPECweb99_SSL, SPECweb99 dan SPECweb96
- Mail server benchmark: MAIL2001
- NFS benchmark: SFS97

SPEC memainkan dua fungsi/peranan berbeda yaitu pengembangan benchmark dan penerbitan hasil. SPEC rating ditentukan sebagai berikut:

$$SPEC \text{ rating untuk } X = \frac{Waktu eksekusi program untuk komputer standar}{Waktu eksekusi program untuk X}$$

Dalam praktiknya, program dari beberapa jenis yang cocok dijalankan dan masing-masing SPEC rating-nya dihitung. SPEC rating merupakan cerminan dari berbagai faktor: kinerja CPU, kinerja memori, organisasi sistem, efisiensi komputer, kinerja sistem operasi, dan sebagainya.

RINGKASAN

Komputer adalah alat elektronika yang dapat diprogram yang berfungsi untuk menyelesaikan berbagai macam permasalahan komputasi dan manipulasi data. Komputer modern mempunyai tiga lapisan:

- Hardware, BIOS
- Sistem operasi, Kompiler
- Program aplikasi

Hardware mempunyai lima macam unit fungsional: memori, ALU (*arithmetic and logic unit*), register, unit kontrol, unit masukan dan keluaran (unit I/O). ALU, register dan unit kontrol disebut *central processing unit* (CPU) atau prosesor. Memori dan CPU terdiri atas sirkuit-sirkuit elektronika dan membentuk inti dari komputer. Unit masukan dan keluaran merupakan unit elektromekanik dan biasa juga disebut perangkat peripheral.

Program adalah urut-urutan instruksi yang digunakan untuk menyelesaikan permasalahan. Semua komputer modern menggunakan *stored program concept*. Program dan data semuanya disimpan dalam suatu memori bersama. Unit kontrol mengambil dan mengeksekusi instruksi dalam urutan satu per satu. Program dapat memodifikasinya sendiri; urutan eksekusi instruksi juga dapat dimodifikasi.

Memori utama adalah unit penyimpanan di mana CPU mengambil (*fetch*) instruksi. Kapasitas total suatu memori adalah jumlah lokasi dikalikan dengan *word length*. Waktu yang dibutuhkan oleh memori untuk membaca isi suatu lokasi memori (atau penulisan informasi ke lokasi memori) disebut waktu akses. Memori utama harus dari sebuah RAM (*random access memory*) yang mempunyai waktu akses yang sama untuk semua lokasi. Sebuah CPU dengan n bit alamat, mempunyai kemampuan mengalami memori sebesar 2^n .

Pada komputer modern, umumnya terdapat bagian kecil dari memori utama yang dialokasikan untuk sebuah ROM (*read only memory*). Beberapa memori yang sifatnya permanen diperlukan untuk program-program kontrol dan BIOS yang disimpan di dalam ROM oleh pabrik.

Memori pembantu (*auxiliary memory*) adalah memori yang bersifat eksternal terhadap inti sistem dan dapat menyimpan program dan data dalam jumlah yang besar.

Perangkat peripheral adalah peranti yang dihubungkan dengan inti sistem oleh pengontrol perangkat (atau disebut juga *I/O controller*). Fungsi utamanya adalah mentransfer informasi antara inti sistem dan perangkat. Secara fungsional, pengontrol perangkat berkomunikasi dengan suatu perangkat melalui antarmuka perangkat yang membawa sinyal antara pengontrol perangkat dan perangkat. Semua pengontrol perangkat berkomunikasi dengan CPU atau memori melalui antarmuka sistem. I/O driver untuk suatu perangkat terdiri atas rutin-rutin yang berfungsi melaksanakan berbagai operasi: baca, tulis dan lain-lain.

Setiap instruksi menetapkan sebuah makro-operasi yang akan dilaksanakan oleh field kode operasi (*opcode field*). Untuk mengeksekusi suatu instruksi, CPU harus melaksanakan beberapa mikro-operasi. Mikro-operasi adalah operasi-operasi dasar yang dilakukan CPU. Sinyal clock digunakan sebagai acuan pewaktuan oleh unit kontrol. Sinyal clock adalah sebuah bentuk gelombang periodik karena bentuk gelombangnya berulang dengan sendirinya. CPU mengambil satu instruksi pada satu waktu, mengeksekusinya dan kemudian mengambil instruksi yang berikutnya. Kegiatan ini dikenal dengan siklus instruksi. Dalam keadaan beroperasi (*running state*), CPU melaksanakan siklus-siklus instruksi dan dalam keadaan *halt state* CPU tidak melakukannya.

Interupsi adalah suatu kejadian yang meminta pelayanan khusus pada CPU di saat CPU melakukan suatu proses. Untuk merespons interupsi, CPU bercabang ke suatu rutin layanan interupsi (*interrupt service routine*, ISR). Jika CPU dapat diinterupsi kembali ketika dia sedang mengeksekusi suatu ISR, maka hal ini dikenal dengan *interrupt nesting*. Jika terdapat interupsi jamak (*multiple interrupt*) pada satu waktu, maka urutan pelayanan interupsi dilakukan berdasarkan prioritas interupsi. Pengenalan interupsi CPU dapat ditunda dengan menghalangi CPU. Bila interupsi dihalangi, mereka tetap ditunda. Interupsi-interupsi tertentu harus dilayani tanpa ada penundaan. Interupsi demikian dikenal dengan interupsi yang tak dapat dihalangi (*non-maskable interrupt*) dan CPU tidak menghalanginya.

Rutin I/O dapat dibagi menjadi tiga metode dalam mentransfer data: *program mode*, *interrupt mode*, dan *DMA mode*. Pada *program mode* (*polling*), rutin I/O (karenanya CPU) merupakan *fully dedicated* untuk transfer data dari permulaan hingga akhir. Pada *interrupt mode*, CPU men-switch antara rutin I/O dan beberapa program lain di antara transfer byte-byte berurutan. Transfer data dari perangkat kecepatan tinggi seperti hard

disk dan floppy disk sulit dilakukan dengan program mode atau interrupt mode karena kecepatan transfer mode-mode ini lebih lambat daripada kecepatan data yang datang dari perangkat ini. DMA mode digunakan untuk perangkat berkecepatan tinggi. Pengontrol DMA mengontrol transfer data antara memori dan pengontrol I/O tanpa pengetahuan (campur tangan) CPU.

Pada mikrokomputer, konsep bus digunakan untuk interkoneksi sinyal antar subsistem. Bus adalah lintasan bersama yang digunakan secara bergantian (sharing) untuk sejumlah sumber dan target. Mikroprosesor mempunyai dua struktur bus yang berbeda: bus internal dan bus eksternal. Urutan kejadian yang dilaksanakan pada bus untuk transfer satu byte melalui bus data disebut siklus bus. Port I/O adalah sebuah *program addressable unit* yang dapat mensuplai data ke CPU atau yang dapat menerima data yang diberikan oleh CPU. Secara fungsional port berfungsi sebagai sebuah jendela untuk komunikasi dengan CPU pengantarmukaan hardware port melalui bus data.

Proses pembacaan (*loading*) sistem operasi ke dalam memori utama segera setelah catu daya dihidupkan pada suatu komputer disebut *booting*. *Boot strap loader* adalah sebuah program kecil yang tersimpan di dalam ROM yang membantu dalam proses booting.

Kinerja suatu komputer diukur dari jumlah waktu yang digunakan mengakses suatu program. Program Benchmark adalah program evaluasi yang digunakan untuk membandingkan kinerja sejumlah komputer. Dahulu istilah MIPS digunakan untuk menunjukkan kecepatan komputer. SPEC adalah suatu organisasi non-profit yang khusus mengevaluasi kinerja komputer. SPEC memilih/mengembangkan program-program aplikasi untuk beberapa area aplikasi dan memublikasikan hasil-hasil kinerja untuk komputer-komputer komersial. Kinerja dari suatu komputer komersial yang ada dijadikan sebagai komputer acuan. Kinerja komputer lainnya dihitung secara relatif terhadap komputer standar. Beberapa teknik telah dikembangkan untuk meningkatkan kinerja sistem.

SOAL-SOAL ULANGAN

1. Beberapa keunggulan proses komputasi menggunakan komputer dibandingkan proses komputasi manual adalah: lebih cepat, lebih akurat, lebih andal, lebih konsisten, dan dapat (1) _____ dalam memori yang besar serta merupakan mesin serbaguna yang dapat (2) _____.
2. Komputer modern terdiri atas dua kelompok yaitu komputer analog dan komputer _____.
3. Komputer yang menggunakan komponen-komponen digital bekerja berdasarkan ada tidaknya tegangan listrik (secara diskrit). Dua keadaan ini disebut _____.
4. Komponen dasar pembangun komputer modern adalah _____ yang digunakan untuk membangun gerbang logika (*logic gate*).
5. Pada umumnya pemrosesan dasar dalam CPU dioperasikan secara aritmetika dan/atau secara (1) _____ di dalam unit yang disebut (2) _____.
6. Operasi yang dilakukan dalam komputer menggunakan kode operasi dan operand yang terdiri atas logika 1 dan 0 disebut bahasa _____.
7. Untuk memudahkan manusia dalam memprogram mesin komputer umumnya digunakan level pemrograman yang disebut _____.
8. _____ adalah sebuah penerjemah bahasa yang mengubah program bahasa tingkat tinggi menjadi program bahasa mesin yang ekivalen.
9. Istilah _____ umumnya merujuk pada sirkuit-sirkuit elektronika yang terdapat di dalam mesin komputer. Secara praktis, istilah ini digunakan untuk semua komponen fisik di dalam sebuah komputer termasuk mekanika, rakitan komponen-komponen listrik dan elektronika.
10. Istilah _____ umumnya merujuk pada semua instruksi atau program yang digunakan untuk dapat menjalankan komputer secara baik.
11. Sistem komputer dikelompokkan dalam beberapa lapisan, mulai dari lapisan paling dalam yaitu hardware, BIOS, sistem operasi/kompiler dan lapisan paling luar adalah _____.
12. Kumpulan I/O driver (program untuk pelaksanaan operasi-operasi I/O) untuk berbagai perangkat peripheral dalam komputer yang dapat dipanggil oleh program lain kapan saja suatu operasi I/O harus dikerjakan, disebut _____.
13. _____ adalah sebuah program yang menyelesaikan permasalahan pengguna.

14. _____ adalah sebuah program yang membantu utilisasi efisien sistem oleh program-program yang lain dan para pengguna.
15. _____ komputer adalah sebuah sains (ilmu) untuk tujuan perancangan sistem komputer seperti set instruksi, format instruksi, kode operasi, jenis operand, register, mode pengalamanan, dan lain-lain.
16. _____ komputer adalah ilmu yang memberikan gambar yang lebih dalam mengenai struktur fungsional dan interkoneksi logika antara unit-unit (blok fungsional), termasuk detail atau rincian hardware yang dapat diketahui oleh pemrogram, seperti sinyal-sinyal kontrol, antarmuka komputer dan peripheral serta teknologi memori yang digunakan.
17. Hardware mempunyai lima macam unit fungsional: memori, ALU (*Arithmetic and Logic Unit*), register, unit kontrol, dan unit _____.
18. *Central Processing Unit* (CPU) terdiri atas ALU, register, dan _____.
19. Program dan data dimasukkan ke dalam komputer melalui unit _____.
20. Memori utama menyimpan (1) _____ dan (2) _____.
21. _____ membaca dan menganalisis instruksi satu per satu dan memberikan sinyal kontrol ke seluruh unit untuk melakukan berbagai macam operasi.
22. _____ adalah bagian/unit mesin yang mampu melakukan operasi aritmetika dan logika.
23. Hasil dari instruksi disimpan di memori dan dapat dibawa ke unit _____.
24. Semua komputer modern menggunakan konsep yang disebut _____ yang awalnya disusun oleh tim desain komputer ISA yang dipimpin oleh John Von Neumann. Karena itu biasa disebut konsep Von Neumann.
25. Media penyimpanan utama dalam komputer dikelompokkan dalam dua bagian yaitu: (1) memori utama, dan (2) _____.
26. Sebuah perangkat peripheral terhubung (*link*) dengan inti sistem (CPU dan memori) oleh suatu _____.
27. Secara fisik keberadaan pengontrol perangkat dapat dibedakan dalam tiga macam yaitu (1) sebagai unit yang terpisah, (2) terintegrasi dengan perangkat, dan (3) _____.
28. Tiga macam sinyal antara perangkat dan pengontrol perangkat yaitu: (1) sinyal kontrol, (2) sinyal status, dan (3) sinyal _____.
29. Sinyal RESET merupakan contoh sinyal _____.
30. Sinyal ERROR merupakan contoh sinyal _____.
31. Istilah _____ menunjukkan jumlah bit pada setiap lokasi memori

32. Kapasitas total memori adalah word length dikalikan dengan _____.
33. Waktu yang dibutuhkan oleh memori untuk membaca (atau menulis) pada sebuah lokasi memori disebut _____.
34. Memori yang mempunyai waktu akses yang sama untuk semua lokasi di dalam memori semikonduktor dinamakan _____.
35. Contoh memori yang bersifat *non-volatile* adalah memori _____.
36. Contoh memori yang bersifat *volatile* adalah memori _____.
37. Register yang berkomunikasi dengan memori utama adalah (1) register _____, dan (2) register _____.
38. Data memori yang akan diakses, alamatnya terlebih dahulu ditransfer ke register _____.
39. Data yang dibaca dari memori atau yang ditulis ke memori harus berada di register _____.
40. Antarmuka (*interface*) CPU dengan memori ada dua jenis yaitu: (1) _____, dan (2) _____.
41. Siklus instruksi terdiri atas dua fase utama yaitu fase (1) pengambilan instruksi dan fase (2) _____ instruksi.
42. Format umum instruksi mesin terdiri atas dua field yaitu: field (1) kode operasi (opcode), dan field (2) _____.
43. _____ sebagai penunjuk alamat instruksi yang akan dieksekusi selanjutnya.
44. Instruksi yang sedang didekode disimpan pada register _____.
45. Program counter meletakkan alamatnya pada register _____.
46. Yang melakukan penerjemahan instruksi adalah _____.
47. Operasi CPU terbagi dua yaitu operasi mikro dan operasi _____.
48. Membersihkan register, menambah isi program counter merupakan contoh dari operasi _____.
49. _____ adalah suatu kejadian dalam sistem komputer yang meminta pelayanan penting pada CPU ketika CPU sedang melakukan pemrosesan.
50. Dalam merespons interupsi, CPU menunda eksekusi program saat itu dan melakukan pencabangan ke _____.
51. _____ adalah sebuah program yang melayani interupsi.
52. _____ adalah keadaan di mana CPU diinterupsi ketika sedang mengeksekusi ISR.
53. Urutan pelayanan interupsi disebut _____.
54. Penghalangan interupsi oleh CPU disebut _____.
55. Interupsi yang tidak dapat ditunda atau dicegah disebut _____.

56. _____ merupakan kumpulan rutin I/O untuk berbagai operasi pada perangkat I/O yang spesifik yang mengawasi pemberian perintah/komando ke pengontrol perangkat, melakukan verifikasi status dari pengontrol perangkat dan perangkat serta menangani operasi I/O.
57. Metode transfer data terdiri atas tiga bagian yaitu (1) *programmed mode (polling)*, (2) _____ dan (3) _____.
58. Dari ketiga metode transfer data yang dimaksud pada Soal 57, yang paling lambat adalah _____.
59. Dari ketiga metode transfer data yang dimaksud pada Soal 57, yang paling cepat adalah _____.
60. Konsep penggunaan instruksi IN dan OUT untuk berkomunikasi dengan port I/O dikenal dengan _____.
61. CPU memperlakukan port I/O seperti halnya lokasi memori dalam melakukan transfer data disebut _____.
62. _____ diperlukan di dalam komputer untuk membawa berbagai jenis informasi di antara subsistem (CPU, memori dan pengontrol I/O)
63. Mikroprosesor/CPU mempunyai dua struktur bus yang berbeda yaitu (1) bus _____ dan (2) bus _____.
64. Urutan kejadian yang dikerjakan bus untuk mentransfer satu byte (word) melalui bus data disebut _____.
65. Ada empat macam siklus bus yang utama yaitu: (1) *Memory read bus cycle*: CPU membaca data dari lokasi memori, (2) *Memory write bus cycle*: CPU menulis data ke lokasi memori, (3) _____ dan (4) _____.
66. Kinerja komputer diukur dari _____ yang digunakan komputer dalam mengeksekusi sebuah program.
67. _____ adalah sebuah program evaluasi yang dikembangkan secara khusus yang dapat digunakan untuk membandingkan kinerja dari komputer yang berbeda.

SOAL-SOAL LATIHAN

- Komputer PC desktop mempunyai ukuran yang lebih besar dan menggunakan daya lebih besar dari notebook PC. Mengapa kemudian notebook lebih mahal dari desktop? (arahan: *portability*)
- Kinerja dua komputer berbeda A dan B (mempunyai arsitektur sama) akan dibandingkan oleh sebuah konsultan sebagai bagian dari proses evaluasi. Komputer A beroperasi pada clock 100 MHz dan memberikan

100 MIPS, sedangkan komputer B beroperasi pada clock 120 MHz dan 80 MIPS. Karena beberapa alasan, komputer B telah dipilih oleh konsultan. Dia juga memberikan beberapa saran untuk pengembangan kinerja komputer B dalam modifikasi desain ke depan. Beberapa saran yang diberikan tersebut adalah:

- Ganti memori utama dengan memori yang lebih cepat
- Perkenalkan sebuah memori cache kecil
- Tingkatkan frekuensi clock hingga 200 MHz.

Anggaplah Anda hanya disuruh memilih satu saran tersebut untuk menjaga harga sebagai faktor utama dalam pilihan Anda (a, b, c)? Yang mana yang membutuhkan perubahan arsitektur? Yang mana yang membutuhkan teknologi yang lebih baik?

- Memori semikonduktor adalah memori yang berukuran kecil dan cepat daripada memori magnetik (*core*). Juga menggunakan daya yang lebih sedikit daripada memori magnetik. Keduanya adalah *random access memory* (RAM). Memori magnetik tetap lebih superior dari memori semikonduktor hanya dalam satu aspek. Aspek apakah itu?
- Prosesor Intel mempunyai alamat 32-bit dan memori yang sifatnya *byte addressable*. Berapakah kapasitas memori fisik maksimum untuk membangun komputer dengan mikroprosesor tersebut?
- Mikroprosesor Intel 80286 mempunyai alamat 24-bit. Berapa word maksimum yang lebarnya 32-bit yang memungkinkan untuk memori tersebut?
- IBM PC berbasis Intel 8088 original diberikan sinyal clock 4,77 MHz pada mikroprosesor tersebut. Berapakah periode sinyal clock komputer?
- Sebuah Program Benchmarking dari 200.000.000 instruksi telah dijalankan pada sebuah komputer dengan operasi 100 MHz. Diperoleh bahwa komputer memberikan kinerja 100 MIPS. Tentukan CPI (*cycles per instruction*), siklus per instruksi untuk komputer ini.
- Sebuah Program Benchmarking dengan gabungan instruksi (*instruction mix*) berikut yang telah dijalankan pada operasi komputer dengan sinyal clock 100 ns: 30% instruksi aritmetika, 50% instruksi load/store dan 20% instruksi pencabangan (branch instruction). Waktu siklus instruksi untuk tipe ini berturut turut adalah 1, 0,6 dan 0,8 us. Hitung CPI komputer ini.
- Anggap sebuah program yang dijalankan dalam komputer A selama 50 detik, komputer tersebut mempunyai clock 500 MHz. Kita akan

- menjalankan program yang sama pada mesin lain, B, dalam 20 detik. Jika mesin B memerlukan 2,5 kali siklus clock lebih banyak dari mesin A untuk program yang sama, hitung berapa clock rate yang harus dimiliki mesin B dalam MHz?
10. Misalkan kita mempunyai dua implementasi arsitektur set instruksi yang sama. Mesin A mempunyai waktu siklus clock 50 ns dan CPI 4,0 untuk beberapa program, dan mesin B mempunyai waktu siklus clock 65 ns dan CPI 2,5 untuk program yang sama. Hitung mesin manakah yang lebih cepat?
11. Seorang perancang kompiler mencoba memutuskan antara dua code sequence untuk suatu mesin khusus. Perancang hardware telah memberikan fakta-fakta berikut:

<i>Klas instruksi</i>	<i>CPI</i>
A	1
B	3
C	4

Untuk sebuah bahasa tingkat tinggi, penulis kompiler mempertimbangkan dua sequence yang memerlukan jumlah instruksi berikut:

<i>Code sequence</i>	<i>Jumlah instruksi (dalam jutaan)</i>		
	<i>A</i>	<i>B</i>	<i>C</i>
1	2	1	2
2	4	3	1

Berapa CPI untuk setiap sequence? Code sequence manakah yang lebih cepat? Berapa nilai tersebut?

12. Misalkan sebuah mesin dengan tiga kelas instruksi dan pengukuran CPI sebagai berikut:

<i>Klas instruksi</i>	<i>CPI</i>
A	2
B	5
C	7

Anggap bahwa kita mengukur code untuk sebuah program dalam dua kompiler berbeda dan diperoleh data berikut:

<i>Code sequence</i>	<i>Jumlah instruksi (dalam jutaan)</i>		
	<i>A</i>	<i>B</i>	<i>C</i>
Kompiler 1	15	5	3
Kompiler 2	25	2	2

Anggap bahwa clock rate mesin adalah 500 MHz. Code sequence manakah yang akan mengeksekusi lebih cepat menurut MIPS? Dan menurut waktu eksekusi?

BAB 2

EVOLUSI KOMPUTER

Sasaran bab ini:

1. Evolusi Komputer
2. Sejarah Komputer
3. Generasi Komputer

Proses desain suatu komputer memerlukan persyaratan metodologi saintifik dan mutakhir. Beberapa konsep dan teknik telah dikembangkan untuk mencapai tujuan tersebut. Beberapa di antaranya telah usang karena inovasi-inovasi teknologi, sedangkan sisanya masih tetap digunakan. Perancang dan pengembang di seluruh dunia bekerja dalam beberapa bidang yang berbeda untuk pengembangan komputer.

Pada bab ini kita akan membahas perkembangan komputer. Pembahasan detail akan mencakup bagaimana pembaca dapat mendesain dengan baik, konsep-konsep dan teknik-teknik komputer, sejarah perkembangan serta generasi komputer.

2.1 PROSES DESAIN KOMPUTER

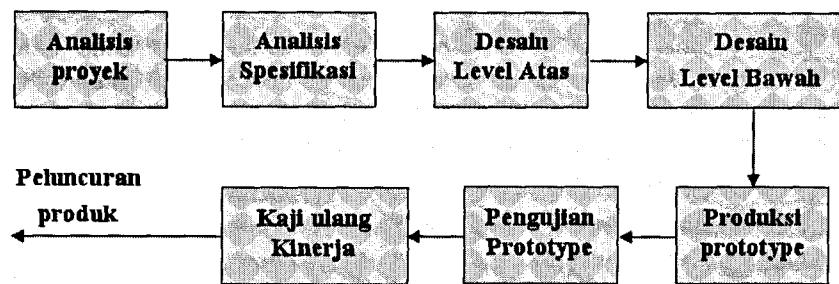
Pengembangan komputer merupakan proses yang mempunyai beberapa langkah seperti yang ditunjukkan Gambar 2.1. Langkah pertama adalah analisis proyek. Pada langkah ini, dikaji mengenai teknik dan kelayakan komersial produk yang ditawarkan (komputer) dan penyiapan laporan kelayakan tersebut. Langkah selanjutnya adalah analisis spesifikasi, di mana dilakukan desain spesifikasi produk dengan menggunakan laporan kelayakan yang sudah dibuat. Dalam desain suatu komputer, hal-hal yang perlu dipertimbangkan adalah:

1. Kinerja yang diharapkan
2. Target aplikasi (bisnis/saintifik/*server/client*, dan lain-lain.)

3. Rentang harga yang diinginkan
4. Fitur-fitur dasar yang diinginkan pengguna dan aspek-aspek teknik

Beberapa hal yang terlibat dalam desain komputer baru adalah:

1. Apakah komputer baru mensyaratkan “*software compatible*” dengan komputer sebelumnya agar semua program yang ada masih dapat digunakan pada komputer baru tersebut?
2. Jika komputer baru secara total mempunyai arsitektur baru dengan sistem operasi yang baru (yang akan dikembangkan), apakah perlu membuat sebuah “*simulator*” (untuk komputer baru) yang dijalankan pada komputer sebelumnya agar program-program yang dikembangkan untuk komputer baru dapat diuji sebelum komputer baru siap?
3. Apakah komputer sebelumnya dapat digunakan untuk “*emulasi*” komputer baru untuk mengkaji kinerja yang diharapkan pada komputer baru?
4. Apakah memungkinkan untuk “*import*” desain subsistem dari komputer sebelumnya agar pengembangan hardware akan menjadi cepat?

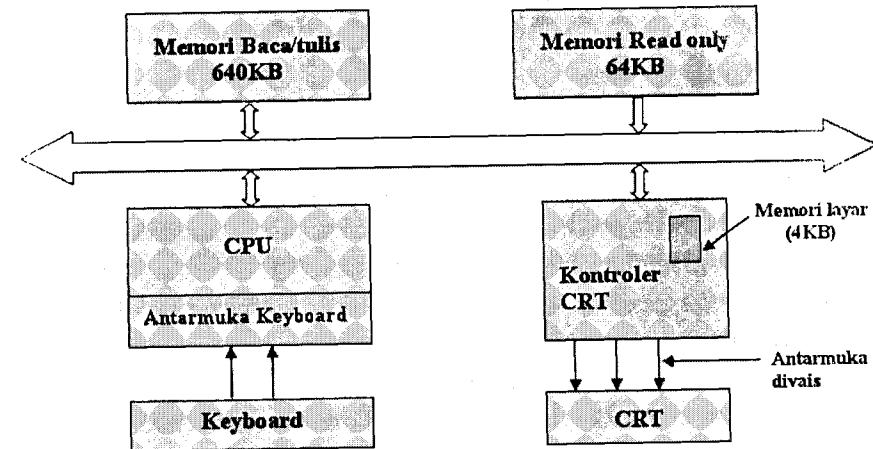


Gambar 2.1 Langkah proses pengembangan komputer

2.2 STRUKTUR KOMPUTER

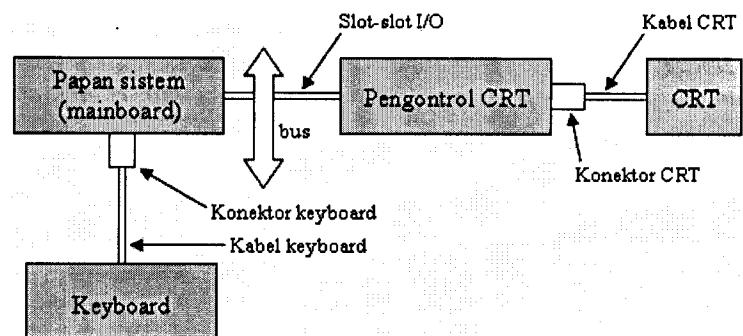
Struktur komputer menunjukkan struktur internal komputer tersebut. Dua aspek struktur komputer adalah: fungsional dan fisik. Struktur fungsional menentukan blok-blok dan hubungan antara blok tersebut. Struktur fungsional merupakan faktor penting untuk mendesain komputer seperti yang tercermin dari kinerjanya. Struktur fisik menetapkan modul

fisik dan hubungan di antaranya. Struktur fisik tidak memengaruhi kinerja tetapi berpengaruh pada keandalan dan biaya. Gambar 2.2 memberikan struktur fungsional parsial dari sebuah PC (*personal computer*) yang menunjukkan empat subsistem: CPU, memori, subsistem CRT (*cathode ray tube*), dan subsistem keyboard.



Gambar 2.2 Struktur fungsional (parsial) sebuah PC

Pada Gambar 2.3 ditunjukkan struktur fisik parsial PC. Papan sistem (papan CPU) berfungsi sebagai papan-utama (*motherboard*) dengan slot I/O yang terinterkoneksi ke berbagai subpapan PCB lainnya. Umumnya subpapan PCB merupakan pengontrol dan memori, misalnya pengontrol CRT merupakan subpapan dalam gambar ini. Papan utama mempunyai empat blok fungsional yaitu CPU, RAM, ROM dan pengontrol keyboard. Pengontrol CRT mempunyai memori penyanga video yang *on-board* (memori layar).

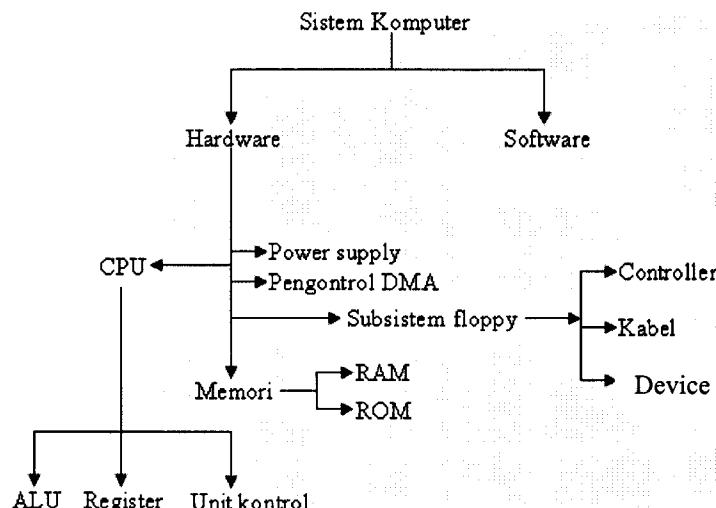


Gambar 2.3 Struktur fisik (parsial) sebuah PC

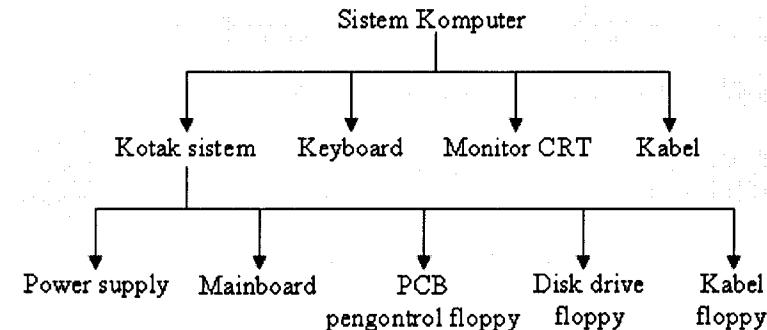
Ruang memori mempunyai tiga komponen:

1. Memori baca/tulis (RAM) untuk program dan sistem operasi
2. ROM untuk BIOS
3. Memori layar (*video buffer*). Secara logika merupakan bagian dari memori utama CPU tetapi juga dapat diakses oleh pengontrol CRT.

Struktur fungsional dan struktur fisik mempunyai komposisi hierarki seperti yang ditunjukkan pada Gambar 2.4 dan Gambar 2.5. Ada dua level komputer. Struktur pada suatu level tidak menunjukkan perilakunya. Perbandingan struktur fisik dari dua komputer menunjukkan komponen-komponen dan teknologinya. Demikian halnya dengan struktur fungsional menunjukkan desainnya tetapi tidak menunjukkan kendala-kendala sinyal waktuan atau urutan-urutan sinyal.



Gambar 2.4 Hierarki struktur fungsional sebuah sistem PC



Gambar 2.5 Hierarki struktur fisik sebuah sistem PC

2.3 FUNGSI KOMPUTER

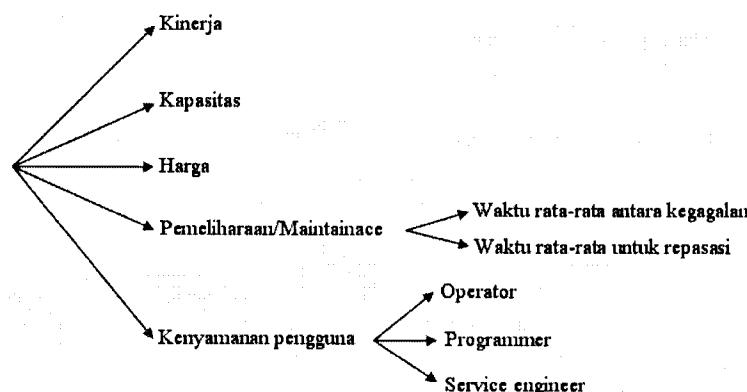
Fungsi komputer menunjukkan perilakunya. Output komputer merupakan fungsi dari input. Pada level keseluruhan (yakni sebagai sebuah sistem), fungsinya adalah eksekusi program yang melibatkan penyimpanan data, pemrosesan data dan transfer data. Pada Tabel 2.1 diberikan fungsi-fungsi pada level-level tersebut.

TABEL 2.1 Fungsi dan Level Komputer

No	Level	Fungsi (contoh kasus)
1	Level Sistem	Eksekusi program
2	Level Subsistem	CPU: pemrosesan instruksi, memori, <i>back-up memory</i> , unit input, pentransferan input pengguna, unit output, sistem pentransferan output ke pengguna.
3	Level Register	PC menunjuk ke alamat instruksi berikutnya, ACC berisi hasil operasi ALU, IR menyimpan instruksi yang diambil dari memori, MAR memberikan alamat lokasi memori untuk operasi baca/tulis.
4	Level Sirkuit Logika	IR terbuat dari sejumlah flip-flop D dengan input clock bersama. Sinyal kontrol $IR \leftarrow MBR$ memicu register.
5	Level Komponen	Keyboard menggunakan switch kapasitor, bila sebuah tombol ditekan maka perubahan kapasitansi antara dua pelat dirasakan sebagai sebuah penekanan tombol.

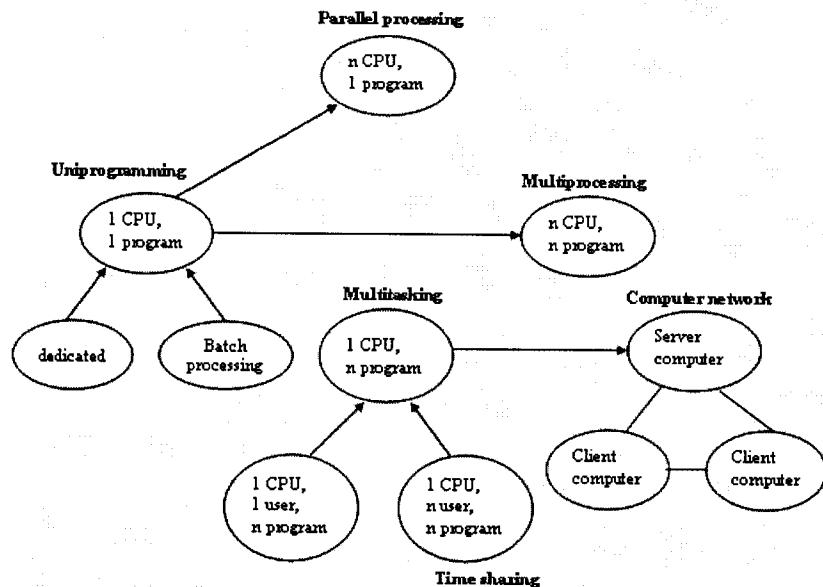
2.4 DIMENSI-DIMENSI EVOLUSI KOMPUTER

Ada lima dimensi untuk mengukur keunggulan komputer yaitu kinerja, kapasitas, harga, *maintanability* dan kenyamanan pengguna. Masalah-masalah ini digunakan untuk mendesain komputer. Kinerja dan kapasitas biasanya lebih banyak dipengaruhi oleh faktor teknologi, konsep dan teknik.

**Gambar 2.6 Dimensi-dimensi penting evolusi**

2.4.1 Evolusi Mode-mode Penggunaan Komputer

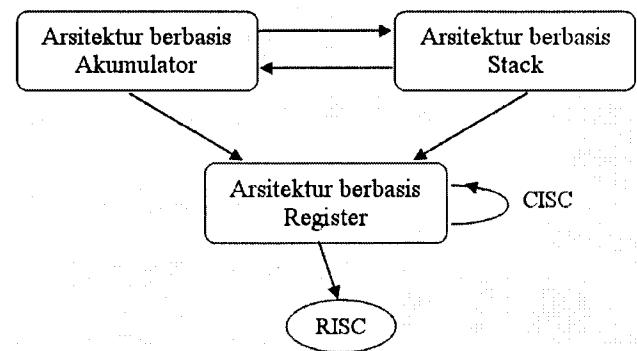
Pada Gambar 2.7 ditunjukkan mode-mode penggunaan sistem komputer pada waktu yang berbeda. Pada setiap mode, sistem operasi dan software sistem yang berhubungan mempunyai beberapa fitur-fitur unik. Misalnya, dalam suatu sistem *microprogramming*, sistem operasi menjamin proteksi setiap program tanpa interferensi dari yang lain.

**Gambar 2.7 Evolusi mode penggunaan komputer**

2.4.2 Evolusi Arsitektur CPU Dasar

Tiga arsitektur CPU dasar ditunjukkan pada Gambar 2.8. Pertama, pada arsitektur berbasis register, operand-operand untuk instruksi disimpan dalam register CPU dan karena itu operand-operand dibaca dengan cepat ketika siklus instruksi. Sedangkan yang kedua yaitu pada arsitektur berbasis akumulator, jumlah instruksi dalam program meningkat tetapi eksekusi instruksi menjadi cepat karena satu operand sudah berada dalam akumulator itu sendiri. Arsitektur yang terakhir adalah arsitektur berbasis

stack, di mana pemrogramannya sangat sederhana karena operasi aritmetika dilakukan pada item teratas dari stack tersebut.

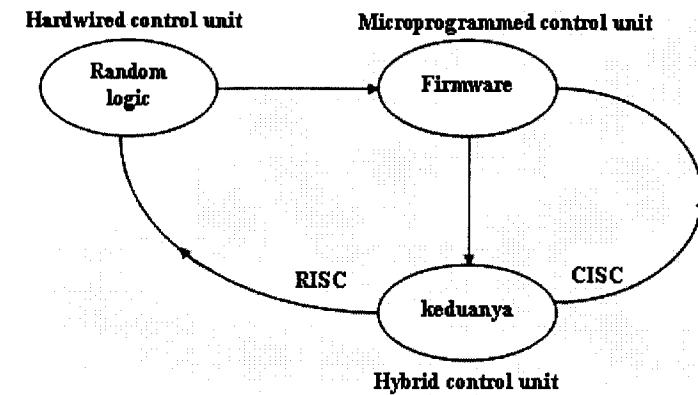


Gambar 2.8 Mode arsitektur CPU

2.4.3 Evolusi Unit Kontrol dan Penerjemahan Instruksi

Pada Gambar 2.9 ditunjukkan evolusi penerjemahan instruksi (*instruction decoding*) pada unit kontrol. *Hardwired control unit* mempunyai sirkuit hardware yang menerjemahkan opcode dalam instruksi dan membangkitkan sinyal-sinyal kontrol yang tepat menggunakan referensi-referensi waktu yang disediakan oleh sinyal clock. *Microprogrammed control unit* menyimpan pola-pola bit untuk setiap instruksi yang sesuai dengan sinyal-sinyal kontrol dalam beberapa mikroinstruksi. Sebuah ROM digunakan untuk menyimpan mikroprogram (urutan mikroinstruksi) untuk setiap instruksi. Selama eksekusi program, unit kontrol mengambil mikroprogram yang sesuai dengan opcode. Jadi, unit kontrol berfungsi sebagai penyedia mikroprogram.

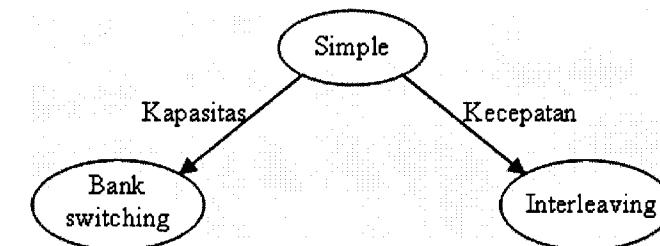
Pengembangan mikroprogram lebih mudah dari perancangan sirkuit untuk hardwired control unit, tetapi kecepatannya lebih rendah karena pola-pola bit harus diambil dari memori kontrol. *Hybrid control unit* menggunakan gabungan kedua ide tersebut, di mana bagian dari unit kontrol yang membutuhkan waktu kritis digunakan hardwired dan selebihnya digunakan microprogrammed. Pada sistem RISC harus secepat mungkin, karena itu biasanya digunakan hardwired control unit.



Gambar 2.9 Evolusi desain unit kontrol—penerjemahan instruksi

2.4.4 Evolusi Teknik-Teknik Memori Utama

Pada Gambar 2.10 ditunjukkan beberapa teknik memori. Kecepatan CPU meningkat secara konstan karena kemajuan teknologi komponen, namun tidak dibarengi dengan teknologi memori yang cepat. Karena itu tujuan utama perbaikan kinerja adalah pada memori dengan teknologi yang ada. *Interleaving* adalah salah satu konsep pembagian memori menjadi dua bagian lokasi yaitu alamat ganjil dan genap. Dengan kata lain, lokasi yang berdekatan ditempatkan dalam modul yang terpisah yang dapat diakses secara bersamaan sehingga secara keseluruhan mengurangi waktu akses.



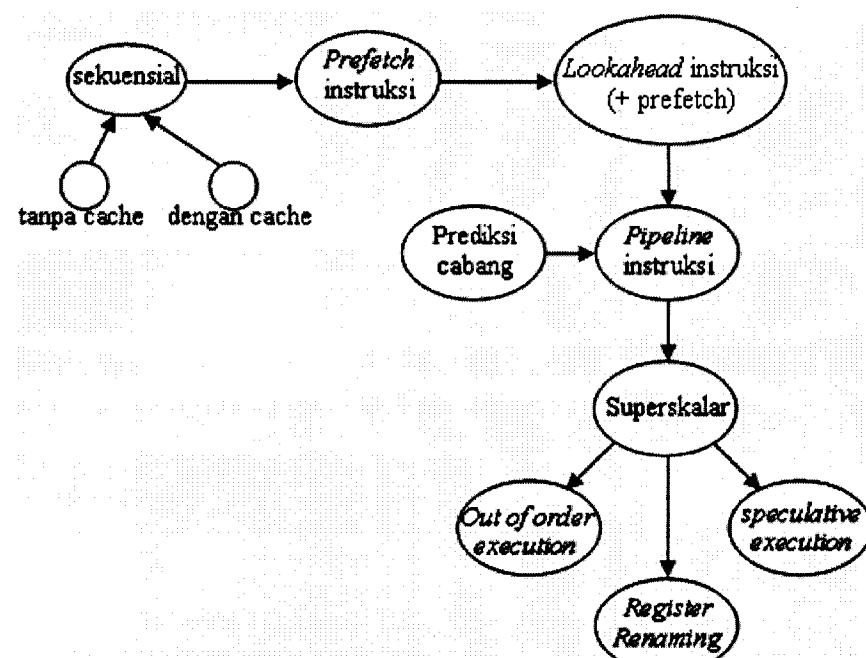
Gambar 2.10 Evolusi konsep memori utama

Pada CPU yang tersedia, kapasitas memori utama dibatasi oleh jumlah bit alamat. *Bank switching* merupakan konsep yang mengatasi masalah ini

tanpa sepengetahuan CPU dengan bantuan atau kerja sama dengan sistem operasi. Bank-bank jamak dengan kapasitas yang sama digunakan dan pada satu waktu salah satunya dipilih.

2.4.5 Evolusi Penanganan Siklus Instruksi

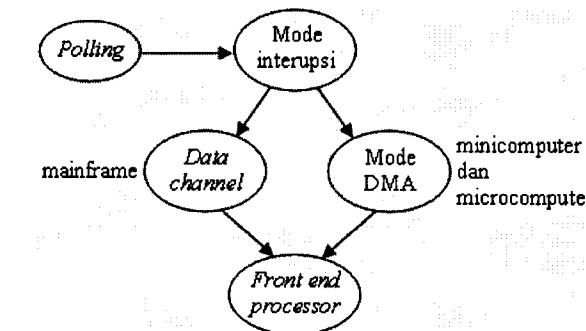
Secara umum, kinerja suatu subsistem dapat ditingkatkan dengan berbagai teknik seperti: paralelisme dengan duplikasi hardware, paralelisme dengan aksi mendahului (lebih dulu), tumpang tindih, dan realokasi. Pada Gambar 2.11 ditunjukkan beberapa teknik penanganan siklus instruksi untuk meningkatkan jumlah instruksi yang diproses setiap detik.



Gambar 2.11 Evolusi pada penanganan siklus instruksi

2.4.6 Evolusi Teknik-Teknik I/O

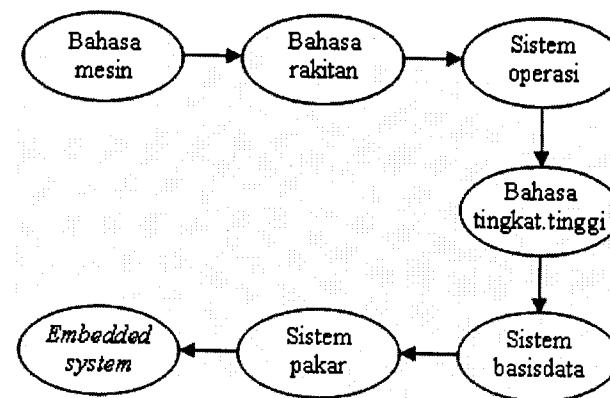
Pada Gambar 2.12 ditunjukkan teknik-teknik transfer data dengan perangkat peripheral. Teknik-teknik I/O tersebut pada setiap perangkat menawarkan kecepatan atau transfer rate yang beragam dan juga biaya (hardware).



Gambar 2.12 Evolusi teknik-teknik I/O

2.4.7 Evolusi Software Sistem

Sistem operasi dan pengembangan software sistem lainnya telah berkembang bersama dengan hardware dan arsitektur. Pada Gambar 2.13 ditunjukkan beberapa pengembangan penting dari software sistem. Pada setiap pengembangan, terjadi redefinisi peranan pada sisi pemrogram (programmer). Pemrogram lebih berkonsentrasi pada algoritma daripada mengatasi masalah-masalah pada level bahasa mesin. Namun, terjadi sejumlah waktu tambahan (*overhead*) pada eksekusi program pada setiap evolusi baru tersebut. Waktu tambahan tersebut menjadi tidak berarti karena kecepatan CPU juga semakin meningkat dan adanya teknik-teknik parallelisme dan *overlap*. Sedangkan pada *embedded system*, software ditanam secara permanen dalam aplikasi tersebut yang telah dilengkapi dengan mikrokontroler yang lebih andal.



Gambar 2.13 Evolusi software sistem

2.5 SEJARAH KOMPUTER

Dahulu di awal pengembangan komputer dicapai dengan kerjasama proyek antara universitas dengan dana yang berasal dari pemerintah. Komputer saat ini merupakan hasil kombinasi dari upaya para ilmuwan sekitar 60-70 tahun lalu. Kebutuhan akan komputasi yang cepat oleh militer Amerika Serikat mempunyai kontribusi yang penting dalam penemuan komputer dahulu, sedangkan pengembangan saat ini banyak didasari oleh kebutuhan industri. Komputer dengan komputasi kecepatan tinggi atau kinerja komputer yang tinggi merupakan hal yang sangat mendasar dalam era perkembangan komputer hingga saat ini. Di samping itu, kapasitas memori dan tentunya harga juga menjadi titik fokus para ilmuwan dalam mendesain komputer yang harus bisa dikompromikan satu sama lain, tergantung bergantung aplikasi yang diinginkan.

Tiga ratus tahun sebelum tahun 1900-an sejumlah teknologi mesin yang kompleks semakin meningkat seperti konstruksi roda *gear*, pengungkit dan katrol digunakan untuk melakukan operasi-operasi dasar seperti penjumlahan, pengurangan, perkalian dan pembagian. Kartu berlubang digunakan untuk mengontrol secara otomatis serangkaian daftar kalkulasi dan telah mampu untuk diprogram. Device Perangkat ini dapat melakukan komputasi matematika tabel logaritma dan fungsi trigonometri dengan pendekatan atau bantuan polinomial. Keluaran atau hasil komputasi dapat disimpan pada kartu berlubang atau dicetak di atas kertas. Relai elektromekanik

digunakan untuk penyambungan sistem telepon pada awalnya, kemudian digunakan untuk melakukan fungsi logika dalam komputer yang dibangun dalam perang Perang dunia Dunia II. Pada saat yang sama, komputer elektronik pertama telah didesain dan dibuat oleh Universitas Pennsylvania dengan teknologi tabung hampa (*vacuum tube*) yang juga digunakan untuk peralatan radio dan radar militer. Tabung hampa, selain dapat melakukan operasi-operasi logika, juga sudah digunakan untuk media penyimpanan data.

Berbagai teknologi telah digunakan untuk pembuatan hardware komputer seperti pabrikasi prosesor, memori dan unit I/O komputer. Generasi komputer dapat dibagi menjadi lima generasi yaitu generasi pertama tahun 1945 hingga 1958, kemudian generasi kedua pada 1958–1966, generasi ketiga 1966–1972 dan generasi keempat 1972–1978 dan generasi kelima 1978 sampai sekarang. Pada Tabel 2.2 diberikan evolusi generasi sistem komputer.

Tabel 2.2. Generasi Sistem Komputer

NO. GENERASI	TEKNOLOGI	DURASI	KOMPUTER POPULER	PENEMUAN BARU YG UTAMA
1	Tabung hampa	1945—1958	Mark I, ENIAC, EDVAC I, IBM 650, IBM 701	Stored Program Concept, memori magnetik magnetik sebagai memori utama, aritmetika biner <i>fixed point</i>
2	Transistor	1958—1966	ATLAS, B 5000, IBM 1401, ICL 1901, PDP-1, MINSK-2	Sistem operasi, multiprogramming, compiler/ompiler, hard disk, magnetik/magnetik, aritmetika biner floating point, minicomputer
3	Sirkut terpadu (SSI dan MSI)	1966—1972	IBM System/360, UNIVAC 1100, hp 2100 A, PDP-8	Multiprocessing, memori semikonduktor, memori virtual, memori cache, supercomputer
4	LSI	1972—1978	ICL 2900, HP 9845 A, Intel 8080	Konsep RISC, microcomputer, kontrol proses, workstation
5	VLSI	1978 —	IBM RS/6000, keluarga SUN Micro System Ultra SPARC	Networking, sistem server, multimedia, embedded system

2.5.1 Komputer Generasi Pertama

Komputer generasi pertama ini murni mesin hardware. Tidak mempunyai sistem operasi. Pemrograman dilakukan dalam bahasa mesin, yang berbeda setiap komputer. Pengguna bekerja pada sejumlah switch/saklar pada panel depan baik untuk start, run dan halt komputer. Status internal ditampilkan pada sejumlah lampu pada panel depan. Umumnya hanya dapat dioperasikan oleh desainer atau programmer pemrogram karena kompleks.

Beberapa kontribusi utama Komputer Generasi-1 adalah:

1. Menggunakan tabung hampa untuk pemrosesan dan penyimpanan
2. Memori kecepatan tinggi bersama untuk program dan data
3. Menggunakan memori utama cepat dan memori sekunder lambat
4. Menggunakan instruksi input-output
5. Pertama diperkenalkan *ferrite core memory*
6. Pertama diperkenalkan bahasa rakitan untuk menghindari kebosanan pemrograman bahasa mesin
7. Menggunakan *electromechanical magnetic drum* sebagai memori sekunder
8. Menggunakan register untuk penyimpanan operand dan hasil dari instruksi di dalam CPU
9. Menggunakan divais perangkat peripheral seperti pita maknetik-magnetik, magnetic drum, pita kertas dan kartu berlubang
10. Menggunakan konsep interupsi

Komputer ENIAC

Komputer ENIAC (*Electronic Numeric Indicator and Computer*) dikembangkan di Universitas Pennsylvania untuk menangani tabel tabel balistik angkatan laut Amerika Serikat. Bekerja dengan bilangan desimal pada sekumpulan akumulator. Lebih cepat 1000 kali dari komputer relai. Pemrogramannya membosankan karena menggunakan saklar manual dan kabel untuk *setting-up*. Digunakan pada saat perang Perang dunia Dunia II untuk kalkulasi otomatis pada tabel balistik, tetapi baru dipublikasikan pada tahun 1946. Pada Tabel 2.3 diberikan fitur-fitur yang dimiliki komputer ENIAC.

Tabel 2.3. Fitur Komputer ENIAC

No.	Fitur	Keterangan
1	Jumlah <i>vacuum tube</i> hampa	18000
2	Konsumsi daya	140 kW
3	Kebutuhan Ruang	1800 feet per segi
4	Aritmetika	Desimal
5	Word length	10 digit
6	Tipe memori utama	Memori program dan data terpisah
7	Kapasitas memori	20 x 10 digit

No.	Fitur	Keterangan
8	Kecepatan	5000 penjumlahan / detik
9	Operasi-operasi utama	Penjumlahan, pengurangan, perkalian, pembagian, kalkulasi akar pangkat dua
10	Divais Perangkat Peripheral	<i>Punch card, electric typewriter</i>

Komputer EDVAC dan Stored Program Concept

Komputer EDVAC (*Electronic Discrete Variable Computer*) merupakan komputer yang sangat sederhana, struktur *fixed physical* dan dapat mengeksekusi berbagai komputasi menggunakan kontrol pemrograman yang tepat tanpa modifikasi unit. EDVAC adalah komputer pertama menggunakan stored program concept. Penggunaan memori meliputi memori utama 1K word yang cepat, memori sekunder 20K word yang lambat. Format instruksi menggunakan 3-alamat:

1. dua alamat untuk penyimpanan operand
2. satu alamat untuk penyimpanan hasil
3. satu alamat untuk penunjuk alamat instruksi berikutnya.

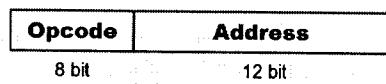
Komputer modern umumnya menggunakan Stored Program Concept, yang awalnya disusun oleh tim desain *ISA computer* dipimpin John Von Neumann. Karena itu biasanya disebut konsep atau **arsitektur Von Neumann**.

Stored Program Concept:

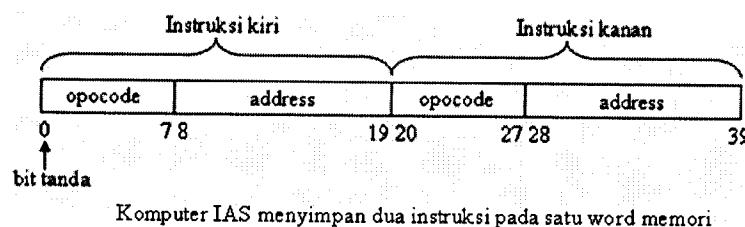
"Program bahasa mesin disimpan di dalam komputer serta data relevan lainnya, dan secara intrinsik komputer mampu memanipulasi program dan data tersebut, misalnya mengambil (load) data/program dari disk ke memori, memindahkannya dari satu lokasi memori ke lokasi memori lainnya, dan menyimpannya kembali ke disk."

Komputer IAS (*Von Neumann Machine*)

Komputer IAS dikembangkan di *Princeton Institute for Advanced Studies* yang merupakan model dasar untuk stored program concept yang diikuti pada hampir semua komputer berikutnya hingga kini. Pemimpin tim proyek adalah John Von Neumann. Instruksi komputer IAS mempunyai dua field yaitu opcode dan address seperti pada Gambar 2.14.



Gambar 2.14(a). Format dasar instruksi IAS yang terdiri dari atas dua field



Gambar 2.14(b). Format instruksi IAS dalam satu word yang terdiri dari atas dua instruksi

Tabel 2.4. Fitur Komputer IAS

No	Fitur	Keterangan
1	Arimetika	Biner, fixed point
2	Jumlah instruksi	21
3	Format instruksi	Single address
4	Panjang instruksi	20 bit
5	Kapasitas memori	1K word, dapat diperluas menjadi 4 K word
6	Panjang memory word	40 bit
7	Jenis dan kapasitas memori sekunder	Magnetic drum; 16 K word

Kelebihan Komputer IAS:

- Merupakan mesin alamat tunggal
- Instruction length pendek sehingga menghasilkan program yang ukurannya kecil dan karenanya keperluan memorinya kecil. Hal ini menyebabkan pengurangan biaya sistem.
- Pengambilan (*fetch*) instruksi dilakukan dua buah sekaligus dan membawanya ke memori, sehingga satu instruksi selalu *prefetched*. Hal ini mengurangi waktu akses untuk instruksi yg yang kedua, sehingga mempercepat waktu siklus instruksi.
- Penggunaan instruksi “*address modify*”, menghasilkan perubahan field alamat instruksi yang lain dalam memori.

Kekurangan Komputer IAS:

- Lemah dalam pelaksanaan operasi I/O. Instruksi input atau instruksi output menghasilkan transfer data antara device perangkat input dan memori atau antara memori dan divais perangkat output. Pada kedua kasus, data harus melalui DPU, karena itu tidak ada “akses memori langsung, DMA” antara memori dan subsistem I/O.
- Tidak mempunyai tipe instruksi “CALL” dan “RETURN”. Karena itu tidak memungkinkan fasilitas subrutin.

Setelah IAS, Eckert—Mauchly Corporation mengembangkan komputer UNIVAC (*Universal Automatic Computer*). Komputer ini cocok untuk aplikasi saintifik dan komersial. Berikutnya diikuti oleh UNIVAC II yang menawarkan kinerja tinggi dan mempunyai kapasitas memori yang besar. Berikutnya seri komputer UNIVAC 1100 dengan kompatibilitas antara berbagai model yang telah dikeluarkan.

2.5.2 Komputer Generasi Kedua

Komputer generasi kedua ditandai dengan penemuan besar yakni dibuatnya komponen elektronika transistor oleh John Bardeen, Walter Brattain dan William Shockley tahun 1948. Transistor ini menggantikan tabung hampa yang sangat revolusioner karena selain hemat energi juga ukurannya yang semakin kecil dibandingkan dengan tabung hampa.

Beberapa kontribusi utama komputer generasi kedua adalah:

1. Menggunakan transistor yang lebih kecil dan juga hemat daya dibandingkan dengan tabung hampa
2. Beberapa perusahaan seperti IBM, NCR dan RCA dll, dengan cepat memperkenalkan teknologi transistor yang meningkatkan keandalan (*reliability*) komputer.
3. Digunakan *Printed Circuit Board* (PCB) sebagai pengganti sirkuit pengkabelan yang bersifat lebih modular yang memudahkan melakukan penggantian.
4. Produksi dan pemeliharaannya lebih mudah.
5. Menggunakan pemrograman bahasa tingkat tinggi yang merupakan lompatan yang besar untuk komputer generasi kedua ini.
6. Pembuat komputer juga telah mengembangkan compiler kompiler yang bervariasi seperti FORTRAN dan COBOL.

7. Mempunyai berbagai macam peripheral seperti *console typewriter, card reader, line printer, CRT display, graphic device*, dan lain-lain.
8. Program aplikasi baru mulai tersedia seperti untuk akuntansi, pajak, *inventory control, purchase order generation, invoicing*, dan lain-lain.
9. Organisasi besar yang menggunakan komputer juga membentuk tim pemrogram untuk pengembangan house program.

Tabel 2.5. Fitur-fitur komputer generasi kedua

No	Fitur	Tipe	Keterangan
1	<i>Operating system</i>	<i>System software</i>	Mengatur sumber daya sistem dan penanganan keperluan pengguna yang berasal dari program aplikasi pengguna.
2	<i>Batch processing</i>	<i>System usage</i>	<i>Multi programmer/user</i> berbagi/sharing dengan sistem besar yang tersentral yaitu dengan mengirimkan programnya untuk <i>batch</i> dan mengambil hasilnya kemudian.
3	<i>Multiprogramming</i>	Peningkatan throughput sistem	Esekusi bersamaan/ <i>concurrent</i> pada multi program; multiplex CPU akan menghindari waktu kosong selama operasi I/O.
4	<i>Timesharing</i>	<i>System usage</i>	<i>Multiple remote user</i> berbagi pada sebuah komputer melalui terminal-terminal; sistem mengalokasikan potongan-potongan waktu ke terminal user yg menawarkan respons yg cepat
5	Bahasa pemrograman tingkat tinggi. Compiler/Kompiler	<i>Programmer aid</i>	Penyederhanaan pemrograman komputer; tidak diperlukan pengetahuan hardware atau bahasa mesin untuk membuat program; produktivitas pemrogram meningkat.

No	Fitur	Tipe	Keterangan
6	Hard disk magnetik	<i>Auxiliary storage</i>	Lebih cepat dan lebih andal dari <i>magnetic drum; read write head</i> melayang
7	<i>Index register</i>	<i>Programmer aid</i>	Digunakan untuk pengalaman operand pada iterasi; menawarkan efisiensi pemrograman.
8	Instruksi CALL dan RETURN	Peningkatan throughput sistem	Menawarkan fasilitas subrutin; menghindari pemrograman yg yang repetitif; meningkatkan produktivitas pemrogram serta utilitas utilisasi ruang memori.
9	Aritmetika floating point	Dikhususkan untuk operasi floating point pada ALU	Untuk aplikasi saintifik yang memerlukan presisi tinggi.
10	<i>Data channel / DMA transfer</i>	Dikhususkan untuk transfer data pada hardware.	Mendukung divais perangka kecepatan tinggi dan juga mengizinkan parallelisme antara CPU dan I/O
11	<i>Minicomputer</i>	<i>Low cost computer</i>	Menghasilkan komputer untuk organisasi dan institusi kecil; mengurangi hardware dibandingkan dengan sistem yg yang besar serta mengurangi kecepatan.

2.5.3 Komputer Generasi Ketiga

Generasi komputer ketiga ini ditandai dengan penemuan rangkaian terpadu (*integrated circuit, IC*) yang terbuat dari silikon yang ditemukan oleh Robert Noyce 1958. Pada waktu itu dengan teknologi IC tersebut dapat dimasukkan puluhan transistor yang dipadukan secara bersama dalam satu chip tunggal. Dengan demikian dapat dibuat komputer yang lebih kecil lagi, semakin kompak dan handal serta lebih murah.

Beberapa kontribusi utama komputer generasi ketiga adalah:

1. Penemuan chip IC yang merupakan sukses besar dalam bidang elektronika untuk membangun sistem mikroelektronika
2. IC mempunyai banyak keuntungan dibandingkan komponen diskrit, misalnya ukuran kecil, kecepatan lebih tinggi, biaya rendah, meningkatkan keandalan (*reliability*)
3. Penggunaan komputer dalam suatu pemrosesan kontinyu dan sektor manufaktur seperti penyulingan BBM dan distribusi daya listrik menjadi populer.
4. Perusahaan yang terkenal seperti IBM, UNIVAC, HP, ICL dan DEC mendominasi industri komputer.
5. Dominasi *minicomputer* membuat kesempatan kerja yang lebih banyak untuk komputer profesional.

Tabel 2.6. Fitur-fitur komputer generasi ketiga

No	Fitur	Tipe	Keterangan
1	Memori Virtual	Biaya berkurang dengan memori fisik terbatas	Sistem mengatur program besar yang sedang berjalan melalui kerja sama antara CPU dan sistem operasi.
2	Pipelining	Parallelisme dalam siklus instruksi	Throughput CPU secara keseluruhan meningkat.
3	Multiprocessing	CPU yang banyak dalam sebuah sistem	Eksekusi secara simultan dari beberapa program dengan CPU yang berbeda.
4	Memori semikonduktor	Memori Teknologi baru pada chip IC	Kecepatan lebih tinggi, ukuran kecil, dan mudah pemeliharaannya dibandingkan memori core.
5	Memori cache	Intermediate hardware buffer antara CPU dan memori utama	Menghemat waktu CPU (dalam pengambilan instruksi/operand dengan mensuplai beberapa instruksi/operand dari memori buffer).
6	Local storage	Register internal dalam CPU	Pengambilan operand dan penyimpanan hasil lebih cepat.
7	Konsep bus	Komunikasi tipe baru antara CPU dan subsistem lainnya	Sharing path; biaya berkurang, komunikasi lebih lambat.

No	Fitur	Tipe	Keterangan
8	Komunikasi data	Komunikasi antar komputer	Transfer data jarak jauh melalui saluran telepon.
9	Micro-diagnostic	Membantu pemeliharaan; <i>auto diagnostic</i>	Untuk aplikasi saintifik yang memerlukan presisi tinggi.

2.5.4 Komputer Generasi Keempat

Pada era ini, teknologi LSI (*large scale integration*) yang menggantikan teknologi SSI (*Small scale integration*) memberikan semakin banyak jumlah transistor yang dapat ditanamkan dalam sebuah chip tunggal. Dengan demikian komputer generasi keempat ini mempunyai kinerja yang semakin baik yang ditandai dengan semakin banyaknya jumlah dan ragam register dalam CPU serta penggunaan memori yang semakin cepat dengan kapasitas yang lebih besar.

Beberapa kontribusi utama komputer generasi ketiga adalah:

1. Teknologi LSI menyediakan kapasitas chip IC yang lebih padat.
2. Penemuan mikroprosesor oleh INTEL melahirkan *microcomputer*.
3. Beberapa perusahaan semikonduktor seperti Motorola, Fairchild, Texas Instrument dan Zilog membuat mikroprosesor yang menawarkan kemampuan yang fantastik.
4. *Workstation* tangguh diperuntukkan bagi aplikasi khusus seperti CAD, pengujian, dll dan lain-lain.
5. Penggunaan *home computer* dan *personal computer* yang lebih luas, misalnya untuk pelaku bisnis kecil, dan lain-lain dll.

Tabel 2.7. Fitur-fitur komputer generasi keempat

No	Nama Fitur	Tipe	Keterangan
1	RISC	Set instruksi sederhana	Unit kontrol lebih sederhana dan peningkatan parallelisme mencapai sedikitnya satu eksekusi instruksi per clock
2	Workstation	Komputer Aplikasi khusus	Sistem kecepatan tinggi untuk aplikasi khusus; hardware khusus dan software yang sesuai

No	Nama Fitur	Tipe	Keterangan
3	Mikroprosesor	Chip tunggal untuk CPU	Komputer biaya rendah sebagai tantangan <i>minicomputer</i> dan penggunaan komputer secara luas pada semua bidang
4	Kontrol Proses	Otomatisasi pabrik	Komputer yang diperuntukkan khusus dalam pengontrolan proses manufaktur

2.5.5 Komputer Generasi Kelima

Seperti halnya komputer generasi keempat, pada generasi kelima ini lebih terfokus pada peningkatan kepadatan chip yang sangat besar hingga jutaan transistor. Teknologi VLSI (*very large scale integration*) meningkatkan kepadatan teknologi LSI pendahulunya. Pembuatan komputer pribadi (*personal computer, PC*) mulai memanfaatkan dengan baik keunggulan teknologi VLSI ini.

Beberapa kontribusi utama komputer generasi-5 adalah:

1. Penggunaan teknologi VLSI dan konsep *artificial intelligence. Expert system, pattern recognition, voice recognition, signaturing capturing and recognition*, robot yang dikontrol dengan mikroprosesor dan sejumlah pengembangan dalam bidang komputer merupakan keistimewaan tersendiri.
2. Perkembangan komputer profesional yang pesat.

Tabel 2.8 Fitur-fitur komputer generasi kelima

No	Nama Fitur	Tipe	Keterangan
3	Sistem Server	Sistem cepat dan kapasitas besar	Menghemat sumber daya pada <i>client system</i> .
4	<i>Embedded system</i>	Produk berbasis Mikrokontroler mikrokontroler	<i>Dedicated intelligent</i> mengontrol peralatan-peralatan instrumentasi termasuk peripheral.
5	Multimedia	Menggabungkan data, suara, dan gambar	Aplikasi baru seperti hiburan, pendidikan, dan lain-lain dll.
6	Internet dan email	Pemakaian komputer berbasis internet	Semua memungkinkan dari rumah mulai dari belajar sampai belanja.

No	Nama Fitur	Tipe	Keterangan
1	Komputer Portable	Membantu eksekutif senior	Rekayasa khusus menawarkan komputer yg yang sangat ringan, operasi baterai dan ketahanan penggunaan sekalipun dalam perjalanan .
2	<i>Networking</i>	Hubungan/link komputer	<i>Sharing sumber daya hardware/software dan komunikasi elektronik.</i>

RINGKASAN

Struktur komputer menunjukkan interkoneksi internal. Struktur fungsional menunjukkan blok-blok fungsional dan hubungan antara blok tersebut. Struktur fisik menunjukkan modul-modul fisik dan interkoneksi antara modul tersebut.

Fungsi komputer menunjukkan tingkah laku komputer tersebut. Keluaran komputer merupakan fungsi dari masukannya. Pada level menyeluruh (yaitu sebagai sebuah sistem), fungsi ini sebagai pengeksekusian program.

Sebuah komputer harus mempunyai keunggulan dalam lima dimensi yaitu: kinerja, kapasitas, harga, maintainability, dan kenyamanan pengguna. Tiga faktor yang berkontribusi bagi superioritas desain komputer adalah: teknologi, konsep, dan teknik yang digunakan. Teknologi secara tetap dan pasti mengembangkan hampir semua aspek komputer yaitu: CPU, memori, divais perangkat peripheral, interkoneksi jalur, software, dan lain-lain.

Di awal pengembangan komputer dahulu dicapai dengan kerjasama proyek antara universitas dengan dana yang bersumber dari pemerintah. Kebutuhan akan kalkulasi yang cepat oleh militer Amerika Serikat mempunyai kontribusi yang penting dalam penemuan komputer dahulu, sedangkan pengembangan saat ini banyak didasari oleh kebutuhan industri. Sejarah komputer dicakup dalam tiga tipe komputer yaitu: komputer mekanik, komputer elektromekanik, dan komputer elektronik. Komputer elektronik selanjutnya dikelompokkan ke dalam lima generasi komputer.

Stored program concept yang diperkenalkan oleh Von Neumann adalah merupakan basis atau dasar bagi hampir semua komputer yang didesain saat ini.

SOAL-SOAL ULANGAN

1. Dalam desain suatu komputer, maka hal-hal yang perlu dipertimbangkan adalah (1) Kinerja yang diharapkan, (2) Target aplikasi (bisnis/saintifik/server/client, dan lain-lain.), (3) Rentang harga yang diinginkan dan (4) _____.
2. Struktur komputer menunjukkan struktur internal komputer tersebut. Dua aspek struktur komputer adalah: (1) struktur _____, dan (2) struktur _____.
3. Struktur _____ menentukan blok-blok dan hubungan antara blok tersebut. Struktur ini merupakan faktor penting untuk mendesain komputer seperti yang tercermin dari pada kinerjanya.
4. Struktur _____ menetapkan modul fisik dan hubungan di antaranya. Struktur ini tidak mempengaruhi kinerja tetapi berpengaruh pada keandalan dan biaya.
5. Ada lima dimensi untuk mengukur keunggulan komputer yaitu (1) _____, (2) _____, (3) harga, (4) maintainability dan (5) kenyamanan pengguna
6. Dalam perkembangannya, tiga arsitektur dasar CPU yang dikenal adalah: (1) arsitektur CPU berbasis akumulator, (2) _____, dan (3) _____.
7. Dalam perkembangannya, tiga unit kontrol yang dikenal adalah: (1) *hardwired control unit*, (2) _____, dan (3) _____.
8. Dalam perkembangannya, tiga teknik I/O yang dikenal adalah: (1) *polling (programmed I/O)*, (2) _____, dan (3) _____.
9. Secara umum, kinerja suatu subsistem dapat ditingkatkan dengan berbagai teknik seperti: (1) paralelisme dengan duplikasi hardware, (2) paralelisme dengan aksi mendahului (lebih dulu), (3) _____, dan (4) _____.
10. Dalam perkembangannya perkembangannya, software sistem yang dikenal adalah: (1) bahasa mesin, (2) bahasa rakitan (*assembly*), (3) sistem operasi, (4) bahasa tingkat tinggi, (5) sistem basis data, (6) sistem pakar dan (7) _____.
11. Dalam perkembangannya era teknologi komputer modern yang dikenal adalah: (1) teknologi tabung hampa (*vacuum tube*), (2) _____, dan (3) _____.

12. *Stored Program Concept*, memori maknetik magnetik sebagai memori utama, aritmetika biner titik-mengambang adalah merupakan ciri utama komputer teknologi _____.
13. Sistem operasi, *multiprogramming*, *compiler*, hard disk maknetik-magnetik, aritmetika biner titik-mengambang, minicomputer adalah merupakan ciri utama komputer teknologi _____.
14. *Multiprocessing*, memori semikonduktor, memori virtual, memori cache, supercomputer adalah merupakan ciri utama komputer teknologi _____.
15. Konsep RISC, mikrocomputer, kontrol proses, workstation merupakan ciri utama komputer teknologi _____.
16. *Networking*, sistem server, multimedia, *embedded system* merupakan ciri utama komputer teknologi _____.

SOAL-SOAL LATIHAN

1. Sebutkan lima generasi komputer disertai dengan fitur utama masing-masing secara singkat.
2. Jelaskan dengan singkat apa yang dimaksud dengan *stored program concept*.
3. Sebuah komputer yang berbasis mikroprosesor 8080 mempunyai ruang memori sebesar 64 KB. Mikrokomputer mempunyai memori fisik 128 KB. Perkirakan teknik apa yang akan digunakan oleh perancang mikrokomputer untuk melipatgandakan memori yang didukung oleh mikroprosesor.

BAB 3

ARSITEKTUR CPU DAN SET INSTRUKSI

Sasaran bab ini:

1. Arsitektur CISC dan RISC
2. Tipe dan Format Instruksi
3. Mode Pengalamatan
4. Tipe Operand

Bab ini membicarakan arsitektur CPU. Ada beberapa parameter penting dari CPU yang berpengaruh langsung pada kinerja dan produktivitas sistem. Arsitektur komputer berfokus pada perancangan set instruksi dan penentuan bit-bit pada setiap rangkaian *field* dalam instruksi. Adanya kelemahan pada desain set instruksi akan memengaruhi secara drastis pemrograman bahasa mesin serta kompiler. Pada bagian ini akan dibicarakan berbagai faktor yang memengaruhi desain set instruksi.

3.1 CISC BANDING RISC

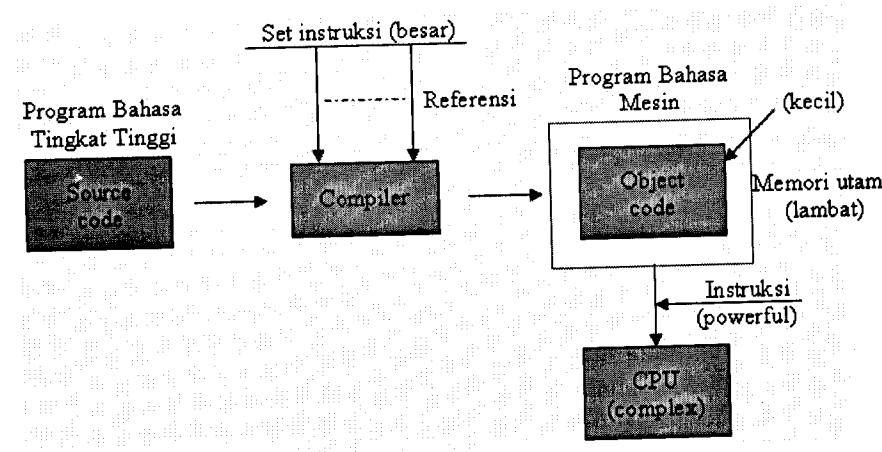
Ada dua konsep populer yang berhubungan dengan desain CPU dan set instruksi:

1. *Complex Instruction Set Computing* (CISC)
2. *Reduce Instruction Set Computing* (RISC)

Semua sistem yang lama (komputer mainframe, komputer mini atau komputer mikro) relatif mempunyai sistem CISC. Walaupun sistem sekarang terdiri atas kedua jenis tersebut. Sistem RISC saat ini lebih populer karena tingkat kinerjanya, dibandingkan dengan sistem CISC. Namun karena biayanya tinggi, sistem RISC hanya digunakan ketika diperlukan kecepatan khusus, keandalan, dan sebagainya.

3.1.1 Trend Teknologi CISC

Umumnya set instruksi pada sistem CISC dibuat efisien dengan memasukkan sejumlah besar *complex instruction* (instruksi kompleks). Tujuannya adalah mengurangi ukuran program yang telah terkompilasi (bahasa mesin) dengan instruksi-instruksi yang terbatas. Pada dasarnya sebuah instruksi kompleks adalah ekivalen dengan tiga atau empat *simple instruction* (instruksi sederhana). Karena program yang telah terkompilasi mempunyai ukuran kecil, kebutuhan memori utama juga kecil. Keuntungan lain dari instruksi kompleks adalah jumlah instruksi di dalam sebuah program (terkompilasi) lebih sedikit, waktu yang digunakan CPU untuk pengambilan instruksi lebih sedikit. Karena itu kita memperoleh dua keuntungan saat mempunyai instruksi kompleks di dalam set instruksi, yaitu mengurangi harga sistem (penggunaan memori kecil) dan mengurangi waktu eksekusi program. Namun demikian, diperlukan kompiler efisiensi tinggi untuk menggunakan instruksi kompleks lebih sering pada saat translasi program bahasa tingkat tinggi ke program bahasa mesin. Karena itu, software sistem (kompiler) menjadi sangat besar untuk membuat kode objek yang kecil. Ilustrasi skenario CISC dapat dilihat pada Gambar 3.1. Saat ini komputer menggunakan memori semikonduktor sebagai memori utama (dan memori cache) yang lebih murah dan lebih cepat.



Gambar 3.1 Skenario CISC

3.1.2 Kelemahan CISC

Beberapa kelemahan sistem CISC adalah:

- Kompleksitas CPU:** desain unit kontrol (utamanya pendekodean instruksi) menjadi kompleks karena mempunyai set instruksi yang besar.
- Ukuran Sistem dan Biaya:** mempunyai banyak sirkuit hardware menyebabkan CPU menjadi kompleks. Hal ini meningkatkan biaya hardware pada sistem dan juga kebutuhan daya listrik.
- Kecepatan Clock:** karena sirkuit yang besar maka *propagation delay* (tunda propagasi) lebih besar dan karena waktu siklus CPU yang besar sehingga kecepatan clock efektif menurun.
- Keandalan:** Dengan hardware yang besar maka cenderung mudah terjadi kegagalan
- Maintainability:** *Troubleshooting* dan pendekripsi suatu kegagalan mengakibatkan pekerjaan menjadi besar karena besarnya sirkuit yang ada. Penemuan *microprogramming* membantu menurunkan beban tersebut.

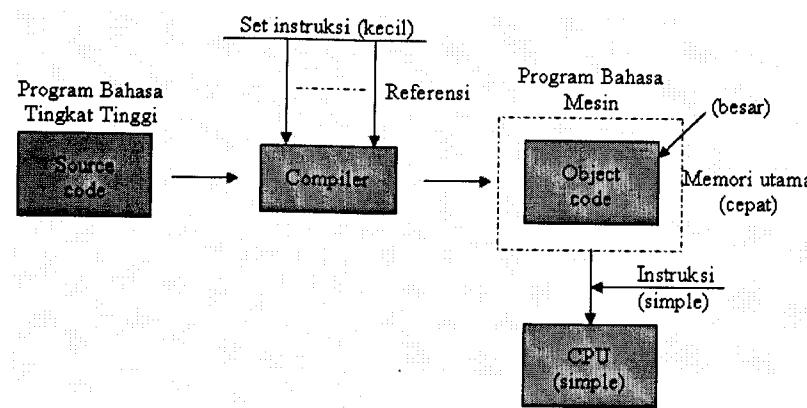
3.1.3 Konsep RISC

Istilah "KISS", yang sering digunakan dalam konsep RISC, merupakan singkatan dari "*Keep it short and simple*". Ilustrasi skenario RISC dapat dilihat pada Gambar 3.2.

Arsitektur RISC mempunyai fitur sebagai berikut:

1. Instruksinya sederhana
2. Set instruksi kecil
3. Panjang instruksinya sama untuk semua instruksi
4. Register untuk penyimpanan operand jumlahnya besar
5. Arsitektur *Load/Store*: Operand untuk instruksi aritmetika seperti "ADD" tersedia di register, bukan di memori. Hasil instruksi "ADD" juga disimpan di register, bukan di memori. Jadi instruksi "LOAD" akan mendahului instruksi "ADD" dan instruksi "STORE" akan mengikuti instruksi "ADD", jika diperlukan. Karena itu kompiler akan memberikan banyak instruksi "LOAD" dan "STORE".
6. Eksekusi instruksi yang lebih cepat (memberikan kecepatan siklus instruksi rata-rata satu clok per instruksi). *Pipeline* instruksi,

memori cache internal (*built-in*) dan arsitektur *superscalar* adalah yang termasuk dalam CPU supaya rata-rata satu instruksi menghasilkan pipeline untuk setiap clock.



Gambar 3.2 Skenario RISC

CPU RISC yang berbasis mikroprosesor maupun yang non-mikroprosesor hingga saat ini telah didesain dan dipasarkan. Berikut beberapa CPU RISC:

1. IBM RS/6000 atau arsitektur POWER
2. Keluarga Sun's SPARC
3. HP's PA (precision architecture)
4. Keluarga Motorola 88000
5. Intel 860
6. Seri MIPS
7. PowerPC

3.1.4 Mikroprosesor RISC

Pabrik mikroprosesor dari dulu tidak memberikan prioritas pengembangan mikroprosesor. Salah satu alasannya adalah ketersediaan kompatibilitas (pada mikroprosesor baru) terhadap mikroprosesor sebelumnya yaitu CPU tipe CISC. Pada umumnya mikroprosesor keluarga Intel (8008 sampai Pentium 4) adalah tipe CISC kecuali Intel 860. Keluarga Motorola 88000 termasuk dalam tipe RISC. Sedangkan PowerPC merupakan CPU tipe

RISC yang dikembangkan bersama oleh IBM, Motorola, dan Apple. Saat ini telah didesain untuk menyediakan mikroprosesor RISC yang murah untuk pengguna.

3.2 DESAIN SET INSTRUKSI

Pekerjaan yang paling signifikan/penting dan kompleks dalam mendesain komputer adalah membuat set instruksi. Untuk mencapai tujuan tersebut, sejumlah pertanyaan berikut harus terjawab. Berapa banyak Instruksi yang diperlukan? Jenis instruksi apa saja yang harus disertakan di dalam set instruksi? Komputer era sebelumnya tidak melakukan perencanaan set instruksi. Kelemahan desain set instruksi adalah mereka secara drastis memengaruhi ruang memori utama karena panjang program (bahasa mesin). Karena itu, dalam desain yang baik, perencanaan awal set instruksi memungkinkan kompiler membuat kode objek yang kompak (tersusun baik dan padat) tersimpan pada ruang memori.

Seorang arsitek komputer harus mempertimbangkan aspek-aspek berikut sebelum menyelesaikan set instruksi:

1. **Kenyamanan pemrograman:** Jumlah instruksi; pemrogram lebih suka mempunyai sebanyak mungkin instruksi supaya operasi yang tepat dapat dikerjakan oleh rangkaian instruksi. Tetapi mempunyai terlalu banyak instruksi dalam set instruksi menghasilkan desain unit kontrol yang kompleks. Pendekodean instruksi memerlukan sirkuit dan waktu yang besar.
2. **Pengalaman yang fleksibel:** Pemrogram senang jika memungkinkan semua mode pengalaman operand ada di dalam arsitektur. Hal ini memberikan fleksibilitas yang banyak kepada pemrogram, walaupun desain unit kontrol menjadi kompleks.
3. **Jumlah General Purpose Register (GPR):** Jika CPU mempunyai register yang banyak, pemrogram memperoleh pemrosesan dan transfer data yang cepat. Tetapi, biaya perangkat keras CPU meningkat dengan banyaknya GPR.
4. **Target segmen pasar:** Sasaran bidang aplikasi untuk komputer memerlukan operasi-operasi khusus untuk pemrosesan data yang efisien. Komputer saintifik harus mempunyai aritmetika *floating-point* yang tingkat presisinya baik/tidak terlalu jelek. Sedangkan komputer

bisnis harus mendukung aritmetika desimal, dan komputer hiburan harus mempunyai operasi-operasi multimedia.

5. **Kinerja sistem:** Jika sebuah program mempunyai instruksi sedikit, kinerja sistem meningkat karena waktu yang digunakan oleh CPU dalam pengambilan instruksi berkurang. Untuk program yang pendek, instruksi yang digunakan harus instruksi kompleks. Jadi, instruksi tunggal harus dapat melakukan beberapa mikrooperasi. Pemrogram menyadari hal ini mengurangi ukuran program. Tapi di sisi lain menambah kompleksitas unit kontrol dan waktu eksekusi instruksi. Konsep modern arsitektur RISC tidak mendukung instruksi-instruksi kompleks, walaupun semua komputer lama yang berbasis CISC menggunakan instruksi-instruksi kompleks.

Secara tradisional, superioritas suatu komputer ditentukan pada basis set instruksinya. Jumlah total instruksi dan ketangguhan yang dimilikinya menjadi sangat penting karena dua faktor tersebut berkontribusi pada efisiensi komputer. Program yang efisien adalah bila program itu pendek dan menempati ruang memori yang sedikit. Waktu eksekusi juga merupakan faktor kunci.

Kecenderungan sekarang adalah mengikuti penggunaan instruksi-instruksi sederhana yang menghasilkan unit kontrol yang sederhana. Sirkuit CPU lebih penting dari ukuran memori. Jadi, kecepatan CPU meningkat pada pemrosesan instruksi dari arsitektur RISC.

Pemilihan set instruksi untuk suatu komputer bergantung pada cara CPU disusun. Secara tradisional, ada tiga organisasi CPU dengan instruksi-instruksi spesifik tertentu:

1. CPU berbasis akumulator
2. CPU berbasis register
3. CPU berbasis stack

3.2.1 Bahasa Rakitan (Assembly)

Bahasa rakitan dalam bentuk simbolik bahasa mesin. Program-program bahasa rakitan ditulis dengan singkatan-singkatan yang pendek yang disebut *mnemonic*. Mnemonic adalah sebuah singkatan yang merepresentasikan instruksi mesin aktual. Pemrograman bahasa rakitan adalah penulisan instruksi-instruksi mesin dalam bentuk mnemonic, di mana setiap

instruksi mesin (nilai biner atau hex) digantikan oleh sebuah mnemonic. Jadi jelas penggunaan mnemonic lebih mempunyai arti daripada nilai biner atau hex, yang akan membuat pemrograman pada level rendah ini lebih mudah dan lebih dapat diatur/ditangani dengan baik.

Bahasa rakitan terdiri atas sebuah rangkaian *assembly statement* di mana *statement* (pernyataan-pernyataan) ditulis satu per setiap baris. Setiap baris dalam sebuah program rakitan dipecah menjadi empat medan (*field*): *Label*, *Opcode* (kode operasi), *Operand* dan *Komentar*. Gambar 3.3 menunjukkan format empat-kolom dari sebuah instruksi rakitan.

<i>Label (optional)</i>	<i>Opcode (disyaratkan)</i>	<i>Operand (disyaratkan untuk beberapa instruksi)</i>	<i>Komentar (optional)</i>
-----------------------------	---------------------------------	---	--------------------------------

Gambar 3.3 Format instruksi bahasa rakitan

Label digunakan untuk menyediakan nama-nama simbolik untuk alamat-alamat memori. Label adalah sebuah pengarah (*identifier*) yang dapat digunakan dalam sebuah baris program untuk bercabang ke baris yang diberi *label*. Juga dapat digunakan untuk mengakses data yang menggunakan nama-nama simbolik. Panjang maksimum sebuah label berbeda-beda dari satu bahasa rakitan dengan bahasa rakitan lainnya. Beberapa bahasa rakitan mengizinkan panjangnya hingga 32 karakter, ada pula yang membatasi hingga 6 karakter. Bahasa-bahasa rakitan untuk beberapa prosesor memerlukan sebuah tanda titik dua (:) sesudah penulisan label, tetapi yang lainnya tidak mensyaratkan demikian. Misalnya, bahasa rakitan SPARC memerlukan tanda titik dua setelah label, tetapi bahasa rakitan Motorola tidak. Bahasa rakitan INTEL memerlukan tanda titik dua setelah label kode, tetapi tidak setelah label data.

Medan (*field*) kode operasi (*opcode*) berisi sebuah singkatan simbolik untuk operasi yang akan dikerjakan. Field operand dapat digunakan untuk menetapkan konstanta, label, *immediate data*, register, atau sebuah alamat. Field komentar menyediakan tempat bagi dokumentasi untuk menjelaskan apa yang dikerjakan untuk tujuan *debugging* maupun untuk *maintenance*. Field komentar pada sebagian bahasa rakitan harus didahului

dengan tanda “\” ada juga yang harus didahului tanda “;”. Pada Tabel 3.1 ditunjukkan beberapa mnemonic dari sebuah prosesor sederhana.

Misalkan kita mengambil sebuah instruksi berikut:

AWAL LD X \ salin isi lokasi X ke AC

Label dari instruksi LD X adalah AWAL, yang berarti alamat memori dari instruksi ini. Label tersebut dapat digunakan dalam sebuah program sebagai sebuah acuan seperti yang ditunjukkan pada instruksi berikut:

BRA AWAL \ bercabang/lompat ke statement dengan label AWAL

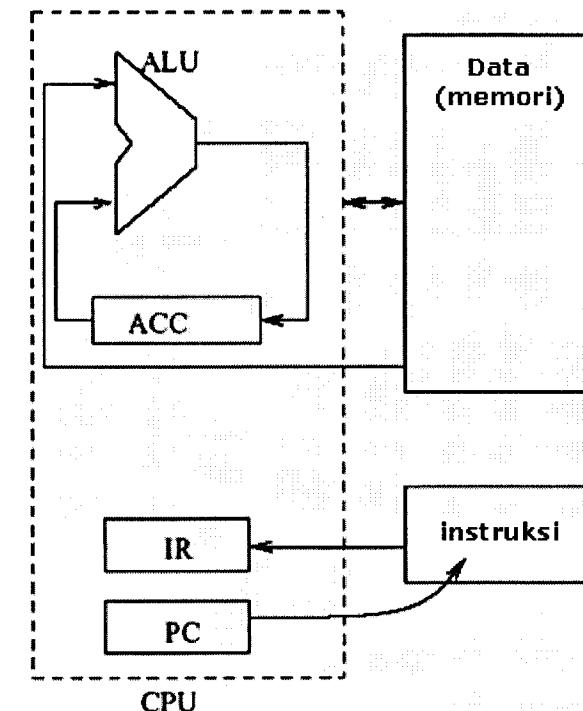
Instruksi lompat akan membuat prosesor lompat ke alamat memori sesuai dengan label AWAL, jadi mengeksekusi instruksi LD X segera setelah instruksi BRA.

TABEL 3.1 Beberapa mnemonic dari sebuah prosesor sederhana

Mnemonic	Operand	Arti Instruksi
LD	x	Baca operand dari memori (lokasi x) ke dalam AC
ST	x	Simpan isi dari AC ke dalam memori (lokasi x)
ADD	x	Tambahkan x ke AC
SUB	x	Kurangi AC dengan x
BRA	adr	Lompat ke instruksi dengan alamat adr
BZ	adr	Lompat ke instruksi adr jika AC = 0

3.2.2 CPU Berbasis Akumulator

Pada mulanya komputer adalah berbasis akumulator. Hal ini merupakan CPU yang sederhana, di mana akumulator berisi satu operand pada instruksi, demikian juga hasilnya disimpan pada akumulator. Isi akumulator disertakan di dalam operasi-operasi aritmetika seperti penjumlahan, pengurangan, dan sebagainya. Hal ini dikenal dengan **mesin satu-alamat**. Arsitektur CPU berbasis akumulator dapat diilustrasikan seperti pada Gambar 3.4. PDP-8 merupakan minicomputer pertama yang mempunyai jenis CPU seperti ini dan digunakan untuk kendali proses dan aplikasi-aplikasi laboratorium. Komputer Mark I juga merupakan komputer khas yang berbasis akumulator. Organisasi CPU ini secara total telah digantikan dengan diperkenalkannya CPU berbasis register yang baru.



Gambar 3.4 Arsitektur CPU berbasis akumulator

Contoh 3.1

Tuliskan sebuah program bahasa rakitan dalam arsitektur CPU berbasis akumulator untuk menyelesaikan statement $X = (A + B) - (C + D)$

Solusi:

- LD C ; salin C ke dalam akumulator
- ADD D ; jumlahkan D dengan isi akumulator dan hasilnya disimpan di akumulator (akumulator berisi C+D)
- ST X ; simpan hasil C+D dalam lokasi X
- LD A ; salin A ke dalam akumulator
- ADD B ; jumlahkan B ke dalam akumulator dan simpan hasilnya dalam akumulator (akumulator berisi A+B)
- SUB X ; perkurangkan isi akumulator dengan X dan simpan hasilnya dalam akumulator
- ST X ; simpan isi akumulator di dalam lokasi memori X

Keuntungan CPU berbasis akumulator adalah:

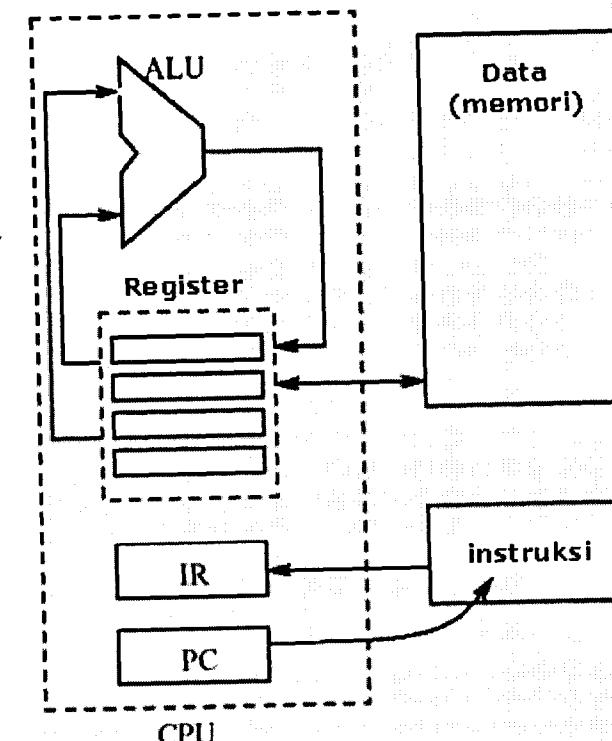
1. Isi akumulator diperuntukkan bagi satu operand, karena itu tidak memerlukan field alamat operand (untuk satu operand) dalam instruksi. Hal ini menghasilkan instruksi yang pendek dan ruang memori yang sedikit. Karena tidak ada field alamat, maka jenis CPU ini mendukung instruksi-instruksi nol-alamat. CPU yang demikian biasanya mempunyai dua jenis instruksi: nol-alamat dan satu-alamat. Instruksi satu-alamat mempunyai satu operand di dalam memori utama dan yang lainnya di dalam akumulator.
2. Siklus instruksi menggunakan waktu yang singkat sebab menghemat waktu dalam pengambilan instruksi karena tidak ada siklus pengambilan operand.

Kekurangan CPU berbasis akumulator adalah:

1. Ukuran program menjadi panjang karena banyak menggunakan instruksi dalam ekspresi-ekspresi kompleks. Karena itu ukuran memori bertambah.
2. Waktu eksekusi program bertambah karena bertambahnya jumlah instruksi dalam program.

3.2.3 CPU Berbasis Register

Pada CPU jenis ini, banyak register yang digunakan sebagai akumulator. Dengan kata lain, ada lebih dari satu akumulator. CPU seperti ini mempunyai organisasi register umum GPR (*general purpose register*). Penggunaan register-register tersebut menghasilkan program yang pendek dengan instruksi yang sedikit. Arsitektur CPU berbasis register dapat diilustrasikan seperti pada Gambar 3.5. Mesin IBM System/360 dan PDP-11 merupakan contoh khas.



Gambar 3.5 Arsitektur CPU berbasis register

Bila ada dua atau tiga operand, biasanya mesin harus mempunyai dua atau lebih register. Secara khas, operasi-operasi aritmetika dan logika memerlukan satu sampai tiga operand. Instruksi dua- dan tiga-operand adalah yang umum karena mereka memerlukan sebuah format instruksi yang relatif pendek. Pada instruksi dua- dan tiga-operand, satu operand seringkali digunakan sebagai sebuah sumber dan sebagai tujuan. Pada instruksi tiga-operand, satu operand digunakan sebagai tujuan dan dua lainnya sebagai sumber.

Pada umumnya mesin (CPU) dengan instruksi dua- atau tiga-operand ada dua jenis: yaitu desain *load-store* dan desain memori-register. Pada mesin *load-store*, hanya instruksi *load* dan *store* yang mengakses memori, selebihnya umumnya melakukan operasi pada register.

Contoh 3.2

Tuliskan sebuah program bahasa rakitan dalam arsitektur CPU berbasis register untuk menyelesaikan statement $X = (A + B) - (C + D)$

Solusi:

```
LD R1, A ; salin A ke dalam register R1
ADD R1, B ; jumlahkan B dengan isi R1 dan hasilnya disimpan di R1
LD R2, C ; salin C ke dalam R2
ADD R2, D ; jumlahkan D ke dalam R2 dan simpan hasilnya dalam R2
SUB R1, R2 ; perkurangkan isi R2 dari R1 dan simpan hasilnya dalam R1
ST R1, X ; simpan hasil di dalam lokasi memori X
```

Dibandingkan dengan Contoh 3.1, terlihat bahwa CPU berbasis register (arsitektur GPR) menghasilkan ukuran program yang lebih pendek daripada CPU berbasis akumulator. Program pada CPU berbasis akumulator memerlukan lokasi memori untuk penyimpanan hasil sementara (parsial). Karena itu, diperlukan akses memori tambahan selama eksekusi program. Jadi, penambahan jumlah register akan menambah efisiensi CPU.

Pada Contoh 3.3 berikut ini diberikan operasi yang sama jika dilakukan pada mesin *load-store*.

Contoh 3.3

Tuliskan sebuah program bahasa rakitan dalam arsitektur CPU berbasis *load-store* untuk menyelesaikan statement $X = (A + B) - (C + D)$

Solusi:

```
LOAD R1, A ; salin A ke dalam register R1
LOAD R2, B ; salain B ke dalam register R2
LOAD R3, C ; salin C ke dalam R3
LOAD R4, D ; salin D ke dalam regisster R4
ADD R5, R3, R4 ; operasi aritmetika  $R5 \leftarrow R3 + R4$ , ( $R5 = C+D$ )
ADD R6, R1, R2 ; operasi aritmetika  $R6 \leftarrow R1 + R2$ , ( $R6 = A+B$ )
SUB R7, R6, R5 ; operasi aritmetika  $R7 \leftarrow R6 - R5$ , ( $R7 = (A+B) - (C+D)$ )
STORE R7, X ; salin isi R7 ke dalam X
```

Keuntungan mesin *load-store* (mesin tiga operand):

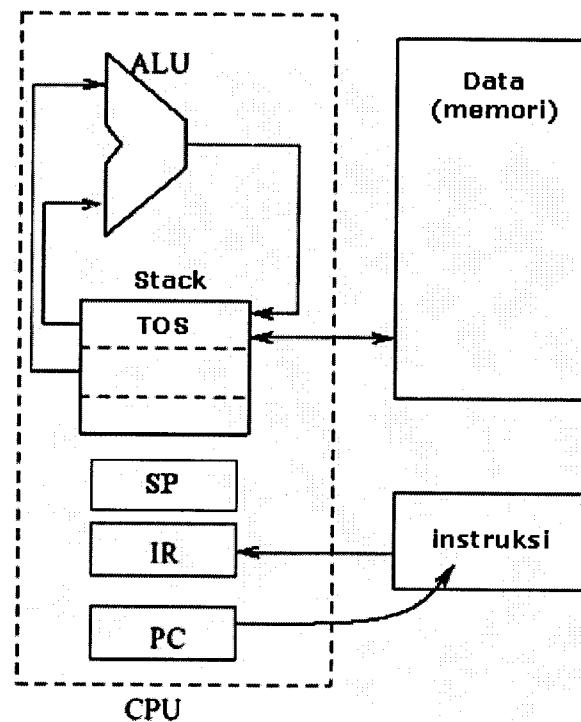
- Sederhana, pengkodean instruksi *fixed length*
- Instruksi menggunakan jumlah siklus yang sama
- Relatif mudah untuk *pipelining*

Kelemahan mesin *load-store*:

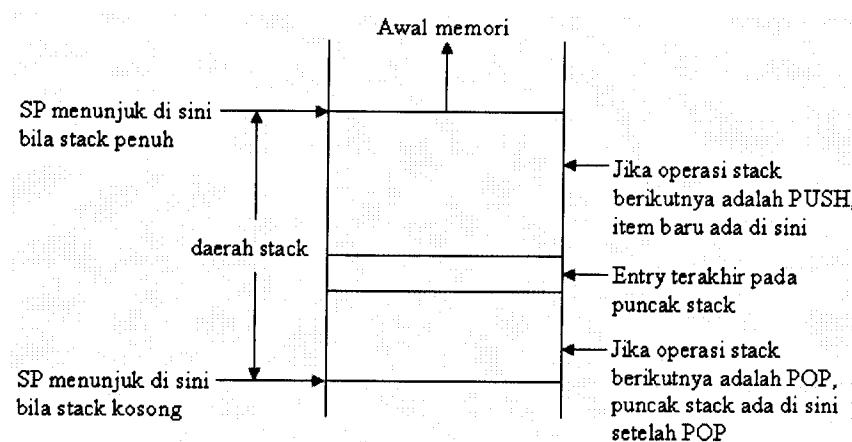
- Jumlah instruksi yg lebih banyak
- Tidak semua instruksi memerlukan tiga operand
- Bergantung pada kompiler yang baik

3.2.4 CPU Berbasis Stack

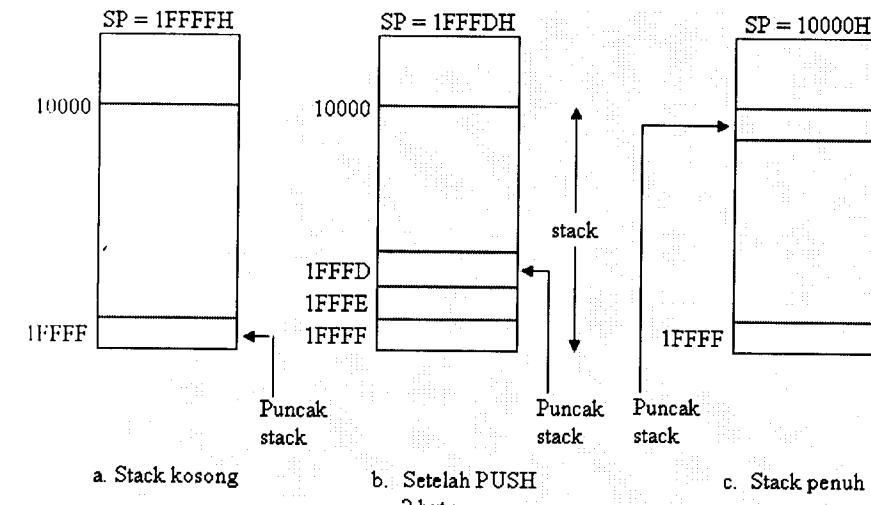
Gambar 3.6 menunjukkan sebuah ilustrasi arsitektur CPU berbasis stack. Stack merupakan daftar yang didorong ke bawah dengan mekanisme akses LIFO (*Last In First Out*). Stack yang menyimpan operand-operand. Penggunaan stack dapat berada di dalam CPU atau merupakan bagian dari memori. Suatu register (atau lokasi memori) digunakan untuk menunjuk ke alamat lokasi kosong pada puncak stack. Register ini dikenal dengan *Stack Pointer* (*SP*). Bila tidak ada yang tersimpan dalam stack, berarti stack kosong dan SP menunjuk ke bagian stack terbawah. Bila suatu item disimpan di dalam stack, maka dinamakan operasi *PUSH*; dan isi SP diturunkan (*decrement*). Bila stack penuh, SP menunjuk ke bagian stack teratas. Bila suatu item diambil dari stack (operasi *POP*), maka SP dinaikkan (*increment*).



Gambar 3.6 Arsitektur CPU berbasis stack



Gambar 3.7 Konsep stack



Gambar 3.8 Operasi stack

Item yang terakhir didorong ke dalam stack, akan keluar pertama jika ada operasi POP berikutnya. Pada CPU berbasis stack, semua operasi oleh CPU dikerjakan pada isi stack. Demikian halnya, hasil suatu operasi juga disimpan pada stack. Gambar 3.7 dan Gambar 3.8 mengilustrasikan konsep dan mekanisme operasi stack. Pada eksekusi suatu instruksi aritmetika seperti ADD, operand-operand teratas yang di-pop. Komputer Burroughs B5000 dan HP 3000 merupakan contoh dari komputer berbasis stack.

Contoh 3.4

Tuliskan sebuah program bahasa rakitan dalam arsitektur CPU berbasis stack untuk menyelesaikan statement $X = (A + B) - (C + D)$

Solusi:

Statement	Isi Stack setelah eksekusi instruksi	Lokasi stack yang diduduki
PUSH A	A	1
PUSH B	A, B	2
ADD	A + B	1
PUSH C	(A + B), C	2

Statement	Isi Stack setelah eksekusi instruksi	Lokasi stack yang diduduki
PUSH D	(A + B), C, D	3
ADD	(A + B), (C + D)	2
SUB	(A + B), (C + D)	1
POP X	Kosong	0

Dari isi stack, dapat dilihat bahwa stack berubah bila beberapa operasi PUSH mendapat tempat. Bila suatu instruksi dieksekusi, operand dipindahkan dari stack dan hasilnya menempati posisi pada puncak stack. Contoh 3.4 menunjukkan bahwa ukuran program untuk komputer berbasis stack lebih besar dibandingkan dengan CPU berbasis register.

Keuntungan CPU berbasis stack adalah:

- Pemrograman mudah/efisiensi kompiler tinggi
- Sangat cocok untuk bahasa-bahasa blok-terstruktur (block-structured language).
- Instruksi tidak mempunyai field alamat; instruksi pendek

Kelemahan CPU berbasis stack adalah:

- Diperlukan sirkuit hardware tambahan untuk implementasi stack.
- Ukuran program meningkat

3.2.5 Panjang Instruksi

Instruksi yang terlalu panjang mempunyai kekurangan:

1. Instruksi menempati ruang memori yang lebih besar, yang meningkatkan kebutuhan memori sistem.
2. Lebar bus data besar atau pengambilan instruksi lebih memakan waktu. Kondisi pertama menambah biaya hardware, sedangkan yang kedua menambah waktu siklus instruksi.

Instruksi yang terlalu pendek mempunyai kekurangan:

1. Terlalu banyak instruksi di dalam program. Karena itu banyak waktu yang terbuang untuk fase pengambilan instruksi (fetch instruction).

2. Ukuran program bertambah, karena itu kebutuhan memori bertambah.

3.2.6 Format Instruksi

Umumnya format instruksi terdiri atas kode operasi dan operand. Suatu instruksi memberikan paling banyak empat informasi pada CPU:

1. Operasi yang akan dikerjakan oleh instruksi.
2. Operand (data) yang harus dioperasikan
3. Lokasi (memori atau register) di mana hasil operasi harus disimpan.
4. Lokasi memori di mana instruksi berikutnya harus diambil

ADD opcode	Alamat operand I	Alamat operand II	Alamat Hasil	Alamat instruksi berikutnya

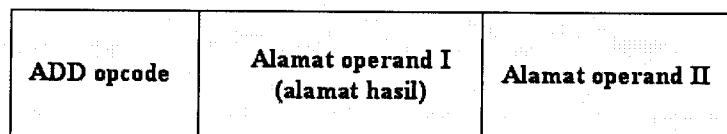
Gambar 3.9 Instruksi empat-alamat

Secara teoritis format instruksi yang menentukan semua empat item untuk sebuah instruksi ADD ditunjukkan pada Gambar 3.9. Praktisnya, beberapa variasi pada format teoretis ini adalah:

1. **Spesifikasi Operand:** Sebagai pengganti pemberian operand secara langsung pada instruksi, lokasi yang ditempati dalam memori utama diidentifikasi dalam instruksi sebagai alamat operand. Hal ini mempunyai dua keuntungan:
 - a) Panjang field operand dalam instruksi umumnya lebih pendek dari operand. Karena itu, menghemat ruang dalam instruksi.
 - b) Pemrogram mempunyai fleksibilitas dalam menempatkan operand pada: memori utama, register CPU, instruksi, port I/O, dan sebagainya.
2. **Alamat Hasil:** Sebagai pengganti penyimpanan dalam suatu lokasi terpisah, umumnya disimpan dalam alamat operand. Kekurangannya, operand awal digantikan oleh hasil. Pemrogram sebaiknya berhati-hati dalam menangani masalah tersebut. Jika operand awal diperlukan oleh program untuk instruksi selanjutnya, salinannya akan ditahan dalam beberapa lokasi lain sebelum instruksi dihadapi oleh CPU dan data rusak.

3. Alamat Instruksi Berikutnya: Pada mayoritas kasus, instruksi berikutnya yang diperlukan secara fisik adalah satu instruksi berikutnya setelah instruksi sekarang. Pada beberapa kasus terbatas, instruksi yang diperlukan berikutnya bukan secara fisik instruksi berikutnya. Hal ini secara tidak langsung kebutuhan/persyaratan field untuk alamat instruksi berikutnya akan menimbulkan pemborosan ruang dan penggunaan panjang instruksi yang tidak efisien. Karena itu, diasumsikan bahwa instruksi yang diperlukan berikutnya adalah instruksi berikutnya secara fisik. Tetapi dalam beberapa kasus, instruksi pencabangan (lompat) disisipkan di antara instruksi sekarang dan (secara fisik) instruksi berikutnya. Instruksi pencabangan menetapkan alamat instruksi berikutnya.

Format instruksi ADD ditunjukkan pada Gambar 3.10 digabungkan dengan variasi di atas. Beberapa komputer menyimpan hasil dalam lokasi operand pertama, sedangkan yang lainnya pada lokasi kedua. Desain unit kontrolnya disesuaikan.



Gambar 3.10 Format umum instruksi ADD

Contoh 3.4

Medan panjang instruksi dan medan alamat operand masing-masing 36 bit dan 14 bit. Jika instruksi dua-operand yang digunakan sebanyak 240, berapa banyak instruksi satu-operand yang memungkinkan?

Solusi:

Panjang instruksi = 36 bit

Sebuah instruksi dua-operand membutuhkan 28 bit untuk alamat operand (2×14). Karena itu ukuran opcode = $36 - 28 = 8$ bit

Total jumlah instruksi yang mungkin = $2^8 = 256$

Jumlah instruksi satu-operand = $256 - 240 = 16$

3.2.7 Lokasi Operand

Ada beberapa pilihan dalam menempatkan operand (lokasi operand) yaitu pada: memori utama, register CPU, I/O port dan pada instruksi itu sendiri. Membiarkan operand dalam register CPU lebih efektif daripada mengambilnya dari memori utama karena waktu akses register CPU lebih singkat. Hasil ini mengurangi waktu siklus instruksi. Penempatan operand dalam instruksi digunakan untuk instruksi-instruksi khusus saja. Isi suatu port dapat digunakan sebagai operand, demikian pula isi suatu lokasi memori. Ada beberapa instruksi yang tidak mempunyai operand. Contoh yang khas adalah instruksi HALT dan NOOP. Ada beberapa instruksi yang berfungsi untuk menguji status komponen-komponen hardware seperti register, flip-flop, lokasi memori, dan sebagainya. Pada kasus ini, tidak terdapat operand. Demikian pula ada beberapa instruksi yang hanya mencari sejumlah sinyal eksternal.

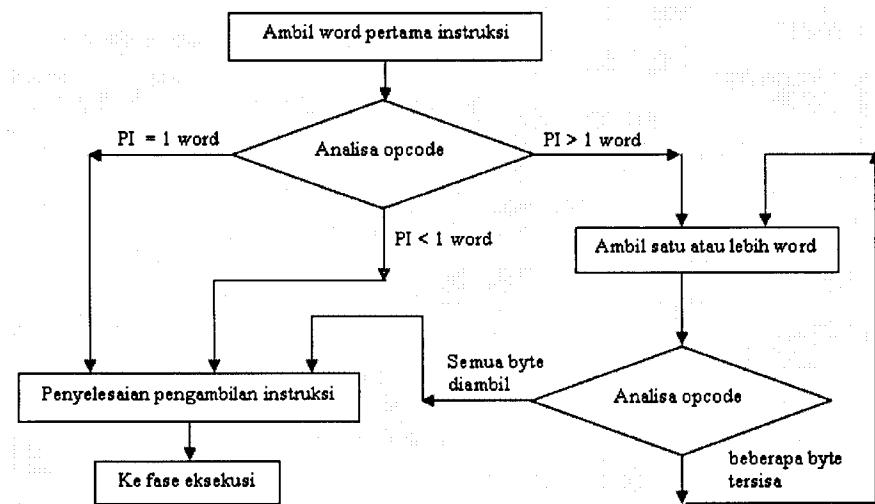
3.2.8 Lokasi Hasil Pemrosesan

Ada beberapa pilihan dalam menyimpan hasil pemrosesan instruksi yaitu: memori utama, register CPU, port keluaran, dan sebagainya. Beberapa instruksi seperti HALT dan NOOP secara eksplisit tidak mempunyai hasil untuk disimpan, sedangkan beberapa instruksi hanya men-set atau me-reset flip-flop atau register.

3.2.9 Variasi Panjang Instruksi

Jika semua instruksi mempunyai panjang yang sama, akan menyebabkan pemborosan ruang memori. Hal ini karena ada beberapa instruksi sederhana seperti HALT, NOOP dan sebagainya yang hanya mempunyai opcode dan karena itu sisa ruang yang ada pada instruksi tersebut tidak terpakai. Hal ini menimbulkan pemborosan waktu pada fase pengambilan instruksi lengkap. Karena itu secara praktis, instruksi-instruksi komputer panjangnya tidak sama. Beberapa di antaranya ada yang pendek (satu byte, dua byte) dan yang lainnya ada yang panjang (tiga byte dan lebih). Opcode instruksi umumnya mengkodekan informasi mengenai panjang instruksi. Dengan melihat pada opcode, CPU dapat memutuskan operasi-operasi baca memori tambahan yang dibutuhkan untuk pengambilan instruksi seutuhnya.

Pemrogram senang jika mempunyai beragam panjang instruksi untuk CPU seperti memberikan fleksibilitas kepada dan juga menghemat ruang memori. Namun, terdapat kerugian pada sistem ini karena desain CPU menjadi kompleks. Sebaiknya mampu melakukan perluasan dan penciutan urutan pengambilan instruksi yang berdasarkan pada opcode seperti yang ditunjukkan pada Gambar 3.11.



Gambar 3.11 Urutan pengambilan instruksi pada instruksi dengan panjang bervariasi

3.2.10 Urutan Data Little-Endian dan Big-Endian

Ada dua metode konvensi yang diikuti untuk penempatan informasi dalam memori dan pengalamanan, yaitu: penetapan dengan *big-endian* dan *little-endian*. Anggap kita mempunyai sebuah informasi 32-bit 12345678 (hexa) yang akan disimpan di dalam lokasi memori 1000 ke atas. Karena terdapat 4-byte, maka informasi menempati alamat 1000 sampai 1003. Pada metode *big-endian*, msB (*most significant byte*) disimpan pada alamat bawah dan lsB (*least significant byte*) disimpan pada alamat atas. Pada metode *little-endian*, lsB disimpan pada alamat bawah dan msB disimpan pada alamat atas. Susunan alamat-alamat memori dan byte diperlihatkan pada Gambar 3.12.

Alamat	→ 1000	1001	1002	1003
Big-endian	12	34	56	78
Little-endian	78	56	34	12

Gambar 3.12 Penyimpanan 12345678 Hexa dengan little-endian dan big-endian

Jika kita mempunyai data 64-bit (8-byte), maka 4-byte sisanya dilanjutkan lagi dari alamat 1004 ke atas dengan menggunakan cara yang sama. Beberapa komputer hanya dapat menangani satu metode dan beberapa yang lainnya dapat menangani kedua metode tersebut.

Metode *big-endian* lebih natural pada sebagian besar orang karena lebih mudah untuk membaca posisi hex. Dengan mempunyai high-order byte yang ada di awal, Anda dapat selalu menguji apakah sebuah bilangan positif atau negatif dengan melihat byte pada offset nol (bandingkan pada *little-endian* di mana Anda harus mengetahui berapa panjang bilangan yang ada dan Anda harus melakukan skip terhadap sejumlah byte untuk mendapatkan posisi informasi bit tanda tersebut). Mesin *big-endian* menyimpan integer dan string dengan urutan yang sama dan tentunya lebih cepat dalam operasi-operasi string. Pada umumnya *bitmapped graphic* dipetakan dengan skema "*most significant bit* sebelah kiri" yang artinya bekerja dengan elemen-elemen grafis yang lebih besar dari satu byte akan dapat ditangani dengan arsitektur ini dengan baik. Hal ini merupakan keterbatasan kinerja untuk mesin-mesin *little-endian* karena mereka harus membalik dulu urutan byte bila bekerja dengan objek grafis yang besar.

Namun, *big-endian* juga mempunyai kelemahan. Konversi dari sebuah alamat integer 32-bit menjadi alamat integer 16-bit harus melakukan penambahan. Aritmetika presisi tinggi pada mesin *little-endian* lebih cepat dan lebih mudah. Umumnya arsitektur yang menggunakan metode *big-endian* tidak membolehkan word ditulis pada area alamat non-word (misalnya jika sebuah word adalah 2 byte atau 4 byte, dia harus selalu dimulai pada alamat byte nomor genap). Hal ini memboroskan tempat. Arsitektur *little-endian* seperti pada INTEL, membolehkan pembacaan dan penulisan alamat ganjil, yang membuat pemrograman pada mesin ini lebih mudah. Catatan bahwa INTEL harus menambahkan suatu instruksi untuk membalik urutan byte di dalam register.

3.2.11 Tipe Instruksi

Instruksi-instruksi diklasifikasikan dalam tipe yang berbeda berdasarkan faktor-faktor berikut:

1. **Opcode:** kode operasi yang harus dikerjakan oleh instruksi
2. **Data:** tipe data: biner, desimal dan sebagainya
3. **Lokasi operand:** memori, register, dan sebagainya
4. **Pengalamanan operand:** metode penentuan lokasi operand (alamat)
5. **Panjang instruksi:** satu byte, dua byte, dan sebagainya
6. **Jumlah medan alamat:** nol alamat, satu alamat, dua alamat, dan sebagainya.

Tidak ada dua komputer yang mempunyai set instruksi yang sama. Hampir setiap komputer mempunyai beberapa instruksi yang unik yang menarik pemrogram. Arsitek komputer memberikan perhatian dalam pembentukan set instruksi karena melibatkan pemrogram dan mesin komputer. Instruksi-instruksi dapat diklasifikasikan ke dalam delapan jenis:

1. **Instruksi transfer data:** instruksi ini menyalin data dari satu register/lokasi memori ke yang lainnya.
2. **Instruksi aritmetika:** instruksi ini melakukan operasi-operasi aritmetika.
3. **Instruksi logika:** instruksi ini melakukan operasi-operasi logika Boolean.
4. **Instruksi transfer kontrol:** instruksi ini melakukan modifikasi/mengubah urutan eksekusi program.
5. **Instruksi I/O:** instruksi ini melakukan transfer informasi antara peripheral eksternal dan inti sistem (CPU/memori).
6. **Instruksi manipulasi string:** instruksi ini melakukan manipulasi *string byte, word, double word*, dan sebagainya.
7. **Instruksi Translate:** instruksi ini melakukan konversi data dari satu format ke format lain.
8. **Instruksi kontrol prosesor:** instruksi ini melakukan kontrol operasi prosesor.

Pada Tabel 3.2 diberikan beberapa contoh instruksi untuk setiap jenis instruksi. Karena pabrik komputer mempunyai dokumentasi/notasi instruksi

masing-masing yang saling berbeda, maka diberikan mnemonic sederhana untuk mewakili secara komprehensif.

TABEL 3.2 Contoh jenis-jenis instruksi

No.	Tipe	Instruksi	
		Nama	Aksi
1	Transfer data	MOVE	Mentransfer data dari lokasi sumber ke lokasi tujuan
		LOAD	Mentransfer data dari lokasi memori ke register CPU
		STORE	Mentransfer data dari register CPU ke lokasi memori
		PUSH	Mentransfer data dari sumber ke stack (puncak)
		POP	Mentransfer data dari stack (puncak) ke tujuan
		XCHG	Tukar; menukar isi sumber dan tujuan
		CLEAR	Reset tujuan dengan semua bit '0'
2	Aritmetika	SET	Set tujuan dengan semua bit '1'
		ADD	Jumlah; hitung jumlah dari dua operand
		ADC	Jumlah dengan carry; hitung jumlah dari dua operand dan bit 'carry'
		SUB	Kurang; hitung selisih dua bilangan
		SUBB	Kurang dengan borrow; hitung selisih dengan 'borrow'
		MUL	Perkalian; hitung hasil kali dari dua operand
		DIV	Pembagian; hitung hasil bagi dan sisa pembagian dari dua bilangan
		NEG	Negate; ganti tanda operand
		INC	Increment; tambahkan 1 pada operand
		DEC	Decrement; kurangkan 1 dari operand
		SHIFT A	Shift arithmetic; geser operand (ke kiri atau kanan) dengan tanda

No.	Tipe	Instruksi	
		Nama	Aksi
3	Logika	NOT	Komplemenkan (komplemen 1) operand
		OR	Lakukan operasi logika OR pada operand
		AND	Lakukan operasi logika AND pada operand
		XOR	Lakukan operasi logika 'exclusive-OR' pada operand
		SHIFT	Geser operand (ke kiri atau kanan) isi bit kosong dengan '0'
		ROT	Rotasi; geser operand (ke kiri atau kanan) dengan berputar
		TEST	Uji kondisi yang ditetapkan dan pengaruh flag yang relevan
4	Transfer kontrol	JUMP	Branch; masukkan alamat yang ditetapkan ke PC; cabang tak bersyarat (<i>unconditional transfer</i>)
		JUMPIF	Bercabang dengan kondisi; masukkan alamat yang ditetapkan ke PC hanya jika kondisi yang ditetapkan terpenuhi; conditional transfer
		JUMPSUB	CALL; simpan 'program control status' yang sekarang dan masukkan alamat yang ditetapkan ke PC
		RET	RETURN; <i>unsafe (restore)</i> 'program control status' (dari stack) ke PC dan register(flag yang relevan lainnya)
		INT	Interupsi; melakukan interupsi software; simpan 'status kontrol program' (ke stack) dan masukkan alamat sesuai dengan kode yang ditetapkan (<i>vector</i>) ke PC

No.	Tipe	Instruksi	
		Nama	Aksi
5	Instruksi Input-output	IRET	<i>Interrupt return</i> ; ambil kembali 'status program kontrol' dari stack ke PC serta register-register dan flag yang relevan lainnya
		LOOP	<i>Iterasi</i> ; turunkan (<i>decrement</i>) isi register dengan 1 dan uji <i>non-zero</i> ; jika tercapai, masukkan alamat yang ditetapkan ke PC
		IN	Input; baca data dari port/divais yang ditetapkan ke register yang ditetapkan atau yang terlibat
		OUT	Output; tulis data dari register yang ditetapkan atau yang terlibat ke suatu port/perangkat
		TEST I/O	Baca status dari subsistem I/O dan set kondisi flag
6	Manipulasi String	START I/O	Sinyal prosesor I/O (atau <i>data channel</i>) untuk memulai program I/O (perintah untuk program I/O)
		HALT I/O	Sinyal prosesor I/O (atau <i>data channel</i>) untuk membatalkan program I/O (perintah untuk operasi-operasi I/O) dalam <i>progress</i>
		MOVS	Salin (move) byte atau word string
		LODS	Salin (Load) byte atau word string
7	Translate	CMPS	Bandingkan byte atau word string
		STOS	Simpan (Store) byte atau word string
		SCAS	Scan byte atau word string
		XLAT	<i>Translate</i> ; ubah kode yang diberikan ke bentuk yang lain dengan <i>table lookup</i>
		HLT	<i>Halt</i> ; hentikan siklus instruksi (pemrosesan)

No.	Tipe	Instruksi
	Nama	Aksi
	STI (EI)	Set interrupt (enable interrupt); men-set interrupt enable flag ke '1'
	CLI (DI)	Clear interrupt (disable interrupt); me-reset interrupt enable flag ke '0'
	WAIT	Penghentian siklus instruksi hingga suatu kondisi terpenuhi (seperti sinyal input menjadi aktif)
	NOOP	No operation; nothing
	CMC	Komplemenkan carry flag
	CLC	Jadikan '0' carry flag
	STC	Jadikan '1' carry flag

3.2.12 Makro dan Subrutin

Makro dan subrutin adalah dua mekanisme khusus yang menghindari pemrograman repetitif. Untuk melakukan suatu pekerjaan khusus pada banyak tempat dalam suatu program dengan data yang berbeda pada setiap tempat, maka dipakai salah satu dari dua mekanisme yang dapat berguna seperti dalam menghindari penulisan rutin pekerjaan yang berulang kali.

Makro adalah suatu rutin yang dapat diminta dalam suatu tempat pada sebuah program dengan hanya mencantumkan namanya (memberikan namanya). Dia merupakan subprogram yang independen dengan sejumlah masukan parameter yang mempunyai nilai yang harus disuplai pada tempat permintaan makro. Sebuah makro dapat diminta pada sejumlah tempat (*multiple*) jika diperlukan. Dalam suatu kode objek program, kode makro disisipkan pada setiap tempat di mana makro diminta. Misalnya, perhitungan akar pangkat dua diperlukan pada sejumlah tempat di dalam suatu program untuk bilangan yang berbeda. Rutin perhitungan akar pangkat dua dapat dibuat seperti sebuah makro dengan nama SQR. Pada contoh berikut, makro SQR diminta pada dua tempat dengan nilai parameter yang berbeda, 1000 dan 2000.

SQR MACRO number

```
.....  
.....  
END MACRO  
.....  
.....  
SQR 1000  
.....  
.....  
SQR 2000  
.....
```

Subrutin adalah suatu program di mana CPU untuk sementara bercabang dari program utama untuk melakukan suatu fungsi khusus. Sebelum memasuki subrutin, program utama menempatkan parameter-parameter pada register atau pada lokasi memori. Dan juga, tempat dalam program utama di mana pencabangan ke subrutin diingat dengan menyimpan nilai *program counter* di dalam memori atau register CPU. Statement CALL dalam program utama menyebabkan pencabangan ke subrutin. Subrutin memperoleh parameter-parameter yang diperlukan dan melakukan operasi. Selanjutnya, subrutin mentransfer kontrol ke program utama. Ini dicapai dengan menggunakan statement RETURN sebagai akhir statement dalam subroutine. Contoh berikut mengilustrasikan konsep subrutin.

MAIN PROGRAM

```
.....  
.....  
CALL SQR  
.....  
.....  
SQR .....  
.....  
.....  
RETURN
```

Program membolehkan mempunyai subrutin yang di dalamnya memanggil subrutin. Keadaan ini disebut *subroutine nesting*.

3.3 MODE PENGALAMATAN

Ada dua cara yang biasa digunakan dalam penempatan operand instruksi yaitu pada lokasi memori utama dan register CPU. Jika operand ditempatkan pada memori utama, alamat lokasi harus diberikan oleh instruksi dalam medan operand. Tidak perlu memberikan alamat secara eksplisit pada instruksi. Banyak metode yang berguna yang dipakai untuk menentukan alamat operand. Mode yang berbeda dalam penentuan alamat operand pada instruksi dikenal dengan *addressing modes* (mode pengalamatan). Suatu komputer bisa saja tidak menggunakan semua mode pengalamatan tersebut. Mode pengalamatan yang populer adalah:

1. Pengalamatan immediate
2. Pengalamatan langsung (absolut)
3. Pengalamatan tak-langsung
4. Pengalamatan tak-langsung register
5. Pengalamatan register
6. Pengalamatan indeks
7. Pengalamatan relatif
8. Pengalamatan *Base* dengan indeks dan *offset*

Mengapa kita perlu begitu banyak mode pengalamatan? Beberapa mode pengalamatan memberikan fleksibilitas pada pemrogram dalam menulis program yang efisien (singkat dan cepat). Berikut adalah tujuan yang memengaruhi arsitek komputer ketika memilih mode pengalamatan:

1. Mengurangi panjang instruksi dengan mempunyai medan yang pendek untuk alamat.
2. Menyediakan bantuan yang tangguh kepada pemrogram untuk penanganan data kompleks seperti pengindeksan sebuah array, kontrol loop, relokasi progam, dan sebagainya.

Mode pengalamatan yang tepat yang digunakan oleh suatu instruksi ditunjukkan pada unit kontrol, ada dua cara yaitu:

1. Medan terpisah dalam instruksi menunjukkan mode pengalamatan yang digunakan, seperti yang diberikan pada Gambar 3.13.
2. Opcode sendiri yang secara eksplisit menunjukkan mode pengalamatan yang digunakan dalam instruksi.

Opcode	Mode pengalamatan	Medan operand I	Medan operand II
--------	-------------------	-----------------	------------------

Gambar 3.13 Medan-medan mode pengalamatan

Komputer yang ada dirancang untuk menggunakan salah satu dari dua teknik tersebut. Penjelasan secara detail dari mode pengalamatan diberikan berikut.

3.3.1 Pengalamatan Immediate

Opcode	Operand
--------	---------

Gambar 3.14 Format instruksi pengalamatan immediate

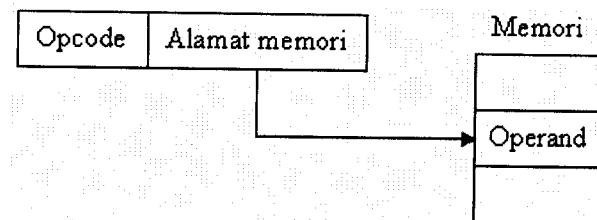
Mode pengalamatan immediate merupakan mode pengalamatan yang tidak melakukan aktivitas pengambilan operand. Pada contoh yang diberikan berikut adalah dalam statement bahasa rakitan. Tanda # digunakan untuk menunjukkan bahwa konstanta yang mengikuti tanda tersebut adalah *immediate operand*.

- MOVE #26, R1 atau MVI R1, 26 → Isikan (load) ekivalen biner 26 ke register R1
- ADD #26, R1 → Tambahkan ekivalen biner 26 ke dalam R1 dan simpan hasilnya pada R1
- CMP #26, R1 atau CMI R1, 26 → Bandingkan isi R1 dengan ekivalen biner 26

Keuntungan: Operand tersedia di dalam instruksi segera setelah pengambilan instruksi berakhir. Karena itu siklus instruksi lebih cepat.

Kelemahan: Nilai operand dibatasi oleh panjang medan operand dalam instruksi. Praktik pemrograman kurang fleksibel karena setiap perubahan nilai operand memerlukan perubahan pada instruksi.

3.3.2 Pengalamatan Langsung



Gambar 3.15 Format instruksi pengalamatan langsung

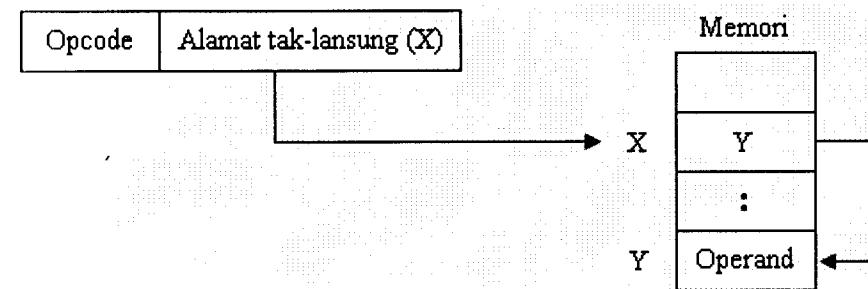
Karena alamat operand secara eksplisit diberikan di dalam instruksi, mode ini disebut mode pengalamatan langsung. Contoh statement bahasa rakitan berikut memberikan ilustrasi mode ini:

- LOAD R1, X → Salin isi lokasi memori X ke dalam register R1
- MOV Y, X → Salin isi lokasi memori X ke dalam lokasi Y. Kedua operand ini menunjukkan penggunaan mode pengalamatan langsung.
- JUMP X → Transfer kontrol program ke instruksi pada lokasi memori X dengan mengisikan X ke PC (X bukan operand, tetapi alamat pencabangan)

Keuntungan: Karena alamat operand tersedia langsung dalam instruksi, maka tidak dibutuhkan langkah kalkulasi alamat operand. Karena itu waktu siklus instruksi berkurang.

Kekurangan: Jumlah bit untuk alamat operand dibatasi oleh medan operand dalam instruksi.

3.3.3 Pengalamatan Tak Langsung



Gambar 3.16 Format instruksi pengalamatan tak-langsung memori

Karena mode pengalamatan tak-langsung dapat melalui sebuah lokasi memori atau register, maka dapat dilakukan dengan dua cara, yaitu:

- Pengalamatan tak-langsung memori: jika sebuah lokasi memori digunakan untuk menyimpan alamat operand
- Pengalamatan tak-langsung register: jika sebuah register digunakan untuk menyimpan alamat operand

Instruksi memberikan alamat lokasi (X), di mana lokasi ini berisi alamat lokasi lain (Y) yang merupakan lokasi operand. Hal ini dapat direpresentasikan sebagai berikut:

$$\langle X \rangle = Y, \langle Y \rangle = \text{operand}$$

Y dikenal sebagai pointer. Nilai Y (alamat) dapat diganti secara dinamis dalam suatu program tanpa mengganti instruksi dengan cara melakukan modifikasi sederhana pada isi lokasi X. Pengalamatan tak langsung yang multilevel dapat dimungkinkan.

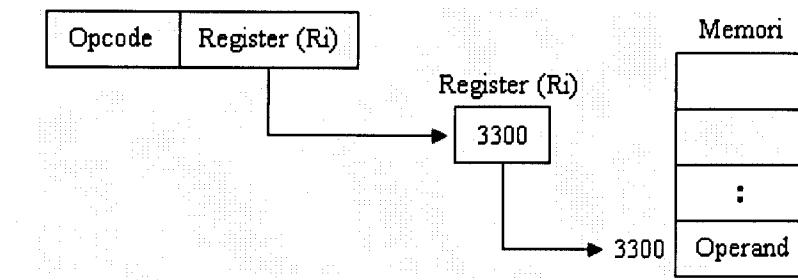
Contoh berikut adalah ilustrasi mode pengalamatan tak langsung.

- MOVE (X), R1 → isi dari lokasi yang mempunyai alamat X disalin ke register R1

Keuntungan: Mempunyai fleksibilitas dalam pemrogram; perubahan alamat selama program berjalan tanpa mengubah isi instruksi.

Kekurangan: Waktu siklus instruksi bertambah karena dua akses memori dibutuhkan untuk sebuah pengalamatan tak langsung single-level.

3.3.4 Pengalamatan Tak Langsung Register

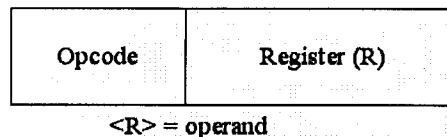


Gambar 3.17 Format instruksi pengalamatan tak langsung register

Pada mode ini, register digunakan untuk menjaga alamat memori dari operand daripada operand itu sendiri. Jadi register bertindak sebagai register alamat memori. Mode ini sangat berguna untuk pengaksesan cepat lokasi memori utama seperti array. Instruksi dengan mode tak-langsung register adalah merupakan bagian dari loop. Pertama, alamat awal dari array disimpan di dalam register. Bila instruksi ditemukan pertama, entry pertama dari array diakses. Kemudian isi register ditambah satu oleh instruksi lain dalam loop, sebelum mendapati instruksi mode tak-langsung register.

Keuntungan: pemanfaatan efektif panjang instruksi, karena nomor register ditentukan dengan sejumlah bit.

3.3.5 Pengalamatan Register



Gambar 3.18 Format instruksi pengalamatan register

Secara konseptual, pengalamatan register mirip dengan pengalamatan langsung kecuali lokasi memori digantikan dengan register untuk menyimpan operand. Instruksi berisi nomor register yang mempunyai operand.

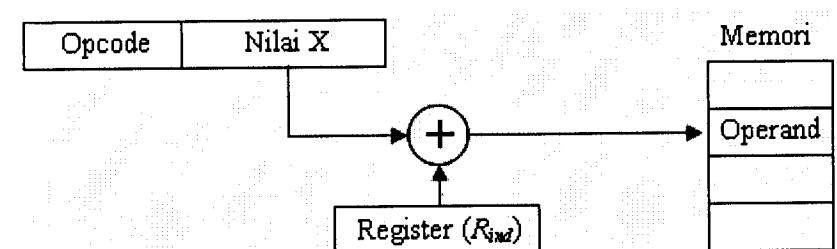
Mode pengalamatan ini sangat berguna untuk suatu program yang panjang dalam penyimpanan hasil-hasil sementara di dalam register daripada di dalam memori. Contoh berikut adalah ilustrasi mode pengalamatan register:

- ADD R1,R2 → jumlahkan isi register R1 dan R2 dan hasilnya disimpan di R1. kedua operand menggunakan pengalamatan register.
- STORE R1, MEM1 → Isi dari register R1 disimpan/disalin ke alamat memori MEM1; operand pertama menggunakan pengalamatan register dan operand kedua menggunakan pengalamatan langsung.

Keuntungan: pengambilan operand lebih cepat tanpa akses memori.

Kelemahan: Jumlah register terbatas dan karena itu utilisasi efektif oleh pemrogram merupakan hal yang esensial.

3.3.6 Pengalamatan Indeks



Gambar 3.19 Format instruksi pengalamatan indeks

Pada mode pengalamatan indeks, alamat operand diperoleh dengan menambahkan sebuah konstanta ke suatu register, yang disebut register indeks. Instruksi ini mengisi register Ri dengan isi lokasi memori yang alamatnya adalah hasil jumlah isi register R_{idx} dan nilai X.

Contoh: LOAD X(R_{idx}), Ri. Instruksi ini menyalin operand alamat hasil penjumlahan nilai X dengan nilai regiter R_{idx} ke dalam register Ri

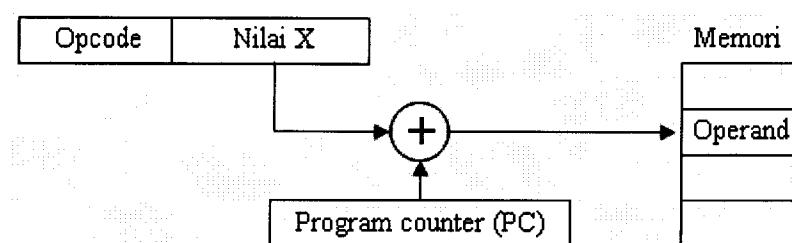
Mode ini berbeda sedikit dengan mode pengalamatan *base register*. Register indeks berisi sebuah *offset* atau perpindahan (*displacement*).

Instruksi berisi alamat yang akan ditambahkan pada offset dalam register indeks, untuk mendapatkan alamat operand efektif.

Umumnya field alamat dalam instruksi memberikan alamat awal array dalam memori. Register indeks berisi ‘*nilai indeks*’ untuk operand yaitu selisih antara alamat awal dan alamat operand. Dengan mengubah nilai register indeks, maka operand dalam array dapat diakses. Umumnya operand-operand (elemen-elemen array) berada dalam lokasi yang berurutan. Mereka diakses dengan increment yang sederhana pada register indeks.

Beberapa CPU mendukung fitur ‘*autoindexing*’, yang melibatkan *auto-increment* (dengan hardware) pada register indeks kapanpun suatu instruksi dengan pengalamatan indeks dieksekusi. Hal ini mengurangi penggunaan instruksi terpisah dalam menambah isi register indeks. Hal ini juga lebih mempercepat aksi serta lebih mengurangi ukuran program, namun memberikan tanggung jawab tambahan ‘*autoindexing*’ pada unit kontrol.

3.3.7 Pengalamatan Relatif



Gambar 3.20 Format instruksi pengalamatan relatif

Pengalamatan Relatif sama seperti pengalamatan indeks kecuali register indeks diganti dengan *program counter* (PC). Instruksi ini mengisi Ri dengan kandungan lokasi memori yang alamatnya adalah hasil jumlah program counter (PC) dan nilai X.

Pada mode ini, instruksi menetapkan alamat operand (lokasi memori) sebagai posisi relatif dari alamat instruksi sekarang yaitu isi PC. Karena itu operand terletak pada ‘jarak pendek’ dari isi PC. Umumnya mode ini

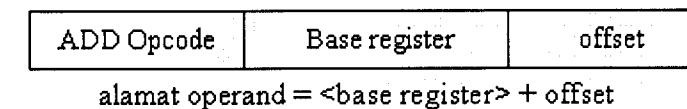
digunakan untuk menetapkan alamat pencabangan dalam instruksi branch, alamat pencabangan berada dekat dengan alamat instruksi.

JUMP + 8 (PC)
JUMP - 8 (PC)

Keuntungan: Jumlah bit dalam medan alamat lebih sedikit.

3.3.8 Pengalamatan Base Register

Mode ini digunakan untuk relokasi program di dalam memori (dari satu area ke area lain). Pada mode pengalamatan *base register*, instruksi tidak berisi alamat. Dia memberikan perpindahan relatif terhadap area memori sekarang ke area memori yang lain, base register diisi dengan alamat base baru. Instruksi tidak perlu dimodifikasi/diubah. Dengan cara ini, keseluruhan program atau suatu segment dari program dapat dipindahkan dari satu area di memori ke yang lain tanpa memengaruhi instruksi, dengan perubahan sederhana isi base register. Hal ini penting untuk sistem multiprogramming karena untuk waktu yang berbeda (*run*), area berbeda di memori tersedia untuk sebuah program. Sebuah CPU dapat mempunyai lebih dari satu base register.



Gambar 3.21 Format instruksi pengalamatan base register

Keuntungan: medan alamat operand dalam instruksi sangat pendek karena hanya memberikan offset (perpindahan); alamat operand dikalkulasi tanpa akses memori.

3.3.9 Pengalamatan Stack

Pada pengalamatan stack (tumpukan), semua operand untuk suatu instruksi diambil dari bagian teratas stack. Instruksi tidak mempunyai medan operand. Misalnya, sebuah instruksi ADD hanya memberikan opcode (ADD). Kedua operand berada dalam stack, di dalam lokasi yang

berurutan. Bila instruksi ADD dieksekusi , dua operand di-pop-off dari stack satu per satu. Setelah penjumlahan, hasilnya di-push ke dalam stack.

Keuntungan: tidak ada medan operand dalam instruksi. Karena itu instruksi pendek.

RINGKASAN

Beberapa parameter CPU mempunyai pengaruh langsung pada kinerja sistem dan produktivitas pemrogram. Pekerjaan yang paling penting dan kompleks dalam desain komputer adalah pembentukan set instruksi. Secara tradisional, superioritas komputer ditetapkan berdasarkan kekayaan set instruksi. Jumlah instruksi total dan ketangguhan instruksi merupakan hal yang sangat penting karena dua faktor tersebut berkontribusi pada efisiensi pemrograman komputer. Set instruksi sistem CISC dibuat tangguh dengan menggabungkan sejumlah besar instruksi yang tangguh. Singkat/kesederhanaan dan kecepatan program tinggi merupakan hal yang dikehendaki.

Kecenderungan modern adalah mengikuti set instruksi yang sederhana untuk perancangan unit kontrol yang lebih sederhana. Dibandingkan dengan ukuran memori, yang lebih penting adalah memberikan sirkuit CPU yang sekecil mungkin. Tujuannya adalah meningkatkan kecepatan CPU untuk pemrosesan instruksi. Arsitektur RISC dicakup oleh fitur berikut: instruksi sederhana, set instruksi yang kecil, panjang instruksi sama untuk semua instruksi, jumlah register yang banyak dan arsitektur LOAD/STORE.

Dahulu banyak komputer mempunyai CPU berbasis akumulator. Komputer sekarang mempunyai CPU berbasis register di mana terdapat banyak register yang semuanya dapat digunakan sebagai fungsi akumulator. CPU berbasis stack jarang, tetapi penting.

Instruksi dapat digolongkan menjadi delapan jenis:

- Instruksi transfer data: instruksi ini menyalin data dari satu register/lokasi memori ke yang lain.
- Instruksi aritmetika: instruksi ini melakukan operasi-operasi aritmetika.
- Instruksi logika: instruksi ini melakukan operasi-operasi logika.
- Instruksi transfer kontrol: instruksi ini melakukan modifikasi urutan eksekusi program.
- Instruksi input/output: instruksi ini melakukan transfer informasi antara peripheral ekternal dan inti sistem (CPU/memori).

- Instruksi manipulasi string: instruksi ini melakukan manipulasi string dari suatu byte, word, double word, dan seterusnya.
- Instruksi translasi: yaitu instruksi yang mengubah data dari satu format ke format lain.
- Instruksi kontrol prosesor: instruksi ini melakukan kontrol terhadap operasi prosesor.

Mode atau cara penetapan alamat operand yang berbeda-beda dalam instruksi dikenal sebagai mode pengalamatan (*addressing mode*). Suatu komputer yang diberikan tidak dapat menggunakan semua mode pengalamatan. Mode pengalamatan yang populer adalah:

1. Pengalamatan immediate
2. Pengalamatan langsung (absolut)
3. Pengalamatan tak-langsung
4. Pengalamatan tak-langsung register
5. Pengalamatan register
6. Pengalamatan indeks
7. Pengalamatan relatif
8. Pengalamatan base dengan indeks dan offset

Tujuan yang memengaruhi arsitek komputer ketika memilih mode pengalamatan:

1. Mengurangi panjang instruksi dengan mempunyai medan yang pendek untuk alamat.
2. Menyediakan bantuan yang tangguh kepada pemrogram untuk penanganan data kompleks seperti pengindeksan sebuah array, kontrol loop, relokasi program, dan sebagainya.

SOAL-SOAL ULANGAN

1. Ada dua konsep populer yang berhubungan dengan desain CPU dan set instruksi yaitu (1) _____, dan (2) _____.
2. Sistem RISC saat ini lebih populer karena tingkat kinerjanya, dibandingkan dengan sistem CISC. Sistem RISC hanya digunakan ketika diperlukan kecepatan khusus, keandalan, dan sebagainya karena _____.

3. Umumnya set instruksi pada sistem CISC dibuat efisien dengan memasukkan sejumlah besar instruksi kompleks. Dua keuntungan mempunyai instruksi kompleks di dalam set instruksi yaitu (1) _____, dan (2) _____.
4. Pada dasarnya sebuah instruksi kompleks adalah ekivalen dengan tiga atau empat _____.
5. Kelemahan penggunaan instruksi kompleks karena desain _____ menjadi kompleks dan mahal.
6. Ciri-ciri atau fitur yang khas pada sistem RISC adalah (1) instruksinya sederhana, (2) set instruksi kecil, (3) semua instruksi panjangnya sama, (4) jumlah register besar untuk penyimpanan operand (5) arsitektur load/store, dan (6) eksekusi instruksi cepat karena siklus instruksi rata-rata _____ per instruksi.
7. Seorang arsitek komputer harus mempertimbangkan sedikitnya lima aspek berikut sebelum menyelesaikan set instruksi yaitu (1) kenyamanan pemrograman, (2) pengalamatan yang fleksibel, (3) jumlah *general purpose register*, (4) target segmen pasar, dan (5) _____.
8. Pemilihan set instruksi untuk suatu komputer bergantung pada cara CPU disusun. Secara tradisional, ada tiga organisasi CPU dengan instruksi-instruksi spesifik tertentu, yaitu (1) _____, (2) _____, dan (3) _____.
9. CPU berbasis akumulator disebut juga dengan mesin _____ alamat.
10. CPU berbasis stack disebut juga dengan mesin _____ alamat.
11. Umumnya format instruksi terdiri atas (1) _____, dan (2) _____.
12. Ada empat pilihan dalam menempatkan operand (lokasi operand) yaitu pada: (1) _____, (2) _____, (3) _____, dan (4) _____.
13. Urutan penyimpanan informasi/data di dalam memori dikenal ada dua yaitu (1) _____, dan (2) _____.
14. Urutan penyimpanan informasi/data di mana lsB (*least significant Byte*) disimpan pada alamat bawah dan msB (*most significant Byte*) disimpan pada alamat atas, disebut _____.
15. Urutan penyimpanan informasi/data di mana msB disimpan pada alamat bawah dan lsB disimpan pada alamat atas, disebut _____.
16. Mode pengalamatan yang membiarkan operand sumber berada di dalam instruksi adalah mode pengalamatan _____.
17. Mode pengalamatan yang alamat operand-nya secara eksplisit diberikan di dalam instruksi disebut mode pengalamatan _____.

18. Mode pengalaman tak langsung terbagi dua yaitu (1) _____ dan (2) _____.

SOAL-SOAL LATIHAN

1. Tipe CPU manakah yang terkecil dan karena itu menjadi murah—CPU berbasis akumulator tunggal atau CPU berbasis register? Berikan alasan jawaban Anda.
2. CPU berbasis register dapat dipandang sebagai CPU berbasis akumulator jamak. Berikan alasan pernyataan ini.
3. Sebuah program mempunyai tiga instruksi lompat dalam tiga lokasi word yang berurutan dalam memori: 0111, 1000, dan 1001. Alamat lompatan masing-masing adalah 1001, 0111, 0111. Dianggap awalnya kita me-load 0111 ke dalam program counter untuk start pada CPU, berapa banyak instruksi dalam lokasi 1000 dibaca (*fetch*) dan dieksekusi?
4. Tuliskan sebuah program untuk mengevaluasi ekspresi $X = (A \times B) + (C \times D)$ untuk:
 - a. CPU berbasis akumulator
 - b. CPU berbasis register
 - c. CPU berbasis stack
5. Set instruksi dari suatu CPU mempunyai instruksi 12-bit. Empat bit diperuntukkan bagi setiap operand field. Mendukung tiga macam instruksi yang berbeda:
 - a. m buah instruksi 0-alamat (*zero-address*);
 - b. n buah instruksi dua-alamat dan
 - c. Sisanya untuk instruksi satu-alamat.
 Berapa banyak kemungkinan instruksi satu-alamat?

BAB 4

ARITMETIKA KOMPUTER

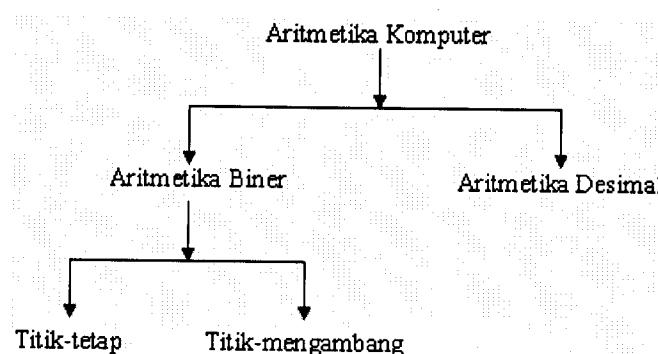
Sasaran bab ini:

1. Representasi Data
2. Penjumlah
3. Penjumlah Paralel
4. Penjumlah-Pengurang
5. Perkalian
6. Algoritma Booth
7. Pembagian
8. Aritmetika Titik-Mengambang

Datapath dalam suatu komputer memainkan satu peranan penting. Datapath berisi ALU, register, penggeser, pencacah, bus dan bagian hardware lainnya serta sirkuit-sirkuit yang diperlukan dalam pemindahan data dan pelaksanaan sejumlah operasi. Komponen-komponen datapath dapat dikonfigurasikan dalam banyak cara (akan dibahas tersendiri). Kecepatan unit datapath bergantung pada algoritma yang digunakan untuk operasi-operasi aritmetika dan desain sirkuit ALU. Pada bagian ini dibahas operasi aritmetika yang dilakukan oleh ALU dan sirkuit-sirkuit yang diperlukan untuk pelaksanaan operasi-operasi dengan sejumlah algoritma yang dapat dipilih. Pengetahuan tentang algoritma-algoritma aritmetika yang ada merupakan hal yang esensial dalam membangun suatu datapath yang efisien. Misalnya, instruksi pembagian dapat diimplementasikan oleh dua algoritma yang berbeda yaitu *restoring* dan *non-restoring*. Algoritma pembagian restoring adalah langsung/tidak berbelit-belit dan mudah diimplementasikan, sedangkan algoritma pembagian non-restoring menyediakan jalan pintas dan menghasilkan desain yang lebih cepat. Pada bagian akhir dari bab ini diberikan operasi bilangan titik-mengambang.

4.1 JENIS-JENIS ARITMETIKA KOMPUTER

Secara teoritis, komputer dapat mengoperasikan berbagai jenis data melalui pemrograman yang tepat. Komputer secara praktis didesain dengan jenis aritmetika yang terbatas. Gambar 4.1 menunjukkan jenis-jenis aritmetika yang populer digunakan dalam komputer.



Gambar 4.1 Jenis-jenis aritmetika komputer

Karena sirkuit-sirkuit internal komputer merupakan dunia biner, maka komputer digital pasti melakukannya dengan aritmetika biner. Walaupun pemakai menggunakan sistem bilangan desimal maka dapat dipastikan sistem desimal ini diubah ke dalam bilangan biner oleh komputer sebelum melakukan operasi-operasi aritmetika. Hasil akhir diubah kembali ke dalam desimal untuk diberikan kepada pemakai. Dalam aplikasi bisnis khusus, jumlah data sangat besar sehingga dalam kasus ini banyak waktu yang terbuang dalam konversi jika hanya digunakan aritmetika biner. Karena itu untuk aplikasi tersebut, komputer didesain dengan aritmetika desimal. Bilangan desimal direpresentasikan dalam bentuk BCD (*Binary Coded Decimal*). Algoritma aritmetika BCD digunakan dalam aritmetika desimal. Aritmetika desimal mempunyai dua kelemahan:

1. Hardware untuk aritmetika desimal kompleks dan mahal.
2. Sistem bilangan BCD tidak efisien dalam menggunakan kombinasi bit biner—ada beberapa yang tidak terpakai. Karena itu penggunaan hardware penyimpanan (memori dan register) juga menjadi besar, yang pada akhirnya meningkatkan biaya hardware.

Aritmetika biner titik-tetap (*fixed-point*) terdapat pada hampir semua komputer. Walaupun format untuk titik-mengambang (*floating-point*) mencakup rentang bilangan yang luas dibandingkan dengan bilangan titik-tetap. Namun hardware aritmetika titik-mengambang kompleks dan mahal. Karena itu, banyak komputer lama menangani data titik-mengambang dengan melakukan konversi ke data titik-tetap. Saat ini mikroprosesor telah mempunyai unit titik-mengambang secara *built-in*, berkat fabrikasi VLSI yang harganya murah.

4.1.1 Komponen-Komponen Hardware

Komponen-komponen yang banyak digunakan dalam aritmetika biner antara lain *gate*, *latch*, *flip-flop*, *register*, *shift register*, *counter*, *multiplexer*, *demultiplexer*, *decoder*, *encoder*, dan sebagainya.

GATE. *Gate* adalah sirkuit logika yang melakukan suatu fungsi pada masukan-masukan dan menghasilkan sebuah keluaran. Jenis-jenis *gate* berdasarkan fungsi logikanya adalah NOT, OR, AND, Exclusive OR, dan sebagainya.

FLIP-FLOP. *Flip-flop* adalah sebuah media penyimpanan biner satu bit (memori satu bit) yang dapat menyimpan 0 atau 1. Keluaran flip-flop dapat berubah jika diberikan pemicuan sinyal clock. Jenis-jenis flip-flop misalnya adalah flip-flop RS, flip-flop D, flip-flop JK dan flip-flop T.

LATCH. *Latch* merupakan flip-flop logika yang mengunci status sinyal; sinyal gate mengontrol pembukaan atau penutupan latch. Latch digunakan untuk mengambil suatu kondisi sebelum kondisi tersebut hilang guna keperluan yang akan datang.

REGISTER. *Register* adalah sekumpulan flip-flop dengan masukan clock bersama. Register digunakan untuk penyimpanan atau menahan sementara word di dalam pemrosesan (operasi aritmetika).

SHIFT REGISTER. *Shift register* adalah jenis register yang berfungsi untuk melakukan pergeseran bit-bit di dalam register. Pergeseran dilakukan ke kiri atau ke kanan per bit setiap masukan sinyal kontrol.

COUNTER. *Counter* berfungsi dalam pencacahan masukan-masukan pulsa clock, pencacahan dapat dilakukan secara menaik atau menurun. Counter digunakan dalam operasi aritmetika dan operasi-operasi kontrol.

MULTIPLEXER. Multiplexer berfungsi untuk memilih salah satu dari beberapa input ke bagian output sesuai sinyal kontrol selector. Multiplexer disebut juga data selector.

DEMULTIPLEXER. Demultiplexer berfungsi mengirim input ke salah satu dari beberapa output sesuai sinyal kontrol selector. Demultiplexer disebut juga data distributor. Demultiplexer berguna untuk konversi serial ke paralel.

DECODER. Decoder berfungsi sebagai identifikasi atau penetapan pola pada n input dengan mengaktifkan salah satu dari 2^n output sesuai angka input. Decoder digunakan sebagai pendekodean instruksi, alamat dan perintah (command).

ENCODER. Encoder merupakan kebalikan dari decoder. Encoder memiliki sejumlah saluran input dan hanya salah satu yang diaktifkan pada satu waktu dan menghasilkan kode output sebanyak n -bit bergantung pada input yang aktif.

4.2 REPRESENTASI DATA

Program-program aplikasi yang berbeda menggunakan tipe data yang berbeda, bergantung pada masalah. Program bahasa mesin dapat bekerja dengan data numerik atau data non-numerik. Data numerik dapat berupa bilangan desimal atau bilangan biner. Data non-numerik adalah data dengan tipe sebagai berikut:

1. Data karakter
2. Data alamat
3. Data logika

Semua data non-biner direpresentasikan dalam komputer dalam bentuk kode biner.

4.2.1 Data Karakter

Kumpulan bit digunakan untuk merepresentasikan sebuah karakter yang mungkin berupa sebuah digit, sebuah alfabet atau simbol khusus dan sebagainya. String dari multikarakter umumnya membentuk sebuah data yang dapat diartikan, contoh:

- C#
- IBM System/360
- Pentium 4

Dahulu, beberapa tipe kode telah digunakan untuk merepresentasikan berbagai macam karakter. Kode MORSE yang populer telah digunakan dalam telegrafi dan teleprinter. Kode ini mempunyai 5-bit kode di mana setiap karakter direpresentasikan dengan 5-bit pola. The American Standard Institute (ANSI) mengumumkan kode ASCII (*American Standard Code for Information Interchange*). Standar ini menggunakan sebuah pola 8-bit di mana 7 bit dispesifikasikan sebagai karakter. Bit ke-8 umumnya digunakan sebagai ‘parity bit’ untuk deteksi error. Tetapi, beberapa komputer tidak menggunakan bit yang ke-8. Beberapa komputer mengikuti standar ASCII. Kode 8 bit lain yang juga populer adalah kode EBCDIC (*Extended Binary Coded Decimal Interchange Code*) yang digunakan oleh beberapa sistem komputer seperti IBM System/360.

4.2.2 Data Alamat

Pada beberapa instruksi, operand adalah alamat-alamat. Suatu operasi (aritmetika atau logika) mungkin terlibat atau tidak. Bergantung pada instruksi, alamat diperlakukan sebagai bilangan biner atau bilangan logika. Pada beberapa instruksi, alamat operand ditetapkan dalam beberapa bagian-bagian seperti yang dibahas pada mode pengalamatan.

4.3 DATA BINER

Data biner dapat direpresentasikan sebagai bilangan titik-tetap atau sebagai bilangan titik-mengambang. Pada representasi bilangan titik-tetap, posisi dari titik biner adalah sudah pasti tetap dalam satu tempat. Pada representasi bilangan titik-mengambang posisi titik biner dapat berada di mana saja. Bilangan titik-tetap dikenal sebagai bilangan integer/bulat, sedangkan bilangan titik-mengambang dikenal sebagai bilangan real/pecahan. Operasi aritmetika bilangan titik-tetap sederhana dan memerlukan sedikit sirkuit hardware. Aritmetika bilangan titik-mengambang sulit dan memerlukan sirkuit hardware yang kompleks. Bilangan titik-mengambang mempunyai dua keuntungan dibandingkan bilangan titik-tetap:

1. Dengan jumlah bit data yang diberikan, nilai maksimum atau minimum yang dapat direpresentasikan dalam representasi bilangan titik-mengambang lebih tinggi daripada representasi bilangan titik-tetap. Bilangan ini berguna untuk keperluan bilangan yang sangat besar atau bilangan yang sangat kecil

2. Representasi bilangan titik-mengambang lebih akurat dalam operasi aritmetika.

4.3.1 Bilangan Titik-Tetap

Pada bilangan titik-tetap, posisi titik biner adalah tetap, tetapi tidak secara eksplisit ditunjukkan dalam komputer. Pada sebagian besar komputer, titik biner dianggap berada pada sebelah kiri dari MSB. Misalnya, representasi biner 1011 sebenarnya mempunyai arti 0.1011. Sama halnya 0100 menandakan 0.0100. Pada beberapa komputer, titik biner dianggap mengikuti LSB. Pada komputer ini, 1011 berarti 1011.0 dan 0100 berarti 0100.0. Karena itu, dalam satu komputer bilangan titik-tetap 1010 dapat direpresentasikan 0.1010, dan 1010.0 dalam komputer lain.

Jenis-Jenis Representasi Bilangan Titik-Tetap

Bilangan titik-tetap dapat berupa bilangan bertanda atau bilangan tak-bertanda. Jika sebuah bilangan titik-tetap yang diberikan tanpa tanda positif atau negatif, disebut sebagai bilangan tak-bertanda. Jika digunakan bilangan bertanda, umumnya satu bit menunjukkan tanda bilangan. Bilangan positif, bit tandanya adalah 0 dan untuk negatif bit tandanya adalah 1. Ada tiga macam representasi untuk bilangan titik-tetap:

1. Representasi *signed magnitude*
2. Representasi komplementen-1
3. Representasi komplementen-2

Representasi Signed Magnitude

Representasi *signed magnitude* biasa juga disebut representasi benar karena pola bit yang digunakan tidak ada perubahan sepanjang bit tandanya. Contoh ilustrasi berikut adalah bentuk *signed magnitude* (posisi bit tanda ditunjukkan oleh anak panah).

<u>Nilai desimal</u>	<u>Representasi <i>signed magnitude</i></u>
+3	0 1 1
-3	1 1 1
+1	0 0 1
-1	1 0 1
	↑
	Bit tanda

Jika kita mempunyai 1 bit tanda dan 2 bit untuk besaran (*magnitude*), maka rentang bilangan yang dapat direpresentasikan oleh sistem 3 bit ini dalam bentuk *signed magnitude* adalah dari -3 sampai +3 seperti ditunjukkan berikut:

Tanda	Besaran	Nilai desimal
0	00	+0
0	01	+1
0	10	+2
0	11	+3
1	00	-0
1	01	-1
1	10	-2
1	11	-3

Jika n bit digunakan untuk merepresentasikan besaran, rentang bilangan yang dapat direpresentasikan oleh representasi *signed magnitude* adalah $-(2^{n-1}-1)$ hingga $(2^{n-1}-1)$. Jika panjang word 8 bit ($n=8$ -bit), maka rentang yang dapat direpresentasikan adalah -127 hingga +127 karena 1 bit ditempati oleh bit tanda dan 7 bit sisanya disediakan untuk besaran. Pada bentuk *signed magnitude*, bilangan negatif maksimum adalah 11111111 yang merupakan nilai desimal dari -127. Bilangan positif maksimum 01111111 adalah nilai desimal +127. Walaupun representasi *signed magnitude* merupakan sistem yang sederhana, namun tidak cocok digunakan untuk operasi-operasi aritmetika.

Representasi Komplementen-1

Pada representasi ini, posisi besaran ditunjukkan dalam bentuk komplementen-1. Komplementen-1 sebuah bilangan diperoleh dengan cara melakukan pem-balikan setiap bit, semua bit 1 diubah menjadi 0 dan semua bit 0 diubah menjadi 1. Contoh berikut mengilustrasikan pembentukan komplementen-1:

<u>Besaran</u>	<u>Komplementen-1</u>
0111	1000
1001	0111
1001	0110

Pada representasi komplemen-1, sebuah bilangan positif direpresentasikan tanpa ada perubahan, mirip dengan bentuk signed magnitude. Tetapi berbeda dalam representasi bilangan negatif. Bit tanda adalah 1 dan bagian *magnitude* (besaran) diletakkan sebagai komplemen-1. Contoh berikut mengilustrasikan hal ini:

<u>Bilangan desimal</u>	<u>Representasi komplemen-1</u>
+7	00111
-7	11000

↑
Bit tanda

Rentang nilai bilangan komplemen-1 sama dengan rentang signed magnitude. Jika n bit digunakan untuk merepresentasikan besaran, rentang bilangan yang dapat direpresentasikan oleh representasi komplemen-1 adalah $-(2^{n-1}-1)$ hingga $+(2^{n-1}-1)$.

Representasi Komplemen-2

Komplemen-2 suatu bilangan diperoleh dengan menambahkan 1 pada komplemen-1 bilangan tersebut. Contoh berikut mengilustrasikan pembentukan komplemen-2:

<u>Besaran</u>	<u>Komplemen-1</u>	<u>Komplemen-2</u>
0111	1000	1001
1000	0111	1000
1001	0110	0111

Pada sistem bilangan komplemen-2, bilangan positif direpresentasikan dalam bentuk orsinil, serupa dengan bentuk signed magnitude. Pada kasus bilangan negatif, bit tanda adalah 1 dan besaran diletakkan sebagai komplemen-2. Hal ini dapat dijelaskan dengan contoh berikut:

<u>Bilangan desimal</u>	<u>Representasi Komplemen-2</u>
+7	00111
-7	11001

↑
Bit tanda

Rentang nilai bilangan yang dapat direpresentasikan oleh representasi komplemen-2 adalah dari $-(2^{n-1})$ hingga $+(2^{n-1}-1)$. Misalnya jika digunakan bilangan 8 bit maka rentang bilangannya adalah dari -128 sampai +127. Pada bentuk komplemen-2, bilangan negatif maksimum adalah 10000000 yang merupakan nilai desimal dari -128. Bilangan positif maksimum 01111111 adalah nilai desimal +127.

4.3.2 Bilangan Titik-Mengambang

Representasi bilangan titik-mengambang mempunyai tiga bagian:

1. Mantissa
2. Basis
3. Eksponen

Contoh berikut mengilustrasikan sistem bilangan titik-mengambang:

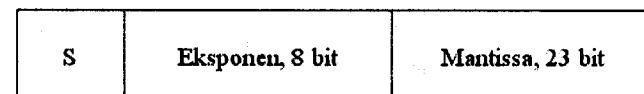
<u>Bilangan</u>	<u>Mantissa</u>	<u>Basis</u>	<u>Eksponen</u>
3×10^6	3	10	6
110×2^8	110	2	8
6132.784	0.6132784	10	4
34.58	0.3458	10	2

Mantissa dan eksponen direpresentasikan secara eksplisit dalam komputer. Tetapi basisnya adalah yang digunakan oleh komputer tersebut. Umumnya komputer mengikuti basis 2. Pada umumnya sebuah bilangan f direpresentasikan sebagai $f = m \times r^e$ di mana m adalah mantissa, r adalah basis dari sistem bilangan dan e adalah eksponen (pangkat dari basis yang digunakan). Gambar 4.2 menunjukkan format umum bilangan titik-mengambang.

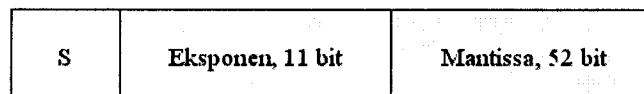
S	Eksponen	Mantissa

Gambar 4.2 Format bilangan titik-mengambang

Semula penggunaan format berbeda-beda antar pabrik komputer untuk merepresentasikan bilangan titik-mengambang. Tetapi saat ini telah digunakan format standar ANSI/IEEE secara luas. Ada dua format standar yang dikeluarkan yaitu untuk presisi tunggal (Gambar 4.3) dan format standar untuk bilangan presisi ganda (Gambar 4.4).



Gambar 4.3 Format bilangan titik-mengambang presisi tunggal (32 bit)



Gambar 4.4 Format bilangan titik-mengambang presisi ganda (64 bit)

4.3.3 Normalisasi

Bilangan titik-mengambang yang diberikan dapat direpresentasikan dengan banyak cara seperti yang ditunjukkan untuk bilangan desimal 110×2^8 :

$$110 \times 2^8, \quad 11 \times 2^9, \quad 1100 \times 2^7, \quad 1.1 \times 2^{10}, \quad 0.11 \times 2^{11}, \quad .011 \times 2^{12} \\ \dots \text{dst.}$$

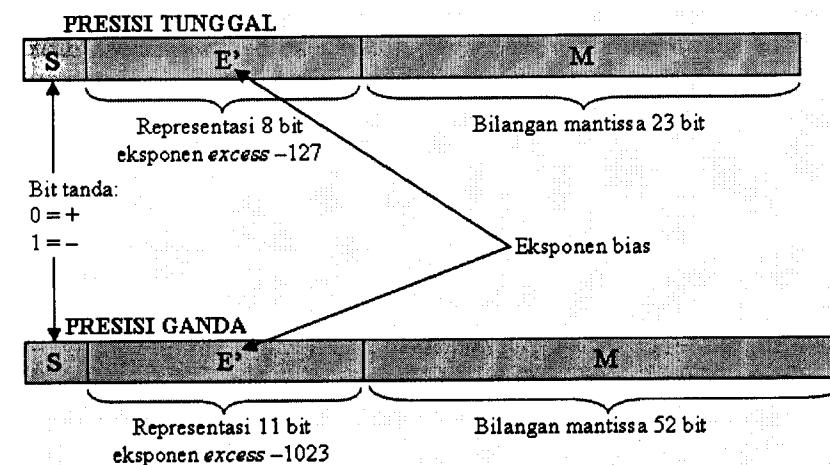
Suatu bilangan titik-mengambang berada dalam bentuk ternormalisasi jika msb dari mantissa adalah non-zero. Untuk mengubah bilangan yang tidak ternormalisasi menjadi bilangan yang ternormalisasi, mantissa harus digeser ke kanan atau ke kiri dengan tepat, menaikkan atau menurunkan eksponen. Jika semua bilangan titik-mengambang direpresentasikan dalam komputer dengan bentuk ternormalisasi, maka posisi bit satu dapat disimpan dengan mengabaikan msb (yang selalu 1), ini disebut *hidden 1 principle*.

4.3.4 Konversi Format Scientific ke Format Standar IEEE 754

Untuk melakukan konversi bilangan titik-mengambang dari format scientific menjadi format standar IEEE 754, dilakukan langkah-langkah sebagai berikut:

1. Ubah menjadi bilangan biner.
2. Normalisasikan bilangan tersebut sehingga terdapat satu digit nonzero di ujung terkiri, lakukan pengaturan eksponen sesuai kebutuhan.

3. Simpan digit biner mantissa ke sisi kanan (lihat format pada Gambar 4.5).
4. Tambahkan 127 ke bagian eksponen ($E' = E + 127$) dan ubah hasil penjumlahan tersebut ke biner unsigned untuk nilai eksponen yang akan disimpan. Untuk presisi-ganda, tambahkan 1023 ke eksponen ($E' = E + 1023$).
5. Bit tanda = 1 untuk bilangan negatif dan bit tanda = 0 untuk bilangan positif.



Gambar 4.5 Format titik-mengambang presisi tunggal dan presisi ganda standar IEEE 754

Contoh 4.1

Konversikan bilangan biner format saintifik $+0.0010110\dots \times 2^9$ menjadi format presisi tunggal standar IEEE 754

Solusi:

Pertama dilakukan normalisasi $+0.0010110\dots \times 2^9$ menjadi $+1.0110\dots \times 2^6$

Diperoleh S = 0; M = 0110... dan E = 6

Sehingga E' dapat dihitung: $E' = E + 127 = 6 + 127 = 133 = 10000101$
Format Standar IEEE 754:

0	10000101	01100000000000000000000000000000
---	----------	----------------------------------

4.4 ARITMETIKA BILANGAN TITIK-TETAP

Operasi-operasi aritmetika biner titik-tetap merupakan aspek penting dalam ALU. Empat jenis operasi aritmetika yang dibutuhkan adalah penjumlahan, pengurangan, perkalian, dan pembagian. Ada dua strategi dalam implementasi mesin aritmetika ini:

1. Perancangan sirkuit ALU untuk empat operasi di atas. Pendekatan ini mengakibatkan harga ALU menjadi mahal dan kompleks.
2. Perancangan sirkuit ALU hanya untuk penjumlahan dan pengurangan, sedangkan operasi perkalian dan pembagian dilakukan dengan menggunakan software. Pendekatan ini mengakibatkan ALU menjadi lambat dan murah.

4.4.1 Perluasan Bit Tanda

Anggap suatu bilangan n bit harus disimpan dalam suatu register atau lokasi memori dengan panjang word yang agak panjang ($\geq n + 1$), maka aturan perluasan bit tanda harus diletakkan pada bit kosong yang tersisa (pada msb). Bit tanda disalin/diletakkan ke posisi paling kiri bit kosong. Jika bilangan positif ($S=0$), semua msb yang kosong diisi dengan 0. Jika bilangan negatif ($S=1$), semua msb yang kosong diisi 1. Konsep ini disebut perluasan bit tanda (*sign extension*).

Contoh 4.2

Tunjukkan bagaimana 5 bit bilangan bertanda berikut disimpan dalam register 8-bit.

- (a) 00111 (b) 11001 (c) 00000 (d) 11000 (e) 01010 (f) 11010

Solusi:

- (a) 00111 disimpan menjadi 00000111
- (b) 11001 disimpan menjadi 11111001
- (c) 00000 disimpan menjadi 00000000
- (d) 11000 disimpan menjadi 11111000
- (e) 01010 disimpan menjadi 00001010
- (f) 11010 disimpan menjadi 11111010

4.4.2 Penjumlahan Integer

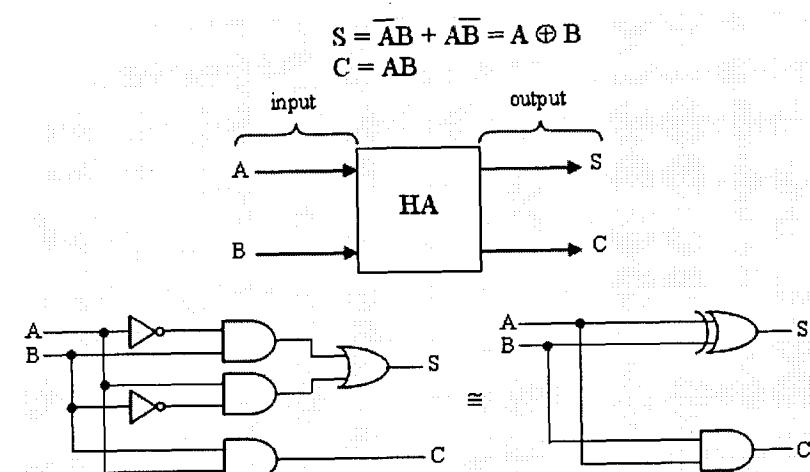
Dua jenis penjumlahan yang dibutuhkan di dalam ALU:

1. **Penjumlahan bilangan tak-bertanda:** kalkulasi alamat operand adalah satu contoh di mana penjumlahan seperti itu diperlukan.
2. **Penjumlahan bilangan bertanda:** kalkulasi numerik dalam komputer umumnya mengikuti penjumlahan jenis ini karena operand-operand-nya berupa bilangan positif dan negatif.

Untuk membangun sirkuit aritmetika komputer maka dibutuhkan sirkuit *adder* (penjumlah) dasar, yaitu *half adder* dan *full adder*.

Half Adder

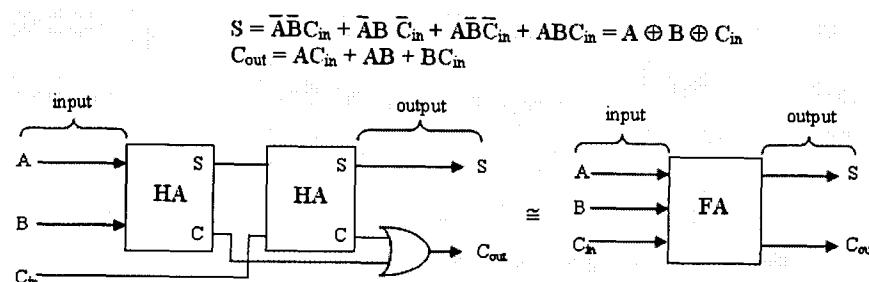
Half adder adalah sebuah penjumlah yang melakukan penjumlahan dua bilangan biner 1 bit. Pada Gambar 4.6 ditunjukkan diagram blok untuk half adder. Sum (S) dan Carry (C) dinyatakan dalam ekspresi:



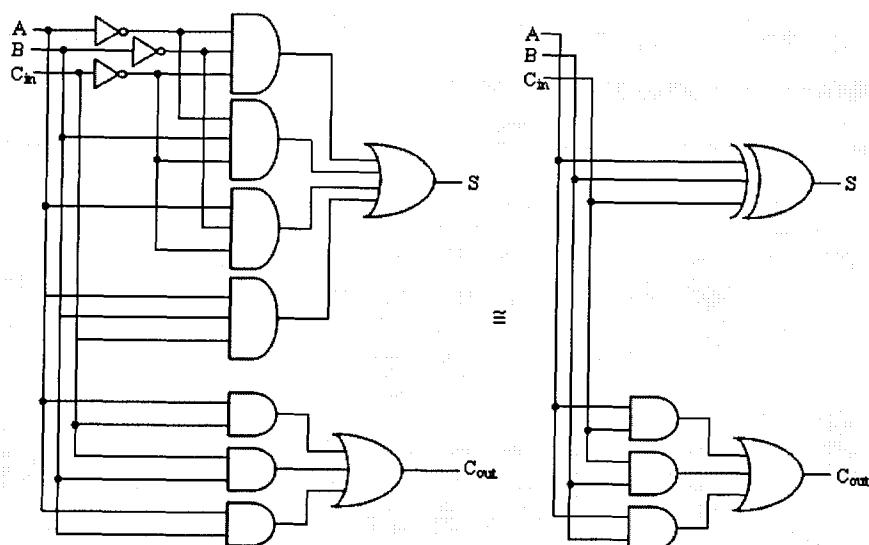
Gambar 4.6 Diagram blok dan sirkuit logika half adder

Full Adder

Full adder adalah penjumlah yang melakukan penjumlahan tiga bilangan biner 1 bit, yaitu dua sebagai masukan A dan B, serta masukan ketiga berasal dari *carry in* (C_{in}). Pada Gambar 4.7 ditunjukkan diagram blok full adder. **Sum (S)** dan **Carry out (C_{out})** dinyatakan dalam ekspresi:



Gambar 4.7(a) Diagram blok full adder

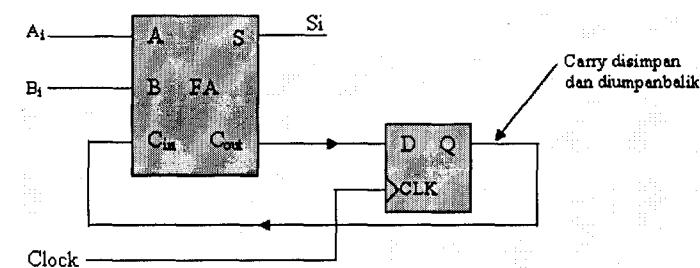


Gambar 4.7(b) Sirkuit logika full adder

Serial Adder

Serial adder hanya mempunyai sebuah penjumlah bit tunggal. Serial adder digunakan untuk penjumlahan dua bilangan secara sekuensial bit demi bit yang dimulai dari LSB. Pada Gambar 4.8 ditunjukkan sebuah serial adder dari dua bilangan n-bit. Operand-operand (A dan B) disuplai bit demi bit yang dimulai dari LSB. Penjumlahan posisi satu bit menggunakan satu siklus clóck. Jadi, untuk sebuah n-bit serial adder, dibutuhkan n-siklus clock untuk menyelesaikan proses penjumlahan. Pada setiap siklus, carry dihasilkan oleh sebuah posisi bit yang akan diingat dalam sebuah flip-flop dan diberikan sebagai masukan selama siklus berikutnya sebagai carry.

Kelebihan sirkuit serial adder adalah lebih kecil dan karena itu sangat murah, kekurangannya lambat karena membutuhkan sebanyak n-clock untuk menyelesaikan penjumlahan bilangan n-bit.



Gambar 4.8 Diagram blok serial adder

Parallel Adder

Parallel adder adalah sebuah penjumlah yang mempunyai beberapa full adder terpisah yang dikaskadekan menjadi satu grup. Parallel adder melakukan penjumlahan dari *multi fulladder* secara simultan. Mekanisme penjumlahan internal berbeda pada setiap jenis parallel adder. Pada dasarnya ada dua jenis parallel adder:

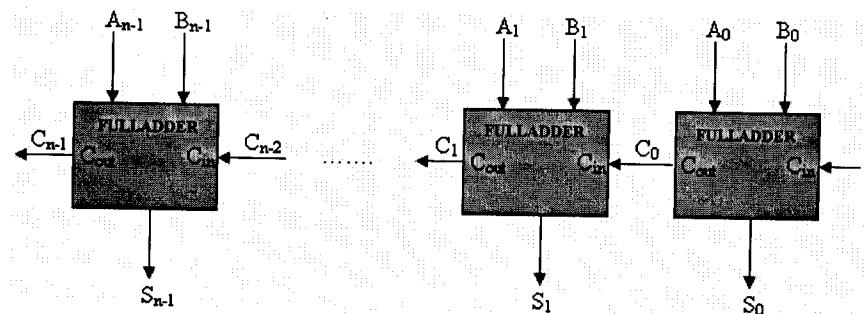
1. *ripple carry adder*
2. *carry look-ahead adder*.

Perbedaan keduanya adalah pada cara carry bit dibangkitkan.

Ripple Carry Adder.

Pada Gambar 4.9 ditunjukkan *ripple carry adder* dengan n -bit. Prosesnya adalah sebagai berikut: ada n -bit full adder yang dihubungkan dengan cara kaskade. Setiap *carry out* dari full adder menjadi *carry in* bagi full adder bit berikutnya. Setiap tingkat melakukan penjumlahan untuk salah satu bit dari dua bilangan dan carry dari posisi bit sebelumnya. Setiap tingkat adder menghasilkan satu bit hasil-jumlah dan satu bit carry.

Setiap carry out pada tingkat adder diberikan pada carry in tingkat adder berikutnya dan merambatkan seterusnya hingga mencapai keluaran tingkat adder terakhir. Keluaran adder (*sum* dan *carry*) valid hanya pada saat selesainya waktu tunda dari sejumlah suplai masukan. Carry in pada tingkat LSB dijaga tetap LOW (0) karena bit ini tidak relevan. Bit ini mempunyai kegunaan khusus, yaitu jika hasil-jumlah diinginkan ditambah satu ($A+B+1$) maka C_{in} ini dibuat 1.



Gambar 4.9 Ripple carry adder dengan n -bit

Waktu tunda dimunculkan oleh carry yang menentukan lamanya penjumlahan. Carry dari posisi LSB harus merambat melalui semua posisi bit. Jika t_d adalah waktu tunda untuk setiap tingkat full adder dan ada sejumlah n -bit, maka waktu tunda rambatan (*propagation delay*) maksimum untuk n -bit *ripple carry adder* adalah $n \times t_d$.

Keuntungan: *ripple carry adder* lebih cepat daripada *serial adder*; kekurangannya: *ripple carry adder* menjadi lambat ketika jumlah bit bertambah. Carry yang terjadi pada LSB harus dirambatkan ke seluruh tingkatan secara berurut. Jika panjang dari operand besar, maka waktu yang

digunakan untuk merambatkan carry dari tingkat LSB ke MSB menjadi besar. Hal ini dikenal dengan istilah *ripple carry problem*.

Contoh 4.3

Dua buah bilangan tak bertanda dari masing-masing dua-bit akan dijumlahkan. Yang mana yang lebih cepat? Apakah *serial adder* atau *ripple carry adder*?

Solusi:

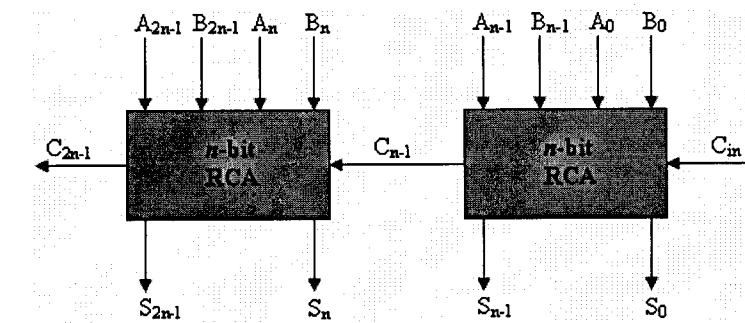
Pada *serial adder*, waktu tunda rambatan dari flip-flop t_{FF} juga berkontribusi secara total terhadap delay. Jika t_d adalah delay yang disebabkan oleh sebuah adder, maka periode minimum dari clock akan menjadi $= t_d + t_{FF}$. Karena itu waktu minimum yang dibutuhkan oleh *serial adder* adalah dua kali lipat sehingga menjadi $2 \times (t_d + t_{FF})$.

Pada *ripple carry adder*, waktu penjumlahan yang dibutuhkan $= 2 \times t_d$.

Jadi *ripple carry adder* yang lebih cepat.

Kaskade Ripple Carry Adder.

Dengan melakukan kaskade pada *ripple carry adder*, kita dapat membuat adder dengan panjang yang dikehendaki. Pada Gambar 4.10 ditunjukkan konstruksi adder $2n$ -bit menggunakan dua adder n -bit dari jenis *ripple carry adder*. Kedua modul melakukan penjumlahan secara simultan dan kemudian mengizinkan perambatan carry dari satu modul ke modul berikutnya.



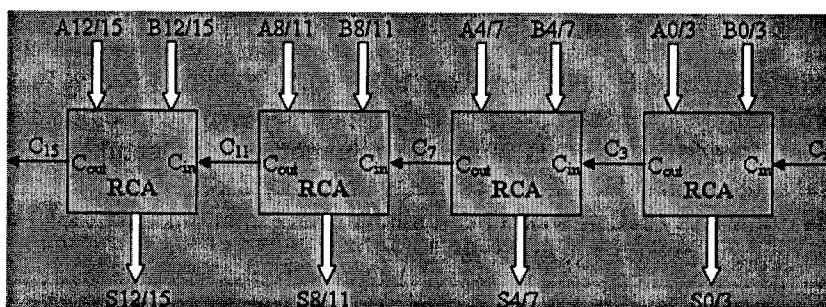
Gambar 4.10 Kaskade ripple carry adder (RCA) $2 \times n$ -bit

Contoh 4.4

Buat konstruksi parallel adder 16 bit menggunakan ripple carry adder 4 bit.

Solusi:

Kita membutuhkan empat ripple carry adder 4 bit untuk membentuk adder 16 bit. Gambar 4.11 menunjukkan interkoneksi di antara empat adder tersebut.



Gambar 4.11 Adder 16-bit menggunakan RCA 4 x 4-bit

Carry Look-Ahead Adder

Carry Look-Ahead Adder adalah penjumlahan kecepatan tinggi yang mengikuti suatu strategi khusus untuk pembangkitan carry yang cepat pada setiap tingkat tanpa menunggu carry dari tingkat sebelumnya. Dia menggunakan determinasi/penentuan dini carry in ke suatu tingkat tanpa menggunakan carry out dari tingkat sebelumnya.

Anggap sebuah adder n -bit, maka carry out dari dua posisi least significant diberikan oleh:

$$C_0 = A_0 B_0; \quad C_1 = A_1 C_0 + A_1 B_1 + B_1 C_0 = A_1 B_1 + C_0(A_1 + B_1)$$

C_0 adalah carry out dari LSB yaitu adder tingkat 0

C_1 adalah carry out dari tingkat 1.

Jika adder dengan i tingkat, maka carry out dari tingkat ini dinyatakan dengan:

$$C_i = A_i B_i + (A_i + B_i) C_{i-1}$$

C_i dan C_{i-1} adalah carry out dari masing-masing tingkat i dan tingkat $i-1$.

Dengan melakukan analisis pada ekspresi ini dengan cermat, kita temukan bahwa ada dua faktor yang memutuskan apakah akan ada *carry bit* atau tidak. Faktor pertama bergantung pada bit-bit data dari tingkat yang sekarang dan faktor kedua bergantung pada pola bit data dari tingkat yang sekarang dan *carry* dari tingkat sebelumnya. Faktor pertama dapat dikatakan sebagai komponen *carry generation* dan faktor kedua dapat dikatakan sebagai *carry propagation*. Jadi sekarang kita dapat menuliskan kembali ekspresi *carry* itu dengan:

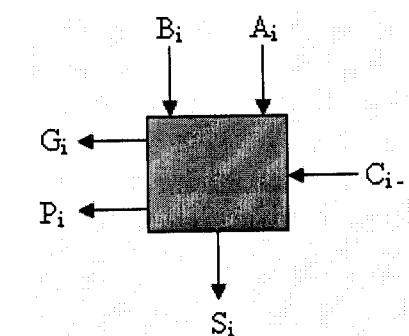
$$C_i = G_i + P_i C_{i-1}$$

di mana G_i disebut "*generate*" dan P_i disebut "*propagate*"

Kesimpulan:

1. Jika A_i dan B_i keduanya '1' ($A_i B_i = 1$) maka tingkat i membangkitkan carry dengan mengabaikan C_{i-1}
2. Jika A_i atau B_i '1' ($A_i + B_i = 1$) maka tingkat i merambatkan/meneruskan C_{i-1}

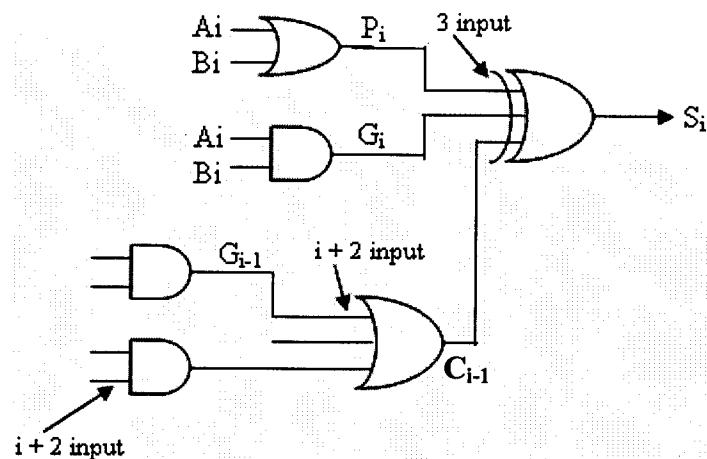
Pada Gambar 4.12 ditunjukkan masukan dan keluaran setiap tingkat.



Gambar 4.12(a) Carry look-ahead adder satu tingkat

Ekspresi untuk carry C_i dapat diperluas sebagai berikut:

$$\begin{aligned} C_i &= G_i + P_i C_{i-1} \\ &= G_i + P_i (G_{i-1} + P_{i-1} C_{i-2}) \end{aligned}$$

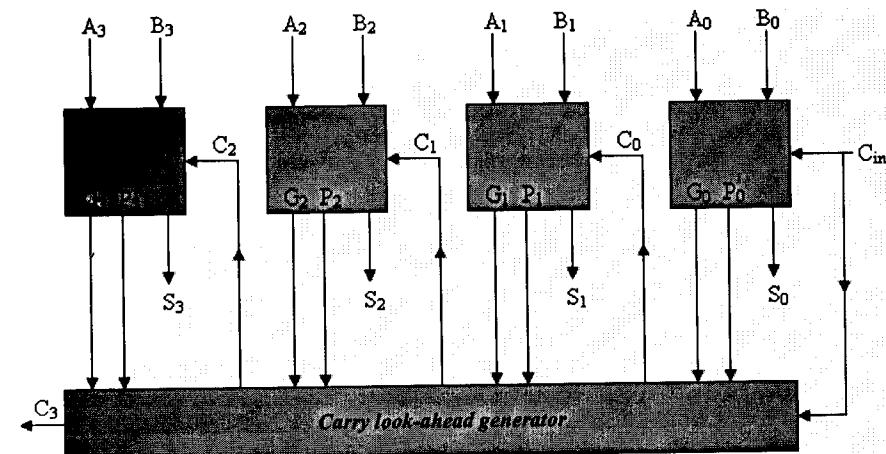


Gambar 4.12(b) Carry look-ahead adder satu tingkat

Jadi C_i adalah jumlah hasil kali keluaran P dan G dari tingkatan sebelumnya. Perluasan pernyataan ini adalah untuk semua carry, jelas bahwa semua carry dalam suatu parallel adder dapat diturunkan langsung dari pola bit data masukan tanpa menunggu status carry dari posisi bit sebelumnya. Hal ini berarti setiap tingkat dapat melakukan penjumlahan tanpa bergantung pada tingkat lainnya. Dengan demikian kecepatan penjumlahan meningkat, tetapi beberapa gate dengan banyak masukan dibutuhkan, seperti panjang adder (n) meningkat.

Pada Gambar 4.13 ditunjukkan digram blok carry look-ahead adder 4-bit. Carry yang muncul diturunkan sebagai berikut:

$$\begin{aligned} C_0 &= G_0 + P_0 C_{in}; \\ C_1 &= G_1 + P_1 G_0 + P_1 P_0 C_{in} \\ C_2 &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in} \\ C_3 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in} \end{aligned}$$



Gambar 4.13 Carry look-ahead adder 4-bit

C_{in} adalah carry in ke tingkat LSB yang biasanya 0. Kalkulasi waktu tunda rambatan dari adder 4-bit mudah dilakukan. Tunda rambatan pada adder 4-bit adalah $4 \times d$ di mana d adalah delay gerbang rata-rata.

Contoh 4.5

Hitung waktu tunda untuk *carry look-ahead adder* 8 bit. Anggap waktu tunda gerbang rata-rata adalah 5 ns.

Solusi:

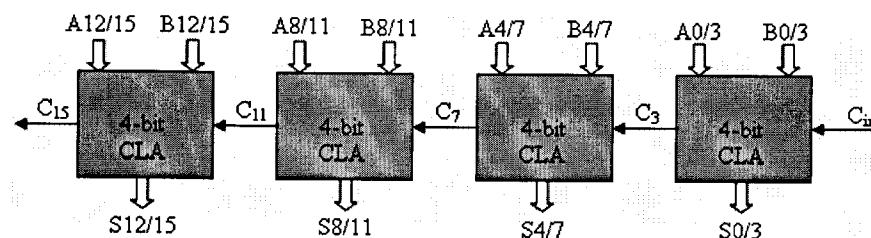
Waktu tunda *carry look-ahead adder* = $4 \times 5 \text{ ns} = 20 \text{ ns}$

Membangun Penjumlahan Besar

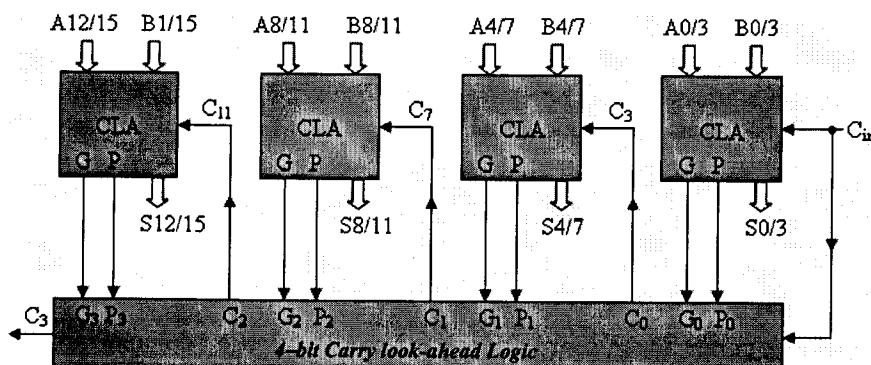
Untuk perancangan suatu adder yang panjang untuk penjumlahan dua bilangan 32-bit, carry out dari tingkat terakhir yaitu C_{31} dibangkitkan oleh sebuah gerbang (gate) OR 33-masukan. Salah satu masukannya ($P_{31}P_{30} \dots P_0C_{in}$) ke gerbang OR ini sendiri dibangkitkan dari sebuah gerbang AND 33-masukan. Gerbang yang besar demikian tidak praktis untuk dibuat karena jarang diperlukan. Karena itu, untuk membangun sebuah adder dengan *word length* yang panjang, sebuah desain multilevel adalah dengan mengkaskadekan modul adder 4-bit atau 8-bit. Sekali lagi, kita mempunyai

dua pendekatan dalam membangun adder 16-bit seperti yang diilustrasikan berikut:

1. Gunakan empat buah modul carry look-ahead adder 4-bit dan kaskadekan dengan cara ripple carry seperti yang ditunjukkan pada Gambar 4.14.
2. Gunakan empat buah modul carry look-ahead adder dan kaskadekan dengan cara carry look-ahead seperti yang ditunjukkan pada Gambar 4.15.



Gambar 4.14 Kaskade 4×4 -bit CLA dengan cara ripple carry



Gambar 4.15 Kaskade 4×4 -bit CLA dengan cara carry look-ahead adder

4.4.3 Penjumlahan Komplemen-2

Penjumlahan bilangan bertanda dapat dilakukan dengan menggunakan representasi bilangan komplemen-1 tetapi penjumlahan bilangan komplemen-2 lebih umum digunakan dalam komputer sebab cocok untuk

implementasi langsung tanpa ada komplikasi. Sebelum membahas cara penjumlahan dengan metode komplemen-2, mari kita membahas beberapa contoh. Pada Gambar 4.16 diberikan ilustrasi penjumlahan dua buah bilangan ($A + B$) dalam enam set yang berbeda. Kita menganggap bahwa kita mempunyai sebuah ALU 5-bit termasuk bit tandanya. Setiap representasi komplemen-2, bilangan positif direpresentasikan dengan bit 0 sebagai bit tanda tanpa ada perubahan (biner yang sebenarnya). Bilangan negatif mempunyai tanda 1 dan besarnya (*magnitude*) dikonversi ke dalam bentuk komplemen-2. Semua bit termasuk bit tanda merupakan bagian yang mengalami proses di dalam penjumlahan. Pada kasus Gambar 4.16(a), bit tandanya berbeda dan ada carry out dari posisi MSB dan dari posisi bit tandanya. Hasilnya benar karena $(+7) + (-3) = (+4)$ dan bilangan ini merupakan bilangan positif. Pada kasus Gambar 4.16(b), bit tandanya berbeda. Tidak ada carry dari posisi MSB atau dari posisi bit tanda. Kasus ini sederhana dari $(-6) + (+5) = (-1)$. Hasilnya negatif dan dalam bentuk komplemen-2. Pada kasus Gambar 4.16(c) kedua bilangan tersebut adalah positif. Ada carry dari posisi MSB tetapi tidak ada dari posisi bit tanda. Bit tanda dari hasil adalah 1 yang menunjukkan nilai negatif yang tidak benar. Kasus ini adalah sebuah contoh klasik *overflow* sebab $(+9) + (+8) = +17$, sedangkan bilangan positif maksimum yang dapat direpresentasikan oleh bilangan 4-bit adalah 15. Karena itu kasus ini hasilnya tidak benar. Pada kasus Gambar 4.16(d), dua bilangan negatif dijumlahkan. Ada carry dari posisi MSB dan posisi bit tanda. Hasilnya negatif dan sah (*valid*) tetapi dalam bentuk komplemen-2, $(-9) + (-3) = (-12)$. Pada kasus Gambar 4.16(e), kedua bilangan adalah negatif tetapi hasilnya adalah positif. Tidak ada carry dari posisi MSB tetapi ada carry dari posisi bit tanda. Kasus ini adalah suatu contoh *overflow* karena $(-9) + (-8) = -17$, dan bilangan negatif maksimum yang dapat direpresentasikan oleh bilangan 4-bit adalah -16. Pada kasus gambar 4.16(f) adalah penjumlahan dua bilangan yang mempunyai magnitude yang sama tetapi berbeda dalam bit tanda. Ada carry dari posisi MSB dan bit tanda. Hasilnya adalah bilangan positif 0 $[(-9) + (+9) = (+0)]$.

$$\begin{array}{r}
 +7 \quad 00111 \\
 -3 \quad 11101 \\
 \hline
 +4 \quad 00100
 \end{array}
 \quad
 \begin{array}{r}
 -6 \quad 11010 \\
 +5 \quad 00101 \\
 \hline
 -1 \quad 11111
 \end{array}
 \quad
 \begin{array}{r}
 +9 \quad 01001 \\
 +8 \quad 01000 \\
 \hline
 10001
 \end{array}$$

(a) C C
 (b)
 (c) NCC

$$\begin{array}{r}
 -9 \quad 10111 \\
 -3 \quad 11101 \\
 \hline
 10100
 \end{array}
 \quad
 \begin{array}{r}
 -9 \quad 10111 \\
 -8 \quad 11000 \\
 \hline
 01111
 \end{array}
 \quad
 \begin{array}{r}
 -9 \quad 10111 \\
 +9 \quad 01001 \\
 \hline
 00000
 \end{array}$$

(d) CO
 (e) CNC
 (f) CC

Gambar 4.16 Contoh kasus penjumlahan komplement-2

Kesimpulan pada penjumlahan komplement-2:

1. Penjumlahan dua bilangan dengan tanda berbeda tidak menimbulkan overflow.
2. Penjumlahan dua bilangan dengan tanda sama dapat terjadi overflow. Jika tanda dari hasil berbeda dengan tanda kedua bilangan yang dijumlahkan, maka ini menunjukkan terjadi overflow. Jika tanda dari hasil sama dengan tanda kedua bilangan yang dijumlahkan, maka tidak terjadi overflow. Overflow ditunjukkan jika status carry dari posisi MSB dan posisi bit tanda berbeda.

4.4.4 Pengurangan Integer

Pengurangan bilangan titik-tetap ($A - B$) dilakukan dengan menjumlahkan bilangan A dan komplement-2 bilangan B. Karena itu, langkah pertama dari pengurangan adalah melakukan konversi operand kedua (B) menjadi bentuk komplement-2. Langkah selanjutnya adalah menjumlahkan kedua bilangan tersebut. Adder dapat dengan mudah melakukan modifikasi pengurangan komplement-2. Pada operasi $(A) - (B)$, kita dapat menggantinya dengan $(A) + (B_k2)$ di mana B_k2 adalah komplement-2 dari B. Komplement-2 dari B dapat diperoleh dengan menjumlahkan 1 dengan komplement-1 dari B. Cara yang mudah untuk melakukan konversi bilangan biner menjadi komplement-1 adalah dengan melakukan penjumlahan Exclusive-OR bilangan itu dengan deretan angka 1 semua (dengan kata lain membalik/

Inversi semua bit). Karena itu, prosedur untuk pengurangan ($A - B$) adalah sebagai berikut:

1. Lakukan Exclusive-OR bilangan B dengan 1111...1
2. Jumlahkan hasil tersebut (hasil komplement-1) dengan bilangan A dengan 1 ke posisi carry in (LSB). Hasil ini adalah ekivalen dengan penjumlahan A dan B_{k2} .

$$\begin{array}{r}
 +7 \quad 00111 \\
 (-) \quad 11101 \rightarrow K2 \\
 \hline
 +3 \quad 00011
 \end{array}
 \quad
 \begin{array}{r}
 +7 \quad 00111 \\
 (+) \quad 01010 \quad (+10) \\
 \hline
 01010
 \end{array}$$

(a)

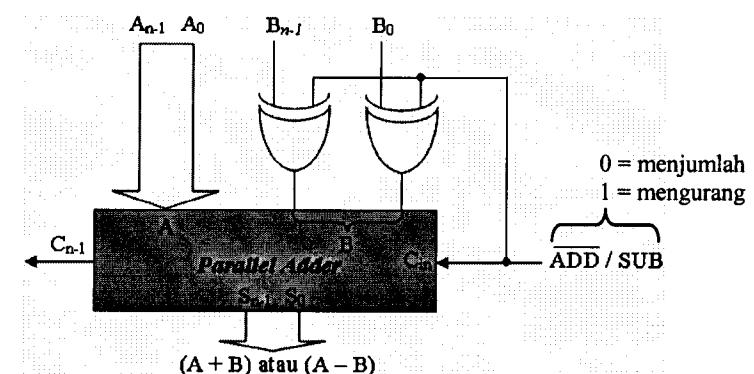
$$\begin{array}{r}
 -6 \quad 11010 \\
 (-) \quad 11011 \rightarrow K2 \\
 \hline
 -5 \quad 00101
 \end{array}
 \quad
 \begin{array}{r}
 -6 \quad 11010 \\
 (+) \quad 11111 \quad (-1) \\
 \hline
 11111
 \end{array}$$

(b)

$$\begin{array}{r}
 -6 \quad 11010 \\
 (-) \quad 00101 \rightarrow K2 \\
 \hline
 -5 \quad 11011
 \end{array}
 \quad
 \begin{array}{r}
 -6 \quad 11010 \\
 (+) \quad 10101 \quad (-11) \\
 \hline
 10101
 \end{array}$$

(c)

Gambar 4.17 Contoh pengurangan komplement-2



Gambar 4.18 Sirkuit penjumlah/pengurang komplement-2

4.4.5 Perkalian Bilangan Tidak Bertanda

Secara teoretis, perkalian dapat dilakukan dengan menjumlah secara berulang. Misalkan perkalian $A \times B$ di mana A adalah *multiplier* (pengali) dan B adalah *multiplicand* (terkali). Jika kita menjumlah B sendiri dengan A kali, hasil jumlahnya akan sama dengan hasil perkalian antara A dan B. Hal ini tidak dilakukan oleh komputer karena prosesnya sangat lambat. Metode yang sederhana dan mungkin lebih praktis adalah mengalikan B dengan bit demi bit pada A dan menjumlahkan semua hasil kali parsial tersebut. Metode ini sama dengan metode perkalian manual menggunakan kertas dan pensil seperti yang ditunjukkan pada Gambar 4.19 untuk 1010 (10 desimal) dengan 1101 (13 desimal). Hasilnya adalah 10000010 (130 desimal).

$$\begin{array}{r}
 \begin{array}{ccccccccc}
 & & B_3 & B_2 & B_1 & B_0 & & \\
 & & A_3 & A_2 & A_1 & A_0 & \times & \\
 \hline
 & A_0B_3 & A_0B_2 & A_0B_1 & A_0B_0 & & & \\
 A_1B_3 & A_1B_2 & A_1B_1 & A_1B_0 & & & \\
 A_2B_3 & A_2B_2 & A_2B_1 & A_2B_0 & & & \\
 A_3B_3 & A_3B_2 & A_3B_1 & A_3B_0 & & & \\
 \hline
 Z_7 & Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0 & +
 \end{array} \\
 \left. \begin{array}{c}
 1010 \\
 1101 \\
 \hline
 1010 \\
 0000 \\
 1010 \\
 \hline
 1010 \\
 + \\
 \hline
 10000010
 \end{array} \right\}$$

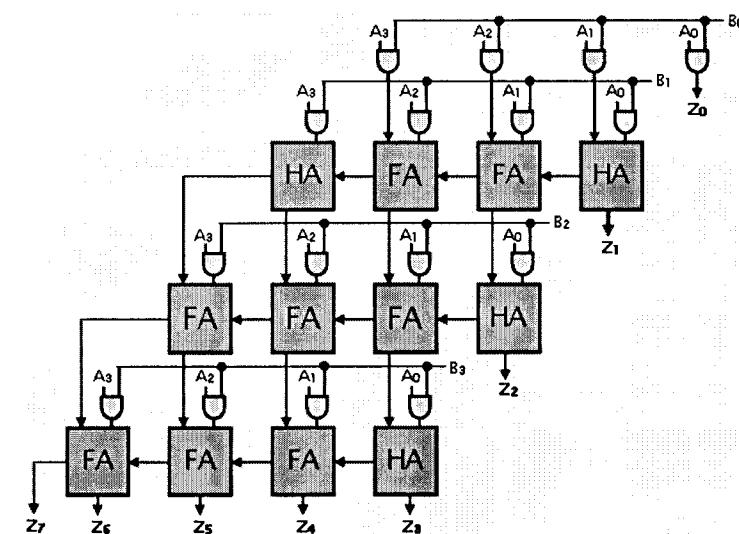
Gambar 4.19 Metode perkalian manual

Metode perkalian manual mengikuti langkah-langkah berikut:

1. Mulai dari LSB multiplier. Jika bit ini 1, maka hasil-kali parsial sama dengan multiplicand. Jika bit LSB 0, maka hasil kali parsial sama dengan 0.
2. Lakukan analisis bit multiplier berikutnya dan bangkitkan hasil-kali parsial seperti pada langkah 1. Hasil-kali parsial yang baru ini ditulis bergeser ke kiri 1 bit terhadap hasil-kali parsial sebelumnya. Ulangi langkah ini hingga semua bit pada multiplier dikerjakan (mendapat giliran).
3. Jumlahkan semua hasil-kali parsial untuk memperoleh hasil-kali sesungguhnya.

Ketika ingin diimplementasikan pada komputer maka prosedur sebenarnya dapat dimodifikasi sedikit supaya perkalian dilakukan secara efisien

oleh hardware dengan menggunakan penjumlah biner paralel. Hasil-kali parsial dijumlahkan segera setiap satu siklus selesai mengantikan penjumlahan di akhir proses supaya kita tidak menyimpan hasil-kali parsial. Untuk merealisasikan mesin perkalian tersebut dibutuhkan 16 buah gerbang AND, 4 buah half adder dan 8 buah full adder seperti yang ditunjukkan pada Gambar 4.20.



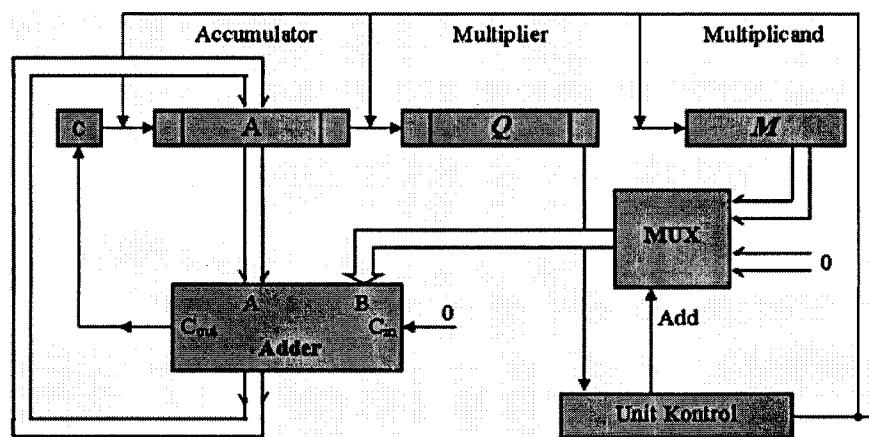
Gambar 4.20 Mesin perkalian menggunakan penjumlah paralel

Mesin perkalian pada Gambar 4.20 tentunya membutuhkan sirkuit yang kompleks sehingga harganya mahal. Waktu tunda keseluruhan array ini diberikan oleh nilai terbesar dari:

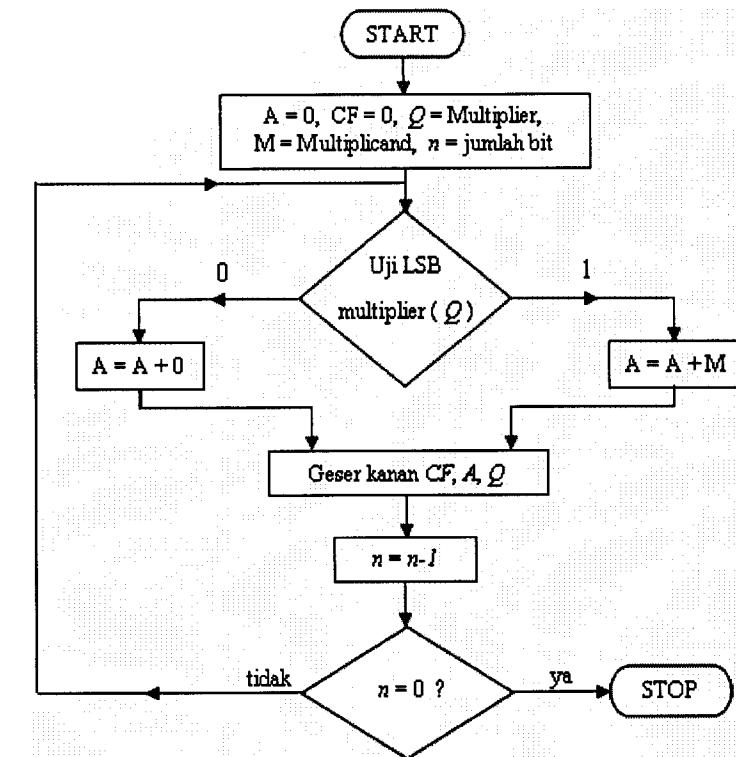
$$t_{AND} + 6t_c; t_{AND} + t_{AF} + 5t_c; t_{AND} + 2t_{AF} + 4t_c; \text{ atau } t_{AND} + 3t_{AF}$$

di mana t_{AND} adalah tunda propagasi dari gerbang AND dua masukan, t_c adalah tunda carry dari sebuah full adder dan t_{AF} adalah tunda keluaran hasil jumlah dari sebuah full adder, dengan asumsi bahwa setiap gerbang AND mempunyai tunda waktu yang identik demikian halnya untuk setiap full adder. Sayangnya sirkuit ini membutuhkan hardware yang lebih banyak jika jumlah bit dalam multiplier dan multiplicand meningkat, juga dibutuhkan sebuah register untuk menyimpan hasil kali yang panjangnya dua kali dari panjang word multiplier/multiplicand.

Metode perkalian seperti itu sebenarnya dapat juga ditempuh dengan menggunakan algoritma yang memberikan hasil yang sama namun dengan hardware yang minim. Dari sana kita bisa melakukan bahwa hasil-kali parsial digeser ke kanan 1 bit setiap satu siklus agar multiplicand dapat dijumlahkan langsung ke siklus berikutnya. Pada Gambar 4.21 ditunjukkan diagram blok operasi perkalian dan pada Gambar 4.22 ditunjukkan algoritmanya. Pertama, akumulator dibersihkan (*clear*) dan multiplier ditempatkan pada register Q dan multiplicand pada register M. Carry out dari adder disimpan pada *Carry Flag* (CF). CF juga ikut digeser ke kanan bersama akumulator dan register multiplier.



Gambar 4.21 Diagram blok mesin perkalian bilangan tak bertanda



Gambar 4.22 Algoritma perkalian bilangan tak bertanda

Contoh 4.6

Lakukan perkalian bilangan tak bertanda: (a) 13×11 (b) 8×6

Solusi:

	CF	A	Q	M
(a) Mulai	0	0000	1101	1011
	0	1011	1101	1011
	0	0101	1110	1011
	0	0010	1111	1011
	0	1101	1111	1011
	0	0110	1111	1011
	1	0001	1111	1011
	0	1000	1111	1011
(b) Mulai	0	0000	1000	0110
	0	0000	0100	0110
	0	0000	0010	0110
	0	0000	0001	0110
	0	0110	0001	0110
	0	0011	0000	0110

4.4.6 Perkalian Bilangan Bertanda

Secara komparatif, perkalian bilangan dalam bentuk *signed magnitude* (besaran bertanda) lebih mudah. Bilangan positif dapat dikalikan dengan cara yang sama seperti bilangan tak-bertanda (*unsigned number*). Jika tanda multiplier dan multiplicand berbeda, maka tanda hasil-kalinya adalah negatif. Jika kedua bilangan tersebut negatif atau keduanya positif, maka tanda hasil-kalinya adalah positif. Jadi aturan tanda bilangan berlaku umum, yaitu: tanda hasil-kali adalah hasil ***Exclusive-OR*** dari tanda bilangan-bilangan tersebut.

Perkalian Komplemen-2

Jika salah satu operand negatif merupakan bentuk komplemen-2, maka metode perkalian sebelumnya tidak dapat digunakan. Bilangan harus diubah menjadi bentuk besar bertanda sebelum perkalian dan hasil-kalinya harus diubah kembali menjadi bentuk komplemen-2. Walaupun dapat dikerjakan dengan baik, tetapi menambah waktu proses perkalian. Ada beberapa metode perkalian menggunakan komplemen-2. Algoritma Booth adalah suatu metode perkalian komplemen-2 yang populer. Metode ini juga meningkatkan kecepatan proses perkalian dengan menganalisis bit-bit multiplier pada satu waktu.

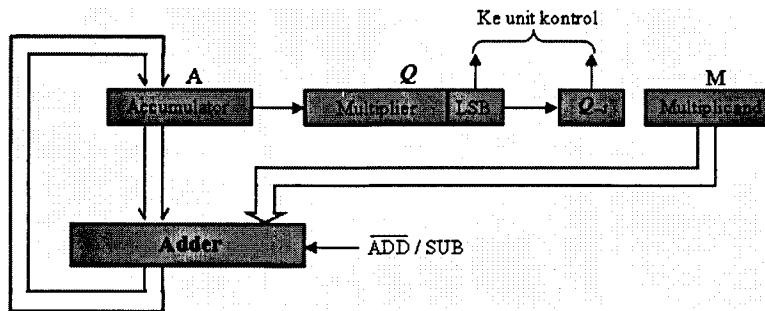
Algoritma Booth

Algoritma Booth menangani operand-operand komplemen-2 positif dan negatif tanpa memerlukan koreksi dalam prosedur atau pada hasilnya. Bila multiplier mempunyai kumpulan angka 1, banyaknya penjumlahan yang diperlukan akan berkurang. Jumlah penurunan waktu bergantung pada pola bit multiplier. Prinsip dasar yang dianut dalam algoritma Booth sebenarnya tidak ada perkalian, yang ada adalah penjumlahan dari pola-pola multiplicand yang digeser secara tepat.

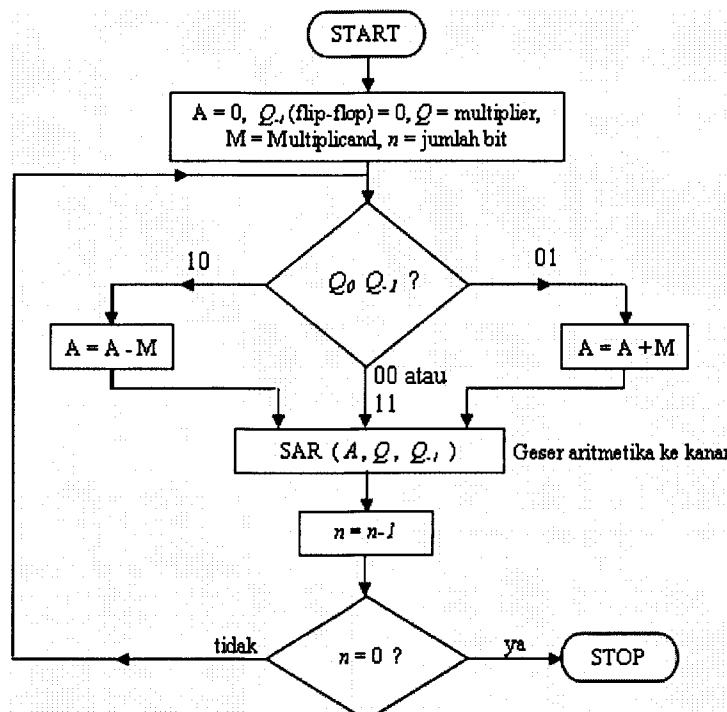
Gambar 4.23 menunjukkan diagram blok dari konfigurasi algoritma Booth. Pada Gambar 4.24 diberikan flowchart implementasi algoritma Booth. Sebuah flip-flop Q_{-1} (register 1 bit) digunakan untuk menyimpan bit LSB multiplier (Q_0) jika terjadi pergeseran ke kanan. Prinsipnya pertama akumulator dan flip-flop dibuat *clear*, serta register multiplier dan multiplicand diisikan (*load*). Langkah-langkah operasinya sebagai berikut:

1. Jika bit Q_0 dan bit Q_{-1} sama (00 atau 11), tidak dilakukan penjumlahan. Isi akumulator (hasil-kali parsial) dan multiplier digeser aritmetika ke kanan (SAR) satu bit. (Bit tanda akumulator akan sama dengan bit sebelum pergeseran. Bit LSB akumulator masuk ke posisi MSB multiplier).
2. Jika bit Q_0 dan bit Q_{-1} adalah 01, tambahkan multiplicand ke akumulator, kemudian akumulator dan multiplier digeser aritmetika ke kanan satu bit.
3. Jika bit Q_0 dan bit Q_{-1} adalah 10, kurangi akumulator dengan multiplicand, kemudian akumulator dan multiplier digeser aritmetika ke kanan satu bit.

4. Lakukan langkah nomor 1 sampai semua bit-bit multiplier telah mendapat giliran.
5. Hasil perkalian berada dalam register akumulator dan multiplier.



Gambar 4.23 Diagram blok algoritma Booth



Gambar 4.24 Diagram alir algoritma Booth

Contoh 4.7

Catatan: Gunakan algoritma Booth untuk menyelesaikan perkalian

$$(a) (+3) \times (+7) \quad (b) (-3) \times (+7)$$

Solusi:

Tabel perkalian menggunakan algoritma Booth untuk dua kasus:

(a) Perkalian $(+3) \times (+7)$

	A	Q	Q_{-1}	M	
Nilai awal	0000	0011	0	0111	$A \leftarrow A \cdot M$
	1001	0011	0	0111	SAR
	1100	1001	1	0111	SAR
	1110	0100	1	0111	$A \leftarrow A + M$
	0101	0100	1	0111	SAR
	0010	1010	0	0111	SAR
Nilai akhir	0001	0101	0	0111	SAR

(b) Perkalian $(-3) \times (+7)$

	A	Q	Q_{-1}	M	
Nilai awal	0000	1101	0	0111	$A \leftarrow A \cdot M$
	1001	1101	0	0111	SAR
	1100	1110	1	0111	$A \leftarrow A + M$
	0011	1110	1	0111	SAR
	0001	1111	0	0111	$A \leftarrow A - M$
	1010	1111	0	0111	SAR
	1101	0111	1	0111	$A \leftarrow A - M$
Nilai akhir	1110	1011	1	0111	SAR

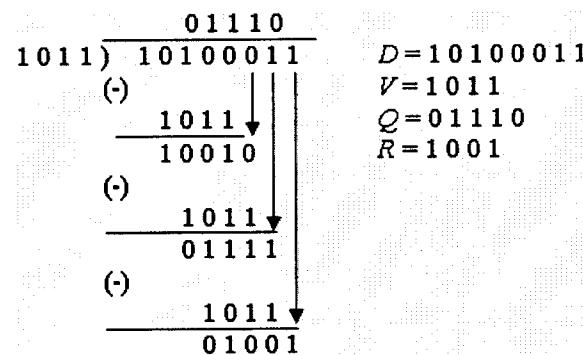
4.4.7 Pembagian Integer

Pembagian merupakan proses aritmetika yang lebih kompleks daripada perkalian. Anggap D adalah *dividend* (terbagi) dan V adalah *divisor* (pembagi), maka pembagian D oleh V akan mengikuti persamaan $D = (Q \times V) + R$, di mana Q adalah hasil-bagi dan R adalah sisa pembagian.

Metode mudah yang tersedia untuk pembagian adalah dengan menggunakan bilangan tidak bertanda (D dan V). Walaupun ada juga beberapa teknik pembagian dengan bilangan bertanda, pembagian dapat dilakukan mirip dengan bilangan tak bertanda dan kemudian pada akhirnya sama dengan bilangan bertanda. Jika operand-operand dalam bentuk komplemen-2, metode yang biasa diikuti adalah pertama mengubahnya menjadi bilangan tidak bertanda sebelum pembagian dan kemudian mengubah kembali hasil yang diperoleh menjadi bentuk komplemen-2.

Pembagian Integer Tidak Bertanda

Pertama mariyah kita memikirkan metode manual menggunakan kertas dan pensil untuk contoh yang ditunjukkan pada Gambar 4.25 di mana $D = 10100011$ (desimal 163) dan $V = 1011$ (desimal 11). Metode ini biasa dikenal dengan metode *longhand*. Langkah-langkahnya seperti berikut:

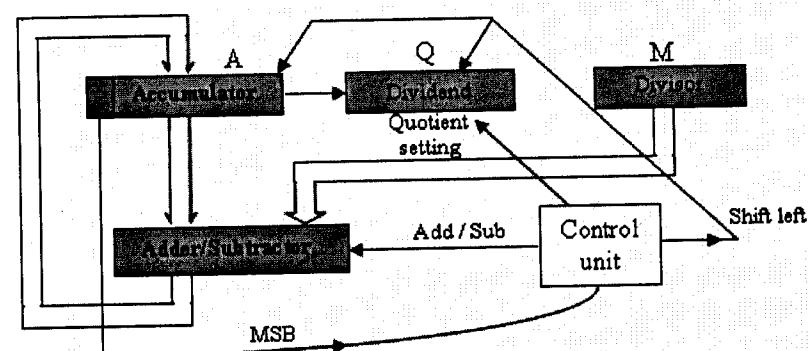


Gambar 4.25 Pembagian metode manual

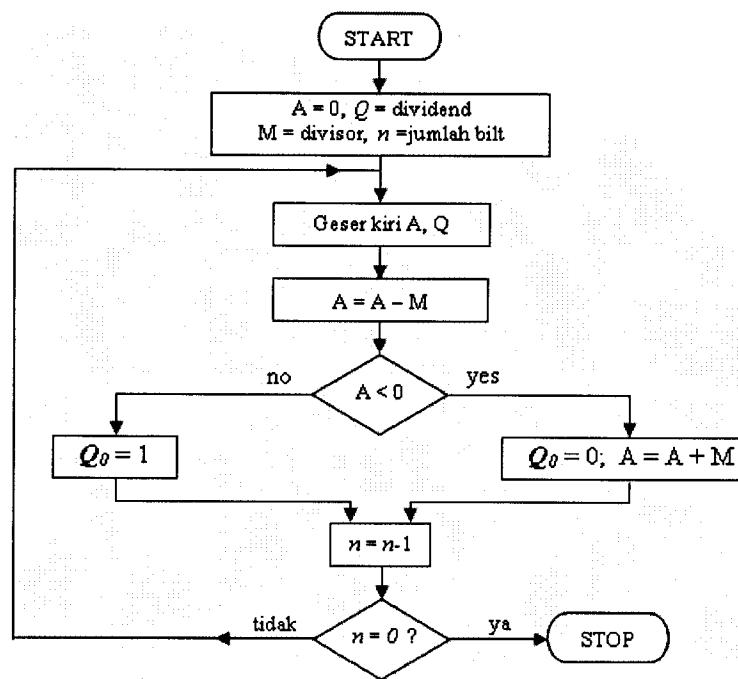
1. Sejajarkan divisor di bawah dividend sehingga bit-bit dividend (dari MSB) yang ada membentuk bilangan biner lebih besar atau sama dengan divisor.
2. Perkurangkan divisor dari dividend untuk membentuk sisa pembagian parsial. Letakkan 1 sebagai bit hasil-bagi.
3. Masukkan satu bit lagi (bit berikutnya pada sisi kanan) dari dividend ke bagian kanan sisa pembagian parsial.
4. Perkurangkan divisor dari sisa pembagian yang baru.

- a. Jika sisa pembagian adalah nol atau lebih besar nol (bilangan positif), maka letakkan 1 sebagai hasil bagi berikutnya;
- b. Jika sisa-pembagian adalah negatif (yaitu lebih kecil dari nol), maka letakkan 0 sebagai bit hasil-bagi berikutnya; simpan sisapembagian parsial yang lama dengan jalan menambahkan kembali divisor.
5. Sejajarkan kembali divisor untuk pengurangan lainnya setelah memasukkan lagi satu bit dari dividend ke sebelah kanan dari sisapembagian parsial.
6. Ulangi langkah 4 dan 5, hingga LSB dividend telah dimasukkan dalam langkah 3, dan langkah 4 juga telah selesai. Sekarang kita peroleh hasil $Q = 01110$ (desimal 14); $R = 1001$ (desimal 9).

Restoring Division



Gambar 4.26 Diagram blok datapath mesin restoring division



Gambar 4.27 Algoritma restoring division

Metode manual mengalami sedikit modifikasi dalam metode restoring division untuk penyesuaian implementasi datapath. Gambar 4.26 menunjukkan susunan datapath dan Gambar 4.27 menampilkan algoritma dalam bentuk diagram alir. Langkah-langkah proses sebagai berikut:

1. Masukkan dividend n bit dalam Q dan divisor n bit dalam M .
2. Clear A . Isi dari A dan Q membentuk item tunggal $2n$ bit.
3. Geser A, Q ke kiri 1 bit.
4. Perkurangkan M dari A , dan ganti A dengan hasil.
5. (a) Jika bit tanda A adalah 0, set $Q_0 = 1$ (pembagian berhasil)

(b) Jika bit tanda A adalah 1, reset $Q_0 = 0$; tambah M kembali ke hasil sekarang dalam A untuk mendapatkan kembali nilai A yang sebelumnya (operasi tidak berhasil; simpan ulang (*restoring*) A)
6. Ulangi langkah 3 dan 5 untuk n kali untuk bit Q yang tersisa (1, 2, ..., n)
7. Q berisi hasil-bagi dan A berisi sisa-pembagian.

Contoh 4.8

Gunakan metode restoring division untuk menyelesaikan pembagian $7 : 4$

Solusi:

Step	A	Q	M
Initial	0000	0111	0100
1	0000	1110	0100
2	1100	1110	0100
3	0000	1110	0100
4	0001	1100	0100
5	1101	1100	0100
6	0001	1100	0100
7	0011	1000	0100
8	1111	1000	0100
9	0011	1000	0100
10	0111	0000	0100
11	0011	0000	0100
Final	0011	0001	0100

Non-restoring Division

Metode *non-restoring division* adalah metode yang lebih cepat dan efisien. Pada *restoring division*, melibatkan pengurangan dan pembagian jika hasil bagi adalah 0. Pada *non-restoring division*, penjumlahan atau pengurangan dilakukan pada semua bit hasil-bagi. Jadi pada *non-restoring division*, melibatkan total n operasi (penjumlahan/pengurangan).

Langkah-langkah proses non-restoring division:

1. (a) Jika tanda A adalah 0, geser A dan Q ke kiri satu bit dan kurangi A dengan M .

(b) Jika tanda A adalah 1, geser A dan Q ke kiri satu bit dan tambahkan M ke A .
2. Sekarang, jika tanda A adalah 0, set $Q_0 = 1$; sebaliknya set $Q_0 = 0$.
3. Jika tanda A adalah 1, tambahkan M ke A .
4. Ulangi langkah 1–3 sebanyak n kali.

4.5. ARITMETIKA BILANGAN TITIK-MENGAMBANG

Aritmetika bilangan titik-mengambang mempunyai aturan sendiri dan memerlukan sirkuit hardware tambahan dibandingkan dengan datapath titik-tetap. Aritmetika perkalian dan pembagian titik-mengambang lebih mudah daripada penjumlahan dan pengurangan. Operand-operand pada titik-mengambang disimpan dalam bentuk ternormalisasi. Demikian halnya hasil operasi aritmetika juga disimpan dalam bentuk ternormalisasi.

Anggap dua buah operand titik-mengambang A dan B yang dinyatakan sebagai:

$$A = M_a \times r^{E_a} \text{ dan } B = M_b \times r^{E_b}$$

di mana M_a dan M_b adalah mantissa dari A dan B, r adalah basis (biasanya 2 atau 10), sedangkan E_a dan E_b adalah masing-masing eksponen dari A dan B. Sebelum dilakukan penjumlahan atau pengurangan antara A dan B, maka salah satu operand harus diubah agar A dan B mempunyai nilai eksponen yang sama. Hal tersebut dilakukan dengan cara menggeser titik basis (biner atau desimal) operand tersebut.

4.5.1 Penjumlahan Bilangan Titik-Mengambang

Anggap penjumlahan $A+B$ di mana A dan B adalah bilangan biner titik-mengambang. Langkah-langkah penjumlahan sebagai berikut:

1. Periksa apakah A atau B adalah 0; jika demikian, hasilnya adalah operand lain.
2. Hitung selisih (ed) antara eksponen, yaitu $(E_a - E_b)$ atau $(E_b - E_a)$.
3. Pilih bilangan A atau B yang mempunyai eksponen yang lebih kecil.
4. Geser ke kanan mantissa bilangan demi bilangan yang dipilih sampai sama dengan ed, yaitu selisih antara eksponen. Sekarang kedua bilangan mempunyai eksponen yang sama.
5. Lakukan penjumlahan mantissa baru: $M_a + M_b$.
6. Hasil penjumlahan adalah $(M_a + M_b) \times r^{ed}$
7. Normalisasikan hasil. Hal ini dilakukan dengan menggeser ke kiri bit-bit mantissa hingga msb menjadi non-zero. Setiap pergeseran ke kiri, eksponen dari hasil harus dikurangi dengan 1.

Contoh 4.9

Jumlahkan dua bilangan desimal titik-mengambang $1.02635126 \times 10^{18}$ dan $1.21300213 \times 10^{15}$.

Solusi:

$$A = 1.02635126 \times 10^{18} \text{ dan } B = 1.21300213 \times 10^{15}$$

Langkah-langkahnya sebagai berikut:

1. Baik A dan B bukan 0. Jadi, penjumlahan harus dilakukan
2. $E_a - E_b = 18 - 15 = 3$
3. B mempunyai eksponen yang lebih kecil, yaitu 15
4. Mantissa B harus digeser ke kanan 3 kali
 M_b (nilai yang digeser) = 0.00121300
5. $M_a + M_b =$

$$\begin{array}{r} 1.02635126 \\ 0.00121300 \\ \hline + \\ 1.02756426 \end{array}$$

6. Hasil = $1.02756426 \times 10^{18}$; normalisasikan hasil, $A + B = 10.2756426 \times 10^{17}$

Pada langkah 3 di atas, kita dapat juga memilih bilangan yang mempunyai eksponen yang lebih besar. Pada kasus ini, pada langkah 4 kita harus menggeser ke kiri mantissa bilangan ini dengan tiga digit, jadi mengurangi eksponen menjadi 15. Namun, hal ini akan memberikan hasil yang kurang presisi.

Guard Digit

Ketika mantissa digeser ke kanan, beberapa digit di bagian MSB akan hilang. Untuk mendapatkan hasil dengan akurasi tinggi, maka satu atau lebih bit ekstra yang disebut "guard bit" ditambahkan pada intermediate step. Bit-bit ini disimpan sementara pada bit yang baru saja digeser dari sisi paling kanan mantissa. Ketika bilangan akhirnya harus disimpan dalam suatu register atau dalam memori sebagai suatu hasil, guard bit tidak disimpan. Namun, berdasarkan guard bit, nilai mantissa dapat dibuat presisi dengan teknik pembulatan.

Pembulatan dan Pemotongan

Pembulatan dan pemotongan bilangan titik-mengambang melibatkan pengabaian guard bit, hal ini dapat mengurangi kepresisan, perluasan bergantung pada nilai guard bit. Jika dilakukan pembulatan, 1 ditambahkan ke LSB mantissa jika MSB guard bit adalah 1. Setelah penjumlahan tersebut, mantissa dipotong hingga mencapai n bit. Prosedur ini memberikan hasil yang lebih akurat. Jika MSB dari guard bit adalah 0, maka tidak ada penambahan ke mantissa dan dilakukan pemotongan.

4.5.2 Pengurangan Bilangan Titik-Mengambang

Pengurangan A-B sama dengan penjumlahan kecuali pada langkah 5, dilakukan pengurangan mantissa.

Contoh 4.10

Kurangkan dua bilangan desimal titik-mengambang $1.02635126 \times 10^{18}$ dengan $1.21300213 \times 10^{15}$.

Solusi:

$A - B$, di mana $A = 1.02635126 \times 10^{18}$ dan $B = 1.21300213 \times 10^{15}$

Langkah 1 sampai 4 sama dengan contoh 8,

$$5. Ma - Mb =$$

$$\begin{array}{r} 1.02635126 \\ 0.00121300 \\ \hline + \\ 1.02513826 \end{array}$$

$$6. \text{ Hasilnya } = 1.02513826 \times 10^{18};$$

Normalisasikan hasil, geser ke kiri mantissa satu digit dan turunkan eksponen,

$$\text{Hasil akhir } = 10.2513826 \times 10^{17}$$

4.5.3 Perkalian Bilangan Titik-Mengambang

Perkalian A dan B melibatkan operasi berikut:

$$(M_a \times M_b) \times r^{Ea + Eb}$$

Langkah-langkah yang harus diikuti adalah:

1. Uji apakah A atau B sama dengan nol; jika benar maka hasilnya adalah nol
2. Jumlahkan eksponen yaitu $Ea + Eb$
3. Kurangkan bias 1276 karena keduanya adalah dalam format excess 127
4. Periksa apakah terjadi overflow eksponen atau underflow eksponen. Jika ya, berhenti dan laporan dengan melakukan setup pada flag yang sesuai
5. Kalikan mantissa: $M_a \times M_b$
6. Normalisasikan hasil
7. Bulatkan hasil dengan menggunakan guard digit

Jika suatu eksponen positif yang akan disimpan lebih besar dari nilai eksponen maksimum yang mungkin, maka dihasilkan eksponen yang overflow. Bilangan yang dihasilkan terlalu besar untuk dapat disimpan dalam komputer.

Jika suatu eksponen negatif yang akan disimpan lebih kecil dari nilai yang mungkin, maka dihasilkan eksponen yang underflow. Bilangan yang dihasilkan terlalu kecil untuk disimpan dalam komputer.

4.5.4 Pembagian Bilangan Titik-Mengambang

Perkalian A dan B melibatkan operasi berikut:

$$(M_a / M_b) \times r^{Ea - Eb}$$

Langkah-langkah yang harus diikuti adalah:

1. Uji apakah A adalah 0; jika benar maka hasilnya adalah 0
2. Uji apakah B adalah 0; jika benar maka hasilnya adalah ∞
3. Kurangkan eksponen: $Ea - Eb$
4. Tambahkan bias 127 karena pada langkah 3 kedua bias tersebut dihapus
5. Periksa overflow eksponen; jika ya, maka stop dan laporan dengan men-set flag
6. Periksa underflow eksponen; jika ya, maka stop dan laporan dengan men-set flag
7. Bagi mantissa: M_a / M_b
8. Normalisasikan hasil
9. Pembulatan hasil

RINGKASAN

Program bahasa mesin dapat bekerja dengan data numerik atau data non-numerik. Data numerik dapat berupa bilangan desimal atau bilangan biner. Data non-numerik adalah data dengan tipe sebagai berikut:

1. Data karakter
2. Data alamat
3. Data logika

Semua data non-biner direpresentasikan dalam komputer dalam bentuk kode biner. Data biner dapat direpresentasikan sebagai bilangan titik-tetap (*fixed-point number*) atau sebagai bilangan titik-mengambang (*floating-point number*). Aritmetika bilangan titik-mengambang sulit dan memerlukan perangkat keras yang kompleks dan mahal. Dibandingkan terhadap bilangan titik-tetap, bilangan titik-mengambang mempunyai dua keuntungan:

1. Dengan jumlah bit data yang diberikan, nilai maksimum atau minimum yang dapat direpresentasikan dalam representasi bilangan titik-mengambang lebih tinggi daripada representasi bilangan titik-tetap. Bilangan ini berguna untuk keperluan bilangan yang sangat besar atau bilangan yang sangat kecil
2. Representasi bilangan titik-mengambang lebih akurat dalam operasi aritmetika.

Penjumlahan paralel mempunyai penjumlahan pada setiap bit. *Carry-look ahead adder* lebih cepat daripada *ripple carry adder*. Pengurangan dilakukan dengan cara penjumlahan bilangan komplemen-2 bilangan tersebut. Di dalam komputer penjumlahan dan pengurangan komplemen-2 lebih baik dibandingkan dengan bilangan *signed magnitude*. Algoritma khusus yang digunakan untuk melakukan perkalian dan pembagian integer mengurangi waktu operasi. Aritmetika desimal digunakan hanya jika terdapat data input-output yang besar daripada dengan banyak pemrosesan. Kasus ini untuk aplikasi-aplikasi bisnis tertentu. Umumnya komputer mempunyai hardware hanya untuk aritmetika biner. Aritmetika desimal dilakukan dengan mengonversi data ke dalam bilangan biner dan hasilnya dikonversi kembali ke dalam bilangan desimal.

SOAL-SOAL ULANGAN

1. Secara garis besar aritmetika dalam komputer dibagi menjadi dua bagian besar yaitu (1) _____, dan (2) _____.
2. Aritmetika biner dibagi menjadi dua bagian yaitu (1) _____, dan (2) _____.
3. Aritmetika desimal menggunakan/direpresentasikan dalam bentuk bilangan _____.
4. Mesin aritmetika biner yang membutuhkan hardware yang lebih kompleks dan mahal adalah _____.
5. Aritmetika biner yang menyediakan rentang bilangan yang lebih luas adalah _____.
6. _____ adalah sekumpulan flip-flop dengan masukan clock bersama yang digunakan untuk penyimpanan atau menahan sementara word di dalam pemrosesan (operasi aritmetika).
7. _____ berfungsi untuk memilih salah satu dari beberapa input ke bagian output sesuai sinyal kontrol selektor.
8. _____ berfungsi sebagai identifikasi atau penetapan pola pada n input dengan mengaktifkan salah satu dari 2^n output sesuai angka input.
9. Standar pengkodean karakter yang menggunakan sebuah pola 8-bit di mana 7 bit dispesifikasi sebagai karakter (bit ke-8 umumnya digunakan sebagai *parity bit* untuk deteksi error), disebut kode _____.
10. Kode 8 bit lain yang juga populer adalah kode _____ yang digunakan oleh beberapa sistem komputer seperti IBM System/360.
11. Ada tiga macam representasi untuk bilangan titik-tetap yaitu (1) _____, (2) _____, dan (3) _____.
12. Rentang nilai bilangan yang dapat direpresentasikan dari $-(2^{n-1})$ hingga $+(2^{n-1}-1)$ merupakan rentang bilangan representasi _____.
13. Representasi bilangan titik-mengambang mempunyai tiga bagian yaitu (1) _____, (2) _____, dan (3) _____.
14. Format bilangan titik-mengambang yang dikeluarkan dengan nama Standar IEEE 754 menggunakan dua tingkat kepresisan yaitu (1) _____ dan (2) _____.
15. Untuk membangun sirkuit aritmetika komputer maka dibutuhkan sirkuit adder (penjumlahan) dasar yaitu (1) _____, dan (2) _____.
16. Pada dasarnya ada dua jenis parallel adder yaitu (1) _____, dan (2) _____.

17. Dari kedua jenis parallel adder, yang lebih cepat namun lebih kompleks dan mahal adalah _____.
18. Jika t_d adalah waktu tunda untuk setiap tingkat full adder dan ada sejumlah n -bit, maka waktu tunda rambatan (*propagation delay*) maksimum untuk n -bit *ripple carry adder* adalah _____.
19. _____ adalah penjumlahan kecepatan tinggi yang mengikuti suatu strategi khusus untuk pembangkitan carry yang cepat pada setiap tingkat tanpa menunggu carry dari tingkat sebelumnya.
20. Algoritma Booth merupakan salah satu algoritma untuk perkalian bilangan _____.

SOAL-SOAL LATIHAN

1. Sebuah ALU 8 bit melakukan penjumlahan dua operand yang memiliki nilai 5 dan 9. Tentukan status flag berikut:

a) carry flag	b) Zero flag
c) Overflow flag	d) Sign flag
2. Sebuah lokasi memori berisi 1100001011110010. Tentukan nilai desimal word biner tersebut jika direpresentasikan sebagai:

a) <i>Sign magnitude integer</i>	b) integer tak bertanda
c) integer komplementen-1	d) integer komplementen-2
3. Lakukan operasi aritmetika 8 bit bilangan berikut ini menggunakan representasi komplementen-2:

a) (+47) + (+25)	b) (-47) + (-25)
c) (+47) + (-25)	d) (-47) - (-25)
4. Konversikan bilangan desimal berikut menjadi format titik-mengambang presisi tunggal IEEE 754

a) -58	b) 3.1823
c) 0.6	d) -38.24
5. Konversikan bilangan titik-mengambang presisi tunggal IEEE 754 berikut menjadi ekivalen desimal:

a) 00111111110000000000000000000000
b) 01000010100110000000000000000000
6. Rancang mesin perkalian array (dilengkapi gambar) dengan menggunakan half adder dan full adder untuk dapat menyelesaikan operasi 25×16

7. Tunjukkan langkah-langkah perkalian menggunakan algoritma Booth untuk operasi $(+12) \times (-9)$.
8. Tentukan langkah-langkah pembagian menggunakan *restoring division* untuk operasi $25:7$

BAB 5

DESAIN PROSESOR DAN DATAPATH

Sasaran bab ini:

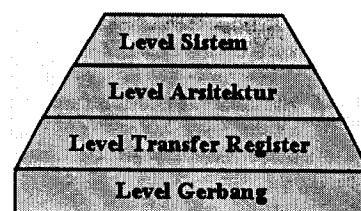
1. Level-Level Desain
2. Fungsi Prosesor
3. Mikro-operasi
4. Bahasa Transfer Register
5. Desain ALU
6. Organisasi Datapath

Desain prosesor merupakan pekerjaan yang paling canggih dalam organisasi suatu komputer. Desain dua sub-unit prosesor yaitu datapath dan unit kontrol merupakan pekerjaan yang saling terkait dan yang satu tidak dapat dikerjakan tanpa pengetahuan yang lainnya. Datapath memainkan peranan penting dalam prosesor. Organisasinya mempunyai pengaruh langsung pada kinerja sistem dan efisiensi pemrograman. Pada bab ini dibahas desain dengan pendekatan prosesor dan difokuskan pada organisasi datapath dan desain unit kontrol.

5.1 FUNGSI PROSESOR

Prosesor dapat dipandang dengan empat level berbeda seperti yang ditunjukkan Gambar 5.1:

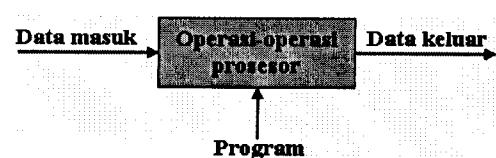
1. Level Sistem
2. Level Set Instruksi atau Level Arsitektur
3. Level Transfer Register
4. Level Gerbang (gate level)



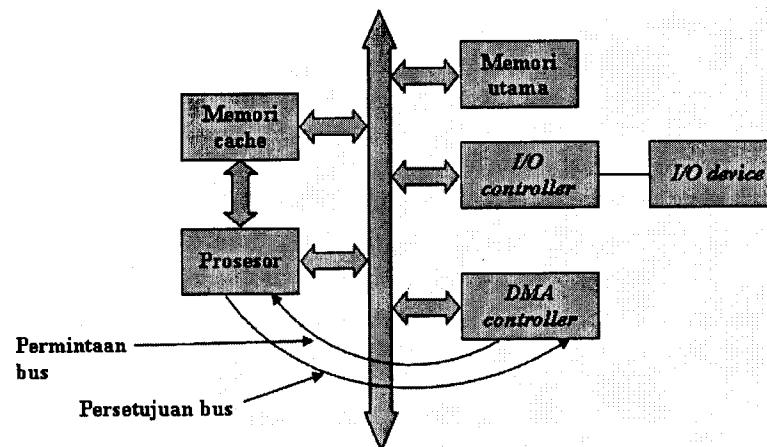
Gambar 5.1 Sudut pandang prosesor

Pada level sistem, prosesor adalah subsistem utama sebuah komputer. Level ini mempunyai dua fungsi:

1. *Eksekusi Program (pelaksanaan siklus instruksi)*; ini melibatkan berbagai operasi-operasi data (Gambar 5.2) seperti pemrosesan data, penyimpanan data, pergerakan data.
2. *Antarmuka dengan subsistem lain* seperti memori utama, memori cache, pengontrol DMA, pengontrol I/O, dan lain-lain. Prosesor bertanggung jawab terhadap koordinasi semua interkoneksi antara subsistem tersebut. Hal ini umumnya dilakukan melalui sebuah struktur bus seperti pada Gambar 5.3. Prosesor adalah bus master yang mempunyai kontrol bus dan dapat memulai transfer bus kapan saja. Pengontrol DMA dapat juga melakukan transfer bus antara memori dan subsistem I/O, tetapi dia harus meminta ke prosesor dan mendapat izin sebelum memulai suatu transfer bus.

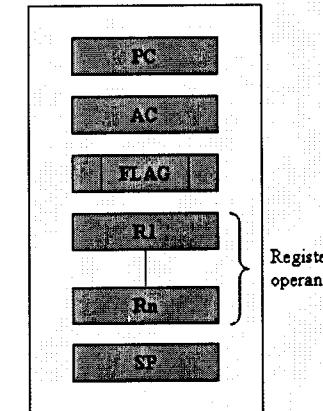


Gambar 5.2 Prosesor sebagai operator data



Gambar 5.3 Prosesor sebagai master bus

Pada level arsitektur, prosesor terdiri atas sumber daya hardware yang dapat dialami program untuk set instruksi seperti program counter, akumulator, register operand, stack, vektor interupsi, status flag, dan lain-lain (Gambar 5.4). Setiap instruksi menentukan sebuah fungsi utama yang dikenal dengan makro-operasi. Makro-operasi yang tepat untuk suatu instruksi ditunjukkan di dalam bagian opcode.



Gambar 5.4 Level arsitektur prosesor

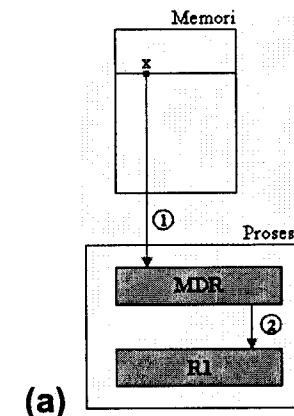
Pada level transfer register, prosesor mempunyai sekumpulan modul-modul digital seperti register, adder, counter, multiplexer, decoder, flip-flop dan sebagainya, masing-masing melakukan satu atau lebih fungsi khusus. Setiap makro-operasi melibatkan satu atau lebih operasi transfer register. Misalnya, makro-operasi MOVE (salin data dari lokasi memori ke register) memerlukan dua operasi transfer register (Gambar 5.5) yaitu baca memori dan transfer register.

Pada level gerbang, prosesor terdiri atas sirkuit hardware yang merupakan gabungan mikro-operasi. Pada kebanyakan kasus praktis, ada overlap antara level transfer register dan level gerbang karena penggunaan LSI dan VLSI.

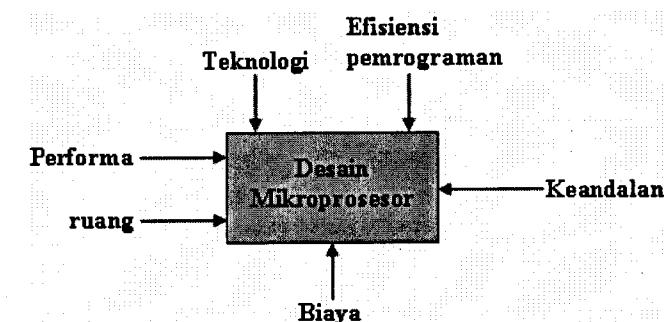
5.2 TUJUAN DESAIN PROSESOR

Semua tujuan desain prosesor adalah untuk mempertemukan kebutuhan set instruksi. Hal ini melibatkan hardware yang tepat agar semua makro-operasi dieksekusi secara akurat seperti pada setiap persyaratan instruksi. Desainer harus mempertimbangkan faktor-faktor lainnya yang bertanggung jawab pada sukses atau gagalnya prosesor di pasar komersial. Gambar 5.6 mengilustrasikan faktor-faktor ini yang memengaruhi desain prosesor. Keputusan-keputusan penting lainnya dibuat pada tingkat awal seperti berikut:

1. Apakah instruksi titik-mengambang akan dieksekusi oleh hardware atau simulasi software?
2. Apa yang kita butuhkan:
 - a. Apakah prosesor harga murah dengan struktur bus tunggal dan register hardware terbatas dan sumber daya lain?
 - b. Apakah prosesor kinerja tinggi dengan multi bus dan sumber daya yang banyak dan/atau paralelisme?

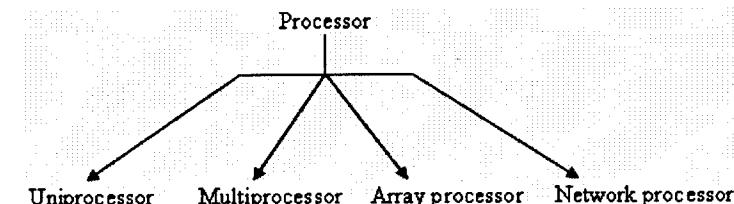


Gambar 5.5 Makro-operasi MOVE dengan dua transfer register



Gambar 5.6 Faktor-faktor yang memengaruhi desain prosesor

Perluasan desain bervariasi pada tipe prosesor. Gambar 5.7 menunjukkan beberapa tipe prosesor.

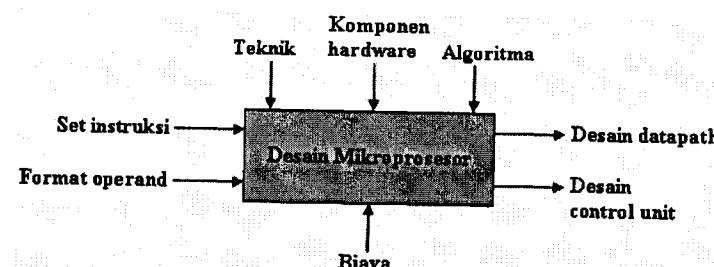


Gambar 5.7 Tipe-tipe prosesor

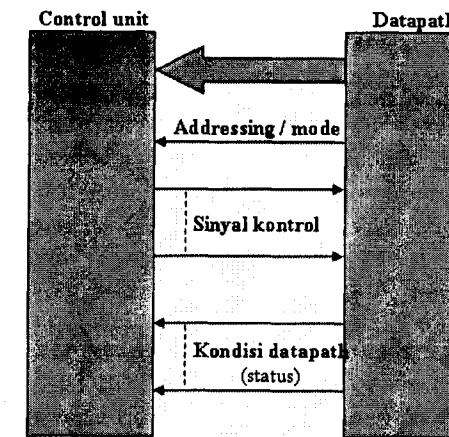
5.3 PROSES DESAIN PROSESOR

Desain sebuah prosesor mengikuti urutan langkah setiap level yang terlibat dalam aktivitas desain. Beberapa langkah saling bergantung. Gambar 5.8 menunjukkan input dan output aktivitas desain prosesor. Ada dua jenis input. Pertama adalah seperti yang ditunjukkan pada sisi kiri dari blok yang diselesaikan oleh arsitek komputer, termasuk penjelasan/deskripsi set instruksi dan format-format operand. Kedua adalah ditunjukkan seperti input dari atas yang berfungsi sebagai basis pengetahuan, termasuk detail berbagai algoritma, fungsi dan tingkah laku dari beberapa komponen hardware, dan pengetahuan beberapa teknik desain. Desainer akan mengevaluasi kegunaan dan keterbatasan dari opsi yang tersedia dan mengambil yang tepat.

Prosesor secara fungsional dibagi menjadi datapath dan unit kontrol. Gambar 5.9 menunjukkan antarmuka antara dua unit ini. Antarmuka ini diimplementasikan pada prosesor murah dengan sebuah bus tunggal. Bus yang sama bertindak sebagai bus internal di dalam datapath. Pada prosesor berbasis kinerja sedang, disediakan dua bus, sedangkan pada prosesor kinerja tinggi mempunyai tiga bus. Secara fisik terdapat perbedaan yang mencolok antara datapath dan unit kontrol. Pada komputer kontemporer (mutakhir), kedua unit hardware tersebut terintegrasi dalam suatu modul fisik tunggal dibandingkan dengan komputer mainframe sebelumnya.



Gambar 5.8 Input dan output dari proses desain prosesor



Gambar 5.9 Antarmuka fungsional antara datapath dan unit kontrol

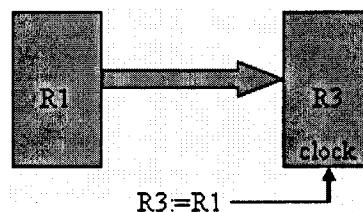
5.3.1 Langkah-Langkah Desain Prosesor

Desain sebuah prosesor dijelaskan dalam urutan langkah berikut:

1. Mengerti dengan baik setiap instruksi. Menetapkan makro-operasi dalam hubungannya dengan arsitektur komputer yang diberikan.
2. Menetapkan sumber daya hardware yang dibutuhkan (untuk mengimplementasikan makro-operasi) dalam hubungannya dengan item hardware yang terlihat oleh pemrogram (register, flag, stack, dan sebagainya).
3. Menerjemahkan setiap makro-operasi dengan satu atau lebih operasi transfer register.
4. Mendesain datapath (dimulai dengan penetapan sumber daya pada langkah 2) yang diperlukan untuk pelaksanaan operasi-operasi transfer register dan penetapan titik-titik kontrol. Menganalisis apakah sirkuit datapath dapat dikurangi dengan menggabungkan desain untuk berbagai instruksi dan mengeliminasi sirkuit-sirkuit yang berlebihan.
5. Daftar urutan pewaktuan sinyal kontrol diperlukan untuk mengaktifkan titik-titik kontrol.

5.3.2 Bahasa Transfer Register

Bahasa transfer register (*register transfer language*, RTL) adalah sebuah notasi yang digunakan untuk menentukan transfer mikro-operasi antar register. RTL adalah sebuah deskripsi yang digunakan untuk menjelaskan tingkah laku instruksi dan organiasi sebuah komputer. Misalnya, statement RTL $R3 := R1$ menunjukkan suatu transfer register sederhana yang melibatkan dua register R1 dan R3. Aksi statement tersebut yaitu isi register R1 ditransfer (disalin) ke register R3. Statement RTL tersebut dapat juga dituliskan dengan $R3 \leftarrow R1$. Gambar 5.10 menunjukkan datapath untuk mikro-operasi ini.



Gambar 5.10 Diagram blok untuk transfer register

Mikro-operasi dikelompokkan ke dalam empat tipe:

1. *Mikro-operasi transfer register*: menyalin isi salah satu register ke register lain tanpa mengubah isi sumber.
2. *Mikro-operasi aritmetika*: melakukan operasi aritmetika pada data dalam register: penjumlahan, pengurangan, increment, decrement, dan pergeseran adalah mikro-operasi yang umum. Tabel 5.1 menjelaskan hal ini.

TABEL 5.1 Mikro-operasi aritmetika

No	Notasi RTL	Keterangan
1	$R5 := R1 + R3$	Isi register R1 dan R3 dijumlahkan dan hasilnya disimpan di R5
2	$R5 := R1 - R3$	Isi R3 dikurangkan dari isi R1 dan hasilnya di R5
3	$R3 := \overline{R3}$	Isi R3 dikomplemenkan

No.	Notasi RTL	Keterangan
4	$R3 := \overline{R3} + 1$	Isi R3 diubah ke komplemen-2
5	$R5 := R1 + \overline{R3} + 1$	Isi R1 ditambahkan dengan komplemen-2 dari R3
6	$R1 := R1 + 1$	Isi R1 di-increment 1
7	$R1 := R1 - 1$	Isi R1 di-decrement 1

3. *Mikro-operasi logika*: melakukan operasi manipulasi bit pada data dalam register. Mikro-operasi ini digunakan untuk pembuatan keputusan logika dan juga untuk manipulasi bit dari data biner. Berikut adalah simbol khusus yang digunakan untuk beberapa mikro-operasi logika:

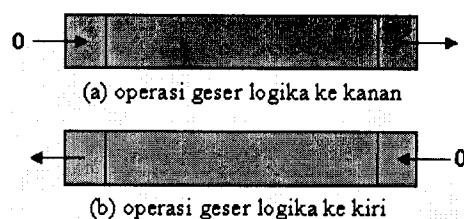
\vee : operasi logika OR

\wedge : operasi logika AND

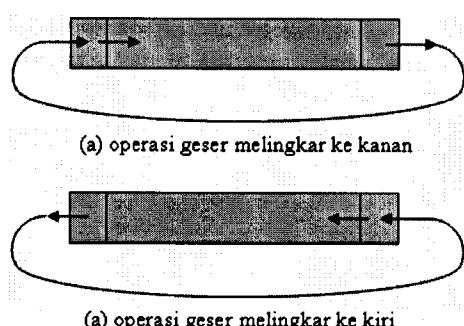
\oplus : operasi exclusive OR

4. *Mikro-operasi pergeseran*: melakukan operasi pergeseran pada data dalam register. Mikro-operasi ini ada tiga macam: operasi geser logika, operasi geser melingkar (disebut juga rotasi), dan operasi geser aritmetika.

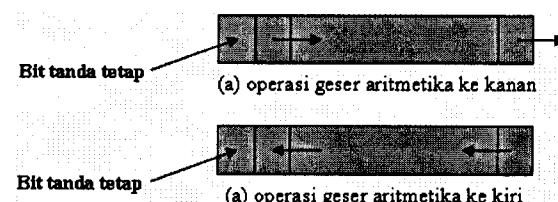
- Pada operasi geser logika, semua bit (termasuk bit tanda) mengalami pergeseran. Bit 0 masuk ke dalam posisi bit kosong (terkiri atau terkanan) seperti yang ditunjukkan pada Gambar 5.11.
- Pada operasi geser melingkar, bit digeser keluar dari satu posisi bit ekstrem masuk ke sisi ekstrem lain seperti yang ditunjukkan pada Gambar 5.12.
- Pada operasi geser aritmetika, bit tanda tidak terpengaruh dan bit lainnya mengalami pergeseran seperti yang ditunjukkan pada Gambar 5.13.



Gambar 5.11 Operasi geser logika



Gambar 5.12 Operasi geser melingkar



Gambar 5.13 Operasi geser aritmetika

5.4 ORGANISASI DATAPATH

Istilah datapath termasuk komponen-komponen hardware berikut:

1. ALU
2. Register sebagai penyimpanan sementara

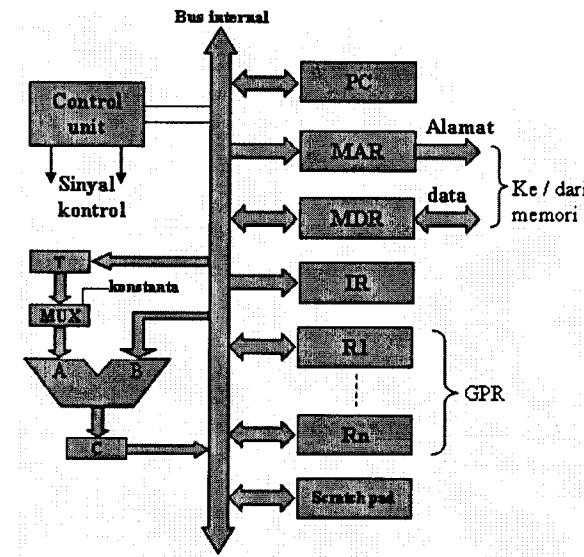
3. Berbagai sirkuit digital untuk pengeksekusian mikro-operasi, termasuk komponen-komponen hardware seperti gate, latch, flip-flop, multiplexer, decoder, counter, complementer, delay logic, dan sebagainya.
4. Lintasan internal untuk pergerakan data antara ALU dan register.
5. Sirkuit driver untuk mentransmisikan sinyal ke unit eksternal (unit kontrol, memori, I/O).
6. Sirkuit receiver untuk menerima sinyal dari unit eksternal.

5.4.1 Datapath Titik-Tetap

Operasi-operasi aritmetika biner titik-tetap sangat penting dalam ALU. Empat jenis operasi yang dibutuhkan: penjumlahan, pengurangan perkalian, dan pembagian. Ada dua strategi untuk mengimplementasikannya:

1. Desain sirkuit ALU untuk keempat operasi tersebut. Pendekatan ini mengakibatkan biaya ALU mahal tetapi cepat.
2. Desain sirkuit ALU hanya untuk penjumlahan dan pengurangan sedangkan operasi perkalian dan pembagian dilakukan dengan menggunakan software. Pendekatan ini mengakibatkan ALU lebih lambat tetapi murah.

ALU terdiri atas sebuah adder dan sejumlah sirkuit seperti counter, shifter, mutiplexer, dan sebagainya. Pada CPU kecil, jumlah register dan komponen-komponen hardware lainnya terbatas dan sebuah bus tunggal yang menghubungkan komponen-komponen datapath. Gambar 5.14 menunjukkan organisasi datapath sederhana. Kecepatan datapath yang demikian terbatas karena mikro-operasi melibatkan pergerakan data yang harus dilaksanakan secara sekuenzial satu setelah yang lain melalui bus internal. Pada CPU besar, di mana kinerja merupakan perhatian utama, terdapat lebih dari satu bus untuk interkoneksi internal dan juga jumlah register yang besar serta berbagai sirkuit hardware untuk melakukan beberapa operasi secara simultan. Gambar 5.15 menunjukkan sebuah datapath dengan dua bus internal. Bus A adalah bus sumber untuk ALU dan bus B adalah bus tujuan bagi ALU. Gambar 5.16 menunjukkan organisasi sebuah datapath kinerja tinggi. Datapath ini menunjukkan tiga bus internal. Bus A dan B adalah sumber input untuk adder, sedangkan bus C membawa output dari adder.

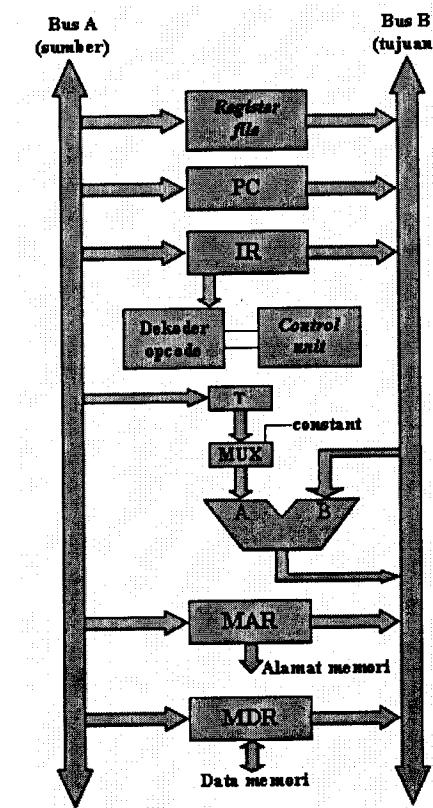


Gambar 5.14 Datapath berbasis bus tunggal

Datapath mempunyai jumlah titik kontrol yang banyak yang diaktifkan oleh sinyal-sinyal kontrol melalui unit kontrol. Selama siklus instruksi (yaitu pengambilan dan pengeksekusian instruksi), unit kontrol menyampaikan rangkaian sinyal kontrol dengan waktu tunda berbeda, bergantung pada instruksi saat itu. Bila suatu sinyal kontrol mengaktifkan suatu titik kontrol sesuai mikro-operasi maka dieksekusi oleh datapath. Beberapa mikro-operasi yang umum antara lain:

1. Geser isi sebuah register
2. Increment sebuah register
3. Decrement sebuah register
4. Pilih satu sinyal keluar dari sejumlah sinyal (*multiplexing*)
5. Tambahkan sejumlah input dari adder
6. Salin isi register ke dalam register lain (*transfer*)
7. Mengkomplemenkan isi register
8. Membersihkan isi register (*reset*)
9. Men-set flip-flop
10. Me-reset flip-flop
11. Memasukkan konstanta ke dalam register (*preset*)

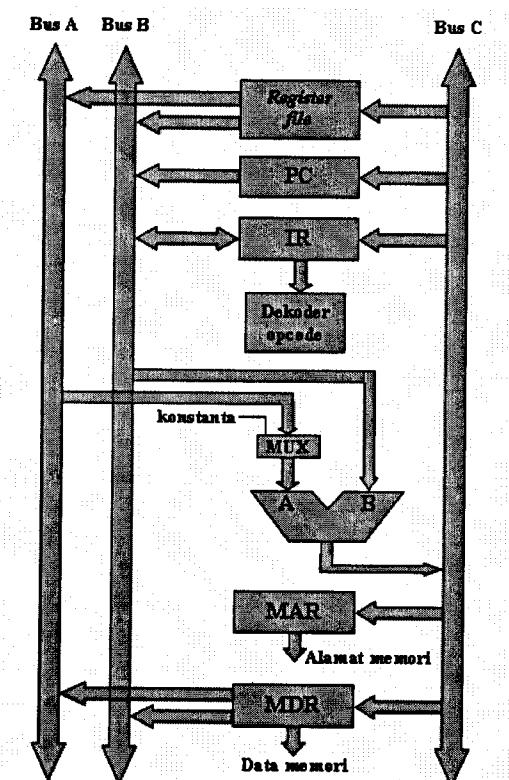
12. Membaca dari memori
13. Menulis ke memori
14. Membaca dari input port
15. Menulis ke output port



Gambar 5.15 Datapath prosesor dua bus

Arti dari mikro-operasi dapat dimengerti sepenuhnya jika algoritma yang digunakan untuk operasi aritmetika diketahui. Pada sebuah CPU tertentu, operasi aritmetika dapat diimplementasikan sepenuhnya oleh datapath atau dengan menggabungkan datapath dan unit kontrol. Misalnya, sebuah operasi perkalian memiliki sirkut ALU yang terdiri atas sirkuit hardware yang banyak. Sebagai alternatif, ALU bisa hanya mempunyai hardware penjumlahan dan pengurangan, sedangkan perkalian

akan dilakukan oleh serangkaian penjumlahan dan pergeseran kiri. Unit kontrol membangkitkan sinyal kontrol yang diperlukan dalam rentetan waktu yang dipersyaratkan ke datapath.

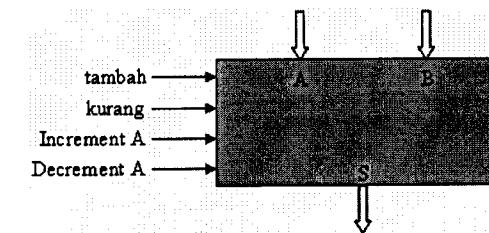


Gambar 5.16 Datapath berbasis tiga bus

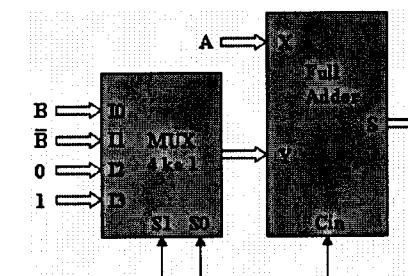
5.4.2 Desain Unit Aritmetika

Unit aritmetika melakukan beberapa mikro-operasi aritmetika seperti penjumlahan, pengurangan, komplementasi, inkrementasi, dekrementasi, dan sebagainya. Gambar 5.17 mengilustrasikan sebuah diagram blok unit aritmetika. Dengan menggunakan sebuah penjumlahan paralel tunggal, semua mikro-operasi ini dapat dilaksanakan dengan menggunakan input yang tepat ke dalam adder seperti yang ditunjukkan pada Gambar 5.18. Dengan

mengacu pada diagram logika, multiplexer quad 4 ke 1 melakukan routing input yang sesuai ke input adder Y. Tabel 5.2 menunjukkan delapan kombinasi S_1, S_0 dan C_{in} dan input adder Y yang sesuai dan output adder S. Input adder X merupakan operand pertama, A. Jadi mikro-operasi aritmetika yang diperlukan dapat dilakukan dengan menggunakan masukan sinyal yang sesuai ke S_1, S_0 dan C_{in} . Walupun ada delapan kombinasi input, sirkuit aritmetika ini hanya melakukan tujuh mikro-operasi berbeda. Kasus 5 dan 8 melakukan mikro-operasi yang sama yaitu transfer A ke S. Dalam desain kedua kombinasi ini dapat dipilih salah satu. Kasus ketujuh harus dianalisis dengan hati-hati. Kombinasi input S_1, S_0 dan $C_{in} = 110$ mengakibatkan input I_3 terpilih oleh MUX, sehingga mensuplai 1 semua ke input adder Y. Semua 1 misalnya 1, 11, 111, 1111 dan seterusnya tidak ada, melainkan komplemen-2 dari 1, 01, 001, 0001 berturut-turut. Jadi kasus ketujuh adalah menjumlahkan komplemen-2 dari 1 ke X. Hal ini ekivalen dengan mengurangkan 1 dari X. Dengan kata lain, mikro-operasi melakukan "decrement A".



Gambar 5.17 Diagram blok unit aritmetika



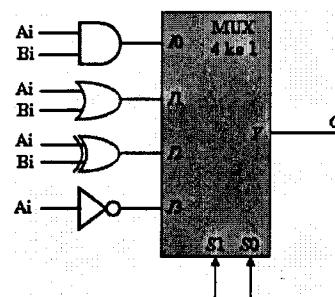
Gambar 5.18 Diagram logika unit aritmetika

TABEL 5.2 Tabel fungsi unit aritmetika

No	S1	S0	Cin	Input MUX	Output MUX	Operasi
1	0	0	0	B	A + B	Tambah
2	0	0	1	B	A + B + 1	Tambah dengan carry
3	0	1	0	\bar{B}	A + \bar{B}	Kurang dengan borrow
4	0	1	1	\bar{B}	A + \bar{B} + 1	Kurang
5	1	0	0	0	A	Transfer A
6	1	0	1	0	A + 1	Increment A
7	1	1	0	1	A - 1	Decrement A
8	1	1	1	1	A	Transfer A

5.4.3 Desain Unit Logika

Penggunaan MUX 4 ke 1 pada setiap posisi bit, unit logika dapat didesain dengan memberikan input yang sesuai ke empat input MUX. Gambar 5.19 menunjukkan desain satu bit yang melakukan empat operasi logika: AND, OR, XOR, dan complement. Tabel 5.3 menunjukkan kombinasi input dan output yang sesuai dan operasi logika dilakukan pada operand A dan B.



Gambar 5.19 Unit logika satu bit

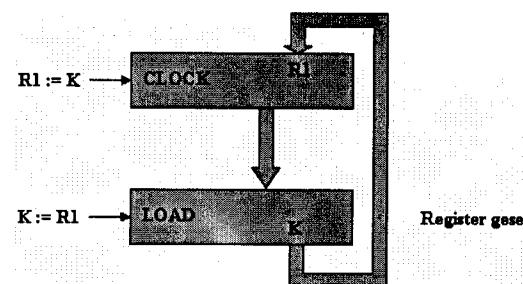
TABEL 5.3 Tabel fungsi unit logika

No	S1	S0	Y	Operasi logika
1	0	0	$A \wedge B$	AND
2	0	1	$A \vee B$	OR
3	1	0	$A \oplus B$	Exclusive OR
4	1	1	\bar{A}	Complement A

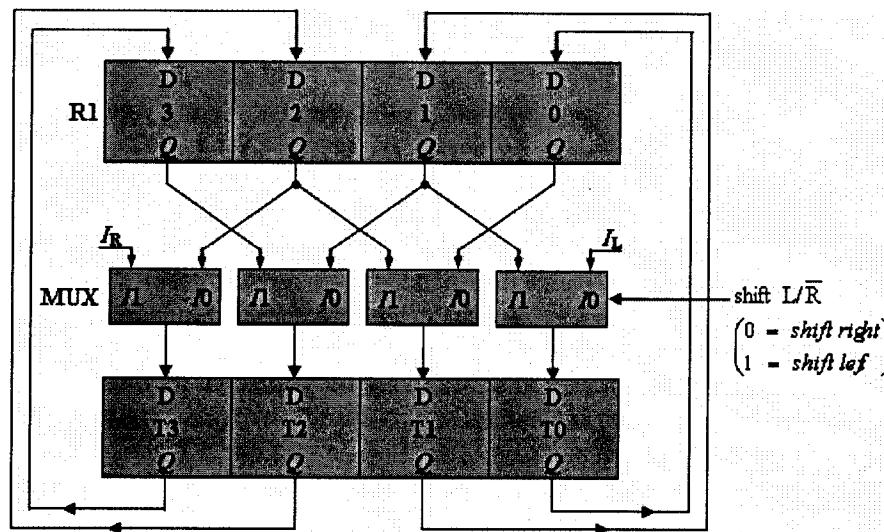
5.4.4 Desain Penggeser (Shifter)

Operasi pergeseran diperlukan dalam ALU karena beberapa alasan: pemrosesan data, perkalian, pembagian, dan sebagainya. Ada dua pendekatan untuk melakukan operasi pergeseran terhadap isi register:

1. Register geser kombinasional yang terpisah dapat bertindak sebagai sumber daya terpusat. Isi register yang akan digeser ditransfer ke input register geser. Output register geser diberikan kembali ke register asal. Gambar 5.20 menunjukkan bagaimana isi R1 digeser dengan menggunakan register geser K. Mikro-operasi K:=R1 mentransfer isi R1 ke dalam input register geser K. Mikro-operasi lain, R1:=K dilaksanakan setelah waktu tunda propagasi register geser. Sekarang output register geser ditransfer ke R1.
2. Register didesain seperti sebuah register geser. Hal ini melibatkan sirkuit hardware tambahan yang diperlukan dalam register R1. Gambar 5.21 menunjukkan bagaimana register R1 dimodifikasi menjadi register geser. Desain memiliki dua set flip-flop untuk setiap bit R1. Flip-flop T0, T1, T2, dan T3 bertindak sebagai kumpulan penyimpan sementara untuk bit-bit yang sesuai dengan R0, R1, R2, dan R3. Sebuah MUX 2 x 2-ke-1 mensuplai ke input D dari setiap flip-flop T. Salah satu input ke MUX mengindikasikan arah pergeseran: kiri atau kanan. Tabel 5.4 menunjukkan tabel fungsi register geser. Efeknya R1 sekarang adalah sebuah register dua bank.



Gambar 5.20 Penggunaan register geser



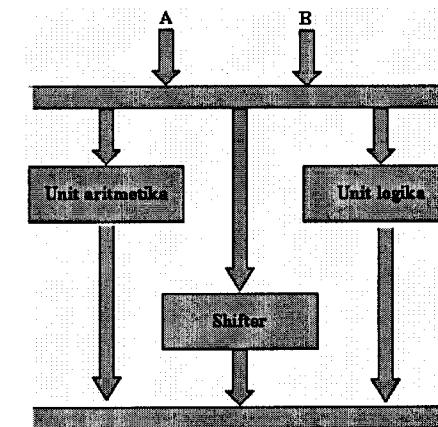
Gambar 5.21 Sirkuit register geser

TABEL 5.4 Tabel kebenaran register geser

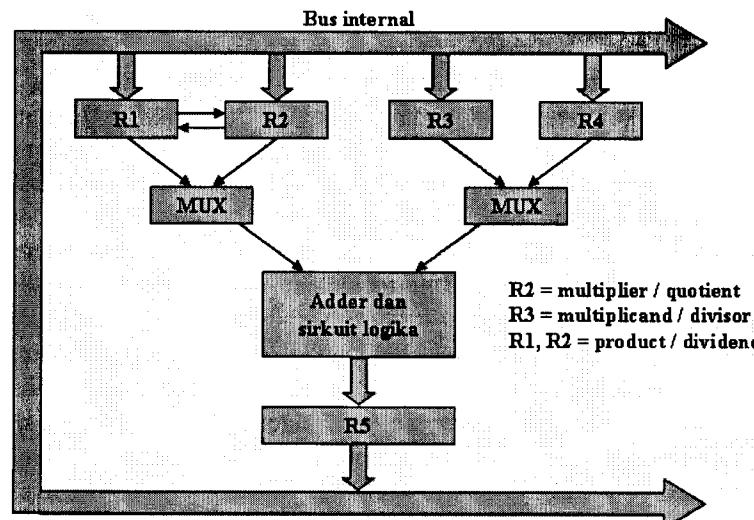
No	shift L/R	T3	T2	T1	T0
1	1	R 1(2)	R 1(1)	R 1(0)	I_L
2	0	I_R	R 1(3)	R 1(2)	R 1(1)

5.4.5 Desain ALU

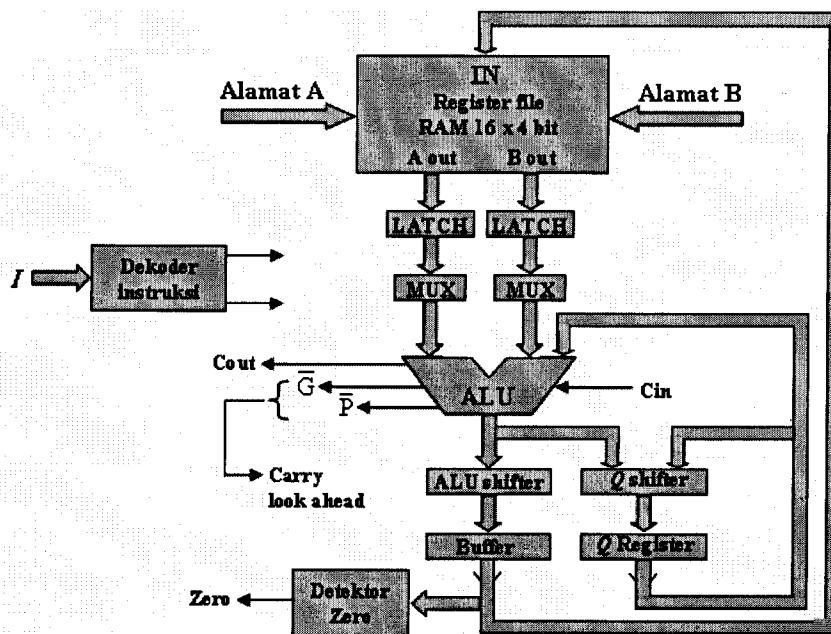
ALU dapat dibangun dari sebuah ALU kombinasional atau ALU sekuensial. ALU kombinasional dapat dibangun dengan menggabungkan desain unit aritmetika, unit logika, dan penggeser (*shifter*). Gambar 5.22 menunjukkan diagram blok ALU kombinasional. Sebuah ALU kombinasional 4-bit tersedia dalam sebuah IC tunggal dengan tipe 74181. Dengan mengkaskadekan beberapa IC 74181, dapat dibangun sebuah ALU dengan word length yang lebih besar. Dimungkinkan memasukkan pengali dan pembagi kombinasional, tetapi dibutuhkan beberapa level logika. Akibatnya, ALU kombinasional menjadi mahal dan lambat dibandingkan dengan ALU sekuensial yang murah. Mereka mengimplementasikan perkalian dan pembagian menggunakan algoritma software. Gambar 5.22(b) menunjukkan sebuah ALU sekuensial. ALU sekuensial 4-bit lengkap disediakan oleh IC AMD 2901 dan AMD 2903. Ini adalah prosesor bit slice yang *micro-programmable*. Gambar 5.23 menunjukkan diagram blok AMD 2903.



Gambar 5.22(a) Diagram blok ALU kombinasional berorientasi bus



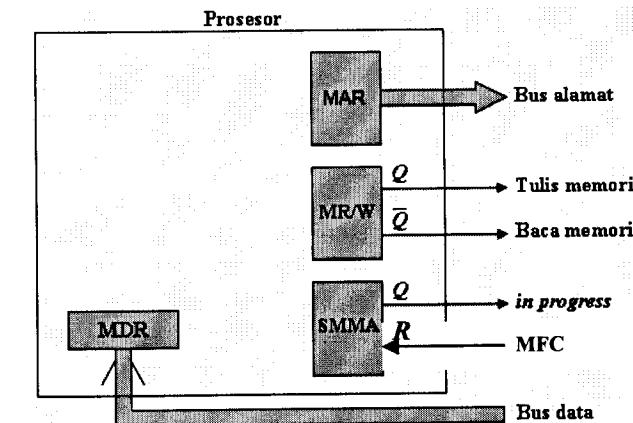
Gambar 5.22(b) ALU sekuensial



Gambar 5.23 Diagram blok AMD 2903

5.5 ANTARMUKA MEMORI UTAMA

Antarmuka antara memori utama dan prosesor ditunjukkan pada Gambar 5.24. Flag MR/W digunakan oleh prosesor untuk menunjukkan operasi yang sedang dilaksanakan. Jika $MR/W=0$, prosesor melakukan operasi baca memori. Jika $MR/W=1$, melakukan operasi tulis memori. Flag SMMA menunjukkan apakah suatu operasi memori telah dalam progress. Jika $SMMA=0$, tidak ada operasi memori dalam progress. Jika $SMMA=1$, operasi memori dalam progress. Pada saat operasi dalam perampungan, memory control logic me-reset flag. Jelas, hal ini akan dilaksanakan setelah waktu akses memori berakhir. Sinyal MFC dari memori dapat digunakan untuk me-reset flag ini.



Gambar 5.24 Antarmuka memori utama

5.5.1 Urutan Pembacaan Memori

Prosesor melakukan operasi-operasi berikut untuk pelaksanaan baca memori:

1. Menempatkan alamat memori dalam MAR.
2. Me-reset flip-flop MR/W. Output \bar{Q} menuju ke sinyal kontrol MEMORY READ.
3. Men-set flag SMMA
4. Memeriksa apakah flag SMMA sudah menjadi 0. Ketika SMMA menjadi 0, prosesor me-load data dari memori ke dalam MDR.

5.5.2 Urutan Penulisan Memor

Prosesor melakukan operasi-operasi berikut untuk pelaksanaan tulis memori:

1. Menempatkan alamat memori dalam MAR
 2. Menempatkan data dalam MDR
 3. Men-set flip-flop MR/W. Output Q menuju ke sinyal kontrol MEMORY WRITE
 4. Men-set flag SMMA
 5. Memeriksa apakah flag SMMA sudah menjadi 0. Ketika SMMA menjadi 0, prosesor memulai operasi lain.

5.6 REGISTER FILE

Register file (penyimpanan lokal) terdiri atas sekumpulan register prosesor (dengan sebuah kontrol read/write bersama). Ada dua tipe register:

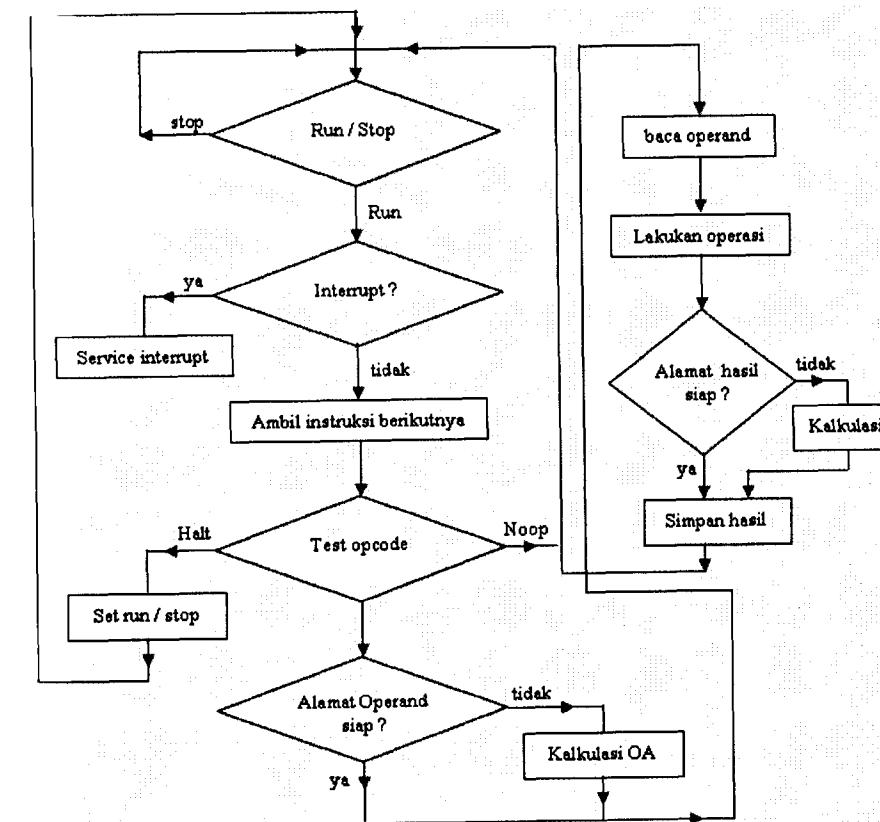
1. *Program addressable register* adalah untuk penyimpanan operand dan hasil. Register ini dikenal dengan *general purpose register*. Register ini dapat juga digunakan sebagai register khusus seperti index register, base register, stack pointer dan lain-lain.
 2. *Scratch pad register* digunakan sebagai register sementara oleh unit kontrol untuk penyimpanan hasil sementara atau konstanta yang diperlukan selama eksekusi suatu instruksi.

Waktu akses register file sangat cepat dibandingkan dengan waktu akses memori utama. Secara fisik, register file berada dalam prosesor. Register file berbeda dengan memori cache yang berfungsi sebagai penyangga/buffer antara prosesor dan memori utama. Memori cache tidak terlihat (*invisible*) oleh program. Program (walaupun kecil) tidak dapat disimpan di register file karena prosesor tidak mengambil instruksi dari register file.

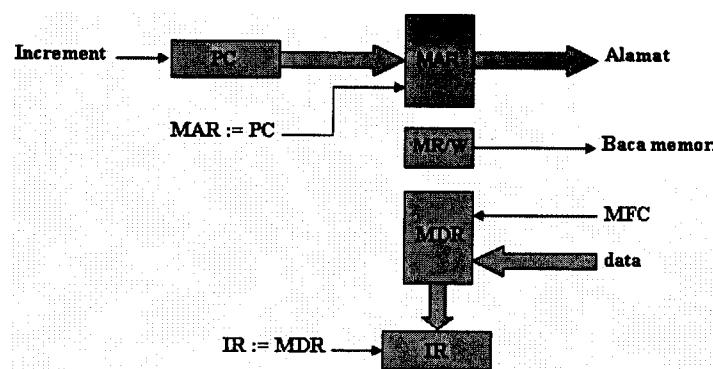
5.7 DATAPATH PADA INSTRUKSI SEDERHANA

Datapath untuk pengambilan (*fetch*) instruksi sama untuk semua instruksi. Gambar 5.25 menunjukkan urutan pengambilan dan Gambar 5.26 menunjukkan hubungan datapath yang terdiri atas PC, MAR, MDR dan IR.

dan juga ditunjukkan titik-titik kontrol dan urutan sinyal kontrol yang diperlukan. Pada akhir fase pengambilan, isi PC dinaikkan (*increment*) oleh panjang byte instruksi yang sedang diambil. Hal ini merupakan persiapan lebih awal untuk pengambilan instruksi berikutnya.



Gambar 5.25 Urutan pengambilan (fetch) instruksi

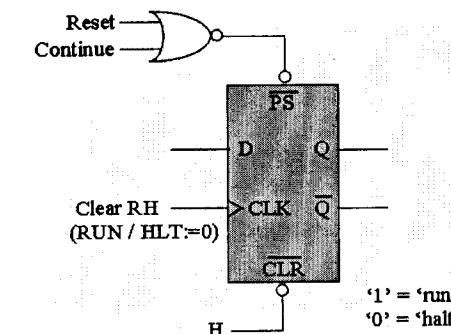


Gambar 5.26 Datapath untuk pengambilan instruksi

5.7.1 Instruksi HALT

Aksi yang dilakukan oleh instruksi HALT adalah menghentikan siklus instruksi. Hal ini digambarkan dengan $\text{RUN}/\text{HALT}=0$. Hubungannya dengan datapath, melibatkan resetting flip-flop RUN/HLT. Gambar 5.27 menunjukkan datapath tersebut. Sinyal kontrol CLEAR RH mengaktifkan titik kontrol yang masuk ke clock flip-flop D. Masukan flip-flop D secara permanen ditarik turun (0). Karena itu tebing positip sinyal CLEAR RH, me-reset flip-flop. Keluaran Q dari flip-flop dirasakan oleh unit kontrol sebelum pengambilan (*fetch*) instruksi dimulai. Jika 0, pengambilan instruksi tidak dilakukan. Unit kontrol secara sederhana tetap menjaga flip-flop RUN/HLT menjadi set kembali. Hal ini dilakukan dengan dua aksi berikut:

1. Sinyal RESET menjadi aktif (HIGH)
2. Sinyal CONTINUE menjadi aktif (HIGH)



Gambar 5.27 Datapath untuk instruksi HALT

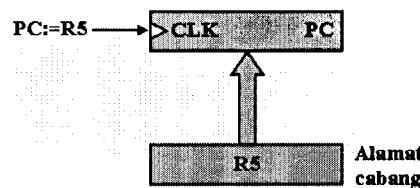
Sinyal gerbang NOR membangkitkan sebuah sinyal LOW ke masukan preset flip-flop. Sinyal RESET dibangkitkan tepat ketika power-on dan pada saat me-reset secara manual karena operator menekan tombol/saklar RESET pada panel depan. Sinyal CONTINUE adalah sinyal khusus yang tidak terdapat pada sebuah komputer sederhana. Hanya terdapat pada sistem khusus seperti sistem multi-prosesor.

5.7.2 Instruksi NOOP

Tidak ada aksi yang dilakukan oleh instruksi NOOP. Karena itu tidak ada datapath untuk instruksi ini. Segera setelah instruksi ini dieksekusi, fase eksekusi telah selesai.

5.7.3 Instruksi JUMP

Instruksi JUMP bercabang ke alamat instruksi seperti yang ditentukan oleh instruksi JUMP. Hal ini digambarkan dengan $\text{PC}:=\text{BA}$ di mana BA adalah alamat cabang yang diberikan oleh instruksi. Alamat ini umumnya dikalkulasi sesuai dengan mode pengalamatan dalam instruksi. Anggap bahwa kalkulasi alamat telah lengkap dan BA tersedia dalam register ALU R5, mikro-operasi yang dikehendaki adalah $\text{PC}:=\text{R5}$. Gambar 5.28 menunjukkan datapath untuk instruksi JUMP.



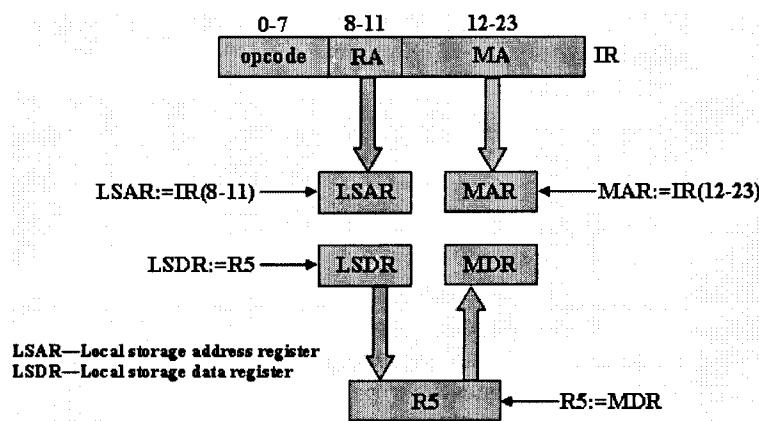
Gambar 5.28 Datapath untuk instruksi JUMP

5.7.4 Instruksi LOAD

Operasi yang dilakukan oleh instruksi LOAD adalah menyalin isi suatu lokasi memori ke dalam suatu register, seperti yang ditetapkan dalam instruksi. Hal ini digambarkan dengan $<\text{RA}>:=<\text{MA}>$ di mana MA adalah alamat memori dan RA adalah alamat register. Urutan operasi yang diperlukan untuk mencapai ini adalah:

1. Melaksanakan pembacaan memori dari alamat MA
2. Menyimpan data dalam register RA

Organisasi datapath ditunjukkan pada Gambar 5.29. Pada akhir pengambilan instruksi, instruksi tersedia dalam IR. Mari kita menganggap bahwa bit 0-7 menyediakan opcode, bit 8-11 menyediakan RA dan bit 12-23 menyediakan MA.



Gambar 5.29 Datapath untuk instruksi LOAD

Berikut diberikan register transfer statement disertai dengan penjelasannya:

1. $\text{MAR}:=\text{IR}(12-23)$ Memasukkan MA ke dalam register alamat memori MAR
2. $\text{MR/W}:=0$ Me-reset read/write flag memori; menandakan pembacaan
3. $\text{SMMA}:=1$ Men-set awal main memory access flag
4. $\text{R5}:=\text{MDR}$ Mentransfer isi MDR ke dalam R5
5. $\text{LSDR}:=\text{R5}$ Mentransfer isi R5 ke dalam local storage data register
6. $\text{LSAR}:=\text{IR}(8-11)$ Mentransfer RA ke dalam local storage address register
7. $\text{LSR/W}:=1$ Men-set local storage read/write flag; menandakan penulisan
8. $\text{SLSA}:=1$ Men-set awal local storage access flag

Unit kontrol akan menjamin bahwa waktu tunda antara operasi 3 dan 4 lebih besar dari waktu akses memori utama. Dengan cara yang sama, waktu tunda antara operasi 8 dan operasi penyimpanan lokal (register file) berikutnya lebih besar dari waktu akses penyimpanan lokal.

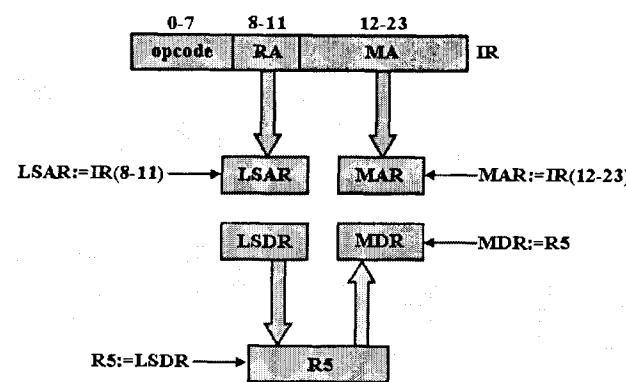
Delapan register transfer statement dapat langsung digunakan sebagai mikro-operasi.

5.7.5 Instruksi STORE

Operasi yang dilakukan oleh instruksi STORE adalah kebalikan dari instruksi LOAD. Isi dari suatu register disalin ke dalam suatu lokasi memori utama. Hal ini digambarkan dengan $<\text{MA}>:=<\text{RA}>$. Urutan operasi yang diperlukan adalah:

1. Membaca dari penyimpanan lokal isi register yang diberikan oleh RA.
2. Menulis pada lokasi memori utama MA.

Organisasi datapath ditunjukkan pada Gambar 5.30. Pada akhir pengambilan instruksi, instruksi tersedia pada IR. Mari kita menganggap bahwa bit 0-7 menyediakan opcode, bit 8-11 menyediakan RA dan bit 12-23 menyediakan MA.



Gambar 5.30 Datapath untuk instruksi *STORE*

Berikut diberikan register transfer statement disertai dengan penjelasannya:

1. $LSAR := IR(8-11)$ Mentransfer RA ke dalam local storage address register
2. $LSR/W := 0$ Me-reset local storage read/write flag; menandakan pembacaan
3. $SLSA := 1$ Men-set awal local storage access flag
4. $R5 := LSDR$ Mentransfer isi LSDR ke dalam R5
5. $MDR := R5$ Mentransfer isi R5 ke dalam main memory data register
6. $MAR := IR(12-23)$ Memasukkan MA dalam main memory data register
7. $MR/W := 1$ Men-set main memory read/write flag; menandakan penulisan
8. $SMMA := 1$ Men-set awal main memory access flag

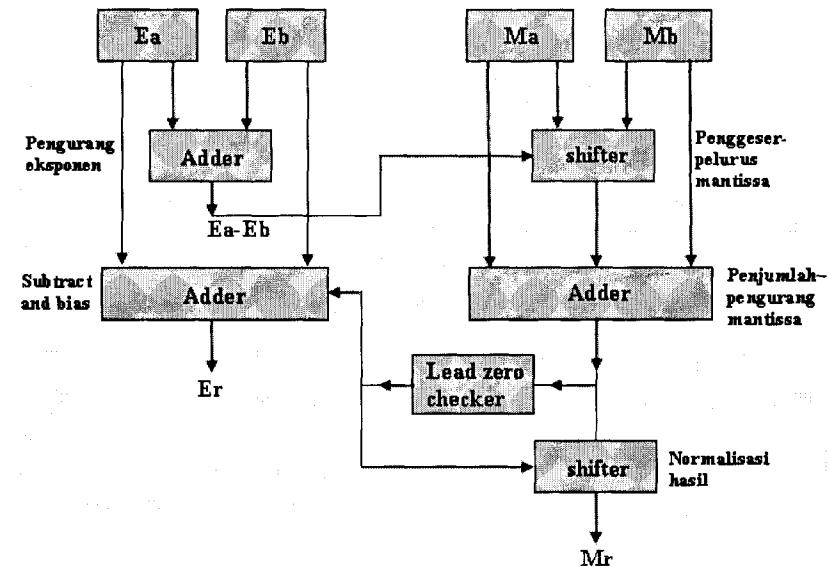
Unit kontrol akan menjamin bahwa waktu delay antara operasi 3 dan 4 lebih besar dari waktu akses penyimpanan lokal (register file). Dengan cara yang sama, waktu tunda antara operasi 8 dan operasi penyimpanan lokal berikutnya lebih besar dari waktu akses memori utama.

Delapan register transfer statement dapat langsung digunakan sebagai mikro-operasi.

5.8 DATAPATH PADA UNIT TITIK-MENGAMBANG

Gambar 5.31 menunjukkan organisasi khas unit titik-mengambang yang umum terdapat pada komputer tingkat menengah dahulu. Ada dua bagian:

1. unit eksponen
2. unit mantissa



Gambar 5.31 Unit titik-mengambang generik

Langkah berikut adalah proses penjumlahan atau pengurangan titik-mengambang:

1. membandingkan eksponen
2. menyajarkan mantissa pada salah satu operand
3. penjumlahan atau pengurangan mantissa.

Unit mantissa tidak memerlukan hardware eksklusif. Unit titik-mengambang mendukung semua empat operasi dasar (add, sub, multiply, dan divide) dan dapat juga digunakan sebagai unit mantissa. Unit eksponen harus melakukan tiga operasi berbeda:

1. Penjumlahan eksponen

2. Pengurangan eksponen
3. Pembandingan eksponen

Penjumlahan eksponen digunakan untuk menentukan $E_a - E_b$. Eksponen yang lebih besar ditentukan berdasarkan tanda dari pengurangan ini. Magnitude selisih (E) memberikan jumlah pergeseran yang diperlukan untuk mantissa dari bilangan yang mempunyai eksponen yang lebih besar.

RINGKASAN

Secara fungsional prosesor dibagi dalam datapath dan unit kontrol. Datapath memainkan peranan penting dalam prosesor. Organisasinya berpengaruh langsung pada kinerja sistem dan efisiensi pemrograman. Prosesor dapat ditinjau pada empat level berbeda: Level Sistem, Level Set Instruksi atau Level Arsitektur, Level Transfer Register, dan Level Gerbang.

Tujuan perancangan prosesor secara keseluruhan adalah mempertemukan kebutuhan set instruksi. Hal ini melibatkan desain hardware yang tepat agar semua makro-operasi dapat dieksekusi secara akurat seperti yang dibutuhkan oleh instruksi. Perancangan suatu prosesor dilakukan secara sistematik mengikuti suatu urutan langkah. Bahasa transfer register adalah notasi yang digunakan untuk menetapkan transfer mikro-operasi antar register. Bahasa transfer register merupakan sebuah tool untuk menjelaskan tingkah laku instruksi dan organisasi dari sebuah komputer.

Mikro-operasi dikelompokkan ke dalam empat jenis.

1. Mikro-operasi transfer register
2. Mikro-operasi aritmetika
3. Mikro-operasi logika
4. Mikro-operasi Pergeseran

Datapath meliputi komponen-komponen hardware berikut:

1. ALU
2. Register untuk penyimpanan sementara
3. Berbagai sirkuit digital untuk pengeksekusian berbagai mikro-operasi. Hal ini termasuk komponen-komponen hardware seperti gate, flip-flop, multiplexer, decoder, counter, completer, delay logic, dan lain-lain.
4. Internal path untuk perpindahan data antara ALU dan register-register
5. Sirkuit driver untuk mentransmisikan sinyal-sinyal ke unit-unit eksternal (unit kontrol, memori, I/O)
6. Sirkuit receiver untuk menerima sinyal-sinyal dari unit-unit eksternal.

Datapath mempunyai jumlah titik-titik kontrol yang banyak yang diaktifkan oleh sinyal-sinyal kontrol yang diberikan oleh unit kontrol.

Selama siklus instruksi (pengambilan dan pengeksekusian suatu instruksi), unit kontrol memberikan urutan sinyal-sinyal dengan waktu tunda yang berbeda bergantung pada instruksi sekarang. Bila suatu sinyal kontrol mengaktifkan suatu titik kontrol maka mikro-operasi yang sesuai dieksekusi oleh datapath.

ALU dapat berasal dari sebuah ALU kombinasional atau sekuensial. ALU kombinasional adalah suatu sirkuit kombinasional yang dapat melaksanakan operasi aritmetika, logika, dan pergeseran. Kombinasional ALU dapat didesain dengan mengkombinasikan desain-desain unit aritmetika, unit logika dan penggeser (shifter). Dimungkinkan memasukkan pengali dan pembagi kombinasional tetapi membutuhkan beberapa level logika. Karena itu menjadikannya mahal dan lambat. Dengan kata lain, ALU sekuensial murah. Mereka mengimplementasikan perkalian dan pembagian dengan menggunakan algoritma.

ALU unit titik-mengambang sepenuhnya berdiri sendiri atau unit-unit mantissa-nya menggunakan ALU unit titik-tetap. Saat ini suatu kecenderungan dalam industri mikroprosesor menggunakan unit titik-mengambang sebagai prosesor tujuan khusus yang bekerja secara baik bersama dengan sebuah prosesor integer seperti sebuah coprocessor. Mikroprosesor seperti 80486 DX dan seri Pentium mempunyai coprocessor terintegrasi dalam chip mikroprosesor tersebut.

SOAL-SOAL ULANGAN

- Prosesor dapat dipandang dengan empat level berbeda yaitu (1) level sistem, (2) level set instruksi atau level arsitektur, (3) level _____, dan (4) level _____.
- Pada level _____, prosesor terdiri atas sumber daya hardware yang dapat dialami program untuk set instruksi seperti program counter, akumulator, register operand, stack, vektor interupsi, status flag, dan lain-lain.
- Setiap instruksi menentukan sebuah fungsi utama yang dikenal dengan makro-operasi. Makro-operasi yang tepat untuk suatu instruksi ditunjukkan di dalam bagian _____.
- Pada level _____, prosesor terdiri atas sirkuit hardware yang melaksanakan mikro-operasi.

- Prosesor secara fungsional dibagi menjadi dua bagian besar yaitu: (1) datapath, dan (2) _____.
- _____ adalah sebuah notasi yang digunakan untuk menentukan transfer mikro-operasi antar register dan merupakan deskripsi yang digunakan untuk menjelaskan tingkah laku instruksi dan organisasi sebuah komputer.
- Mikro-operasi dikelompokkan ke dalam empat tipe yaitu: (1) Mikro-operasi transfer register, (2) mikro-operasi aritmetika, (3) _____, dan (4) _____.
- Mikro-operasi _____ adalah menyalin isi salah satu register ke register lain tanpa mengubah isi sumber.
- Mikro-operasi _____ adalah mikro-operasi yang digunakan untuk pembuatan keputusan logika dan juga untuk manipulasi bit dari data biner.
- ALU terdiri atas sebuah _____ dan sejumlah sirkuit seperti counter, shifter, multiplexer, dan sebagainya.

SOAL-SOAL LATIHAN

- Kalkulasi waktu yang digunakan untuk pengambilan instruksi dalam suatu CPU yang mempunyai waktu tunda propagasi:
 - operasi setiap transfer register: 10 ns
 - waktu akses memori utama: 80 ns
 Penyelesaian satu mikro-operasi ke awal berikutnya dianggap memiliki waktu tunda nol, tunjukkan kalkulasi Anda secara terpisah.
- Instruksi manakah berikut ini yang menggunakan waktu lebih besar untuk siklus instruksi?
 - LDA
 - STA
 - NOOP
 - HALT
 - BUN (*branch unconditionally*)
 Buktikan jawaban Anda dengan estimasi kuantitatif.
- Instruksi yang manakah pada nomor 2 yang menggunakan waktu yang lebih kecil untuk siklus instruksi? Buktikan jawaban Anda dengan estimasi kuantitatif.
- Berikan notasi transfer register untuk instruksi berikut:

(a) MOV AL,BL	(b) ADC AL,45h
(c) SUB BL,CL	(d) INC CX

BAB 6

UNIT KONTROL

Sasaran bab ini:

1. Fungsi Unit Kontrol

2. Organisasi *Hardwired Control Unit*

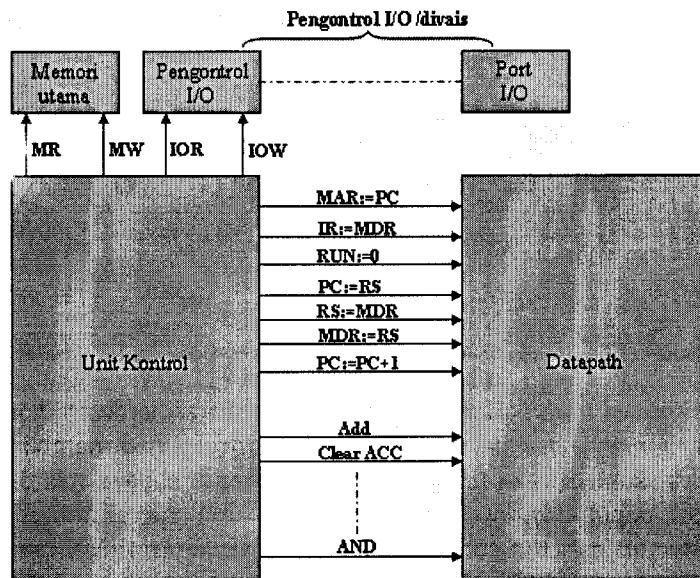
3. Organisasi *Microprogrammed Control Unit*

Unit Kontrol merupakan ‘otak’ atau pusat syaraf hardware komputer. Unit kontrol mengawasi pelaksanaan siklus instruksi dan membangkitkan sinyal-sinyal kontrol relevan pada saat yang tepat supaya mikro-operasi yang tepat dapat dikerjakan pada CPU dan unit-unit eksternal lainnya seperti memori dan I/O controller/devices. Unit kontrol dirancang untuk suatu organisasi datapath spesifik (ALU, register, dan sebagainya). Bagian ini membahas operasi unit kontrol dan kedua teknik perancangan unit kontrol: *hardwired* dan *micropogramming*.

6.1 FUNGSI UNIT KONTROL

Unit kontrol bertanggung jawab pada pengoordinasian aktivitas dalam komputer. Gambar 6.1 memberikan gambaran terhadap keseluruhan fungsi/peranan unit kontrol. Sinyal-sinyal kontrol disampaikan oleh unit kontrol untuk mencapai berbagai *hardware logic* di dalam prosesor dan unit-unit eksternal lainnya. Sinyal-sinyal ini yang memulai operasi-operasi dalam komputer. Mikrooperasi-mikrooperasi dilaksanakan bila sinyal kontrol yang relevan mengaktifkan titik-titik kontrol. Memori utama dikontrol oleh dua sinyal kontrol: *memory read* dan *memory write*. Semua pengontrol I/O menerima sinyal kontrol IOR dan IOW. Datapath adalah unit yang menerima sinyal kontrol yang terbanyak. Bagaimana unit kontrol mengetahui sinyal kontrol yang akan disampaikan? Dia mempunyai

semacam daftar tugas yang diberitahu oleh program apa yang harus dikerjakan/dieksekusi.

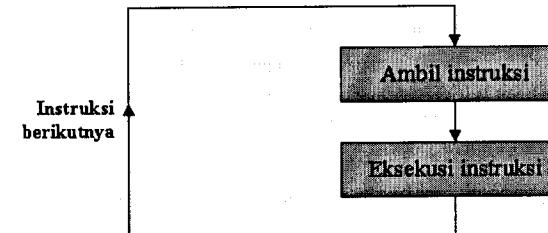


Gambar 6.1 Fungsi-fungsi unit kontrol

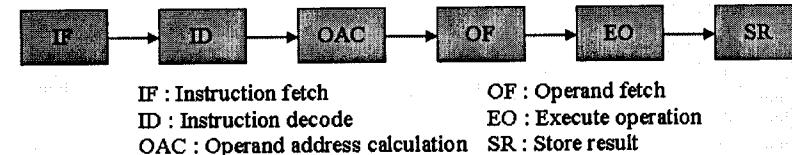
Pada Tabel 6.1 diberikan hubungan jenis-jenis instruksi dan aksi-aksi CPU. Prosesor mengeksekusi program dengan melakukan siklus-siklus instruksi seperti yang ditunjukkan pada Gambar 6.2. Setiap siklus instruksi terdiri atas beberapa langkah seperti yang ditunjukkan pada Gambar 6.3. Dua langkah pertama (pengambilan instruksi dan dekode instruksi) diperlukan oleh semua instruksi. Ada atau tidaknya sisa langkah berikutnya adalah bergantung pada masing-masing instruksi. Hal ini dijelaskan pada Tabel 6.2, Tabel 6.3, dan Tabel 6.4 yang memberikan tindakan-tindakan yang diperlukan pada beberapa instruksi yang dikenal.

TABEL 6.1 Jenis-jenis instruksi dan tindakan CPU

No	Tipe Instruksi	Tindakan CPU	Keterangan
1	Transfer data	Salin informasi; membaca dari sumber dan menulisnya ke target	Sumber atau target atau keduanya bisa dari memori
2	Aritmetika	Melakukan operasi ALU yang diperlukan dan men-set kode kondisi dan flag	Operand-operand harus dibawa ke ALU jika tidak tersedia dalam ALU
3	Logika	Sama dengan di atas	Sama dengan di atas
4	Kontrol program	Program counter diperbarui	Pencabangan
5	I/O	Memasukkan atau mengeluarkan data; melakukan siklus baca I/O atau siklus bus tulis I/O	Melakukan transfer data



Gambar 6.2 Siklus instruksi



Gambar 6.3 Langkah-langkah dalam siklus instruksi

Fungsi unit kontrol secara keseluruhan dapat diringkaskan sebagai berikut:

1. Mengambil/membaca instruksi
2. Mendekode/interpretasi opcode dan mode pengalamatan
3. Membangkitkan sinyal kontrol yang diperlukan yang sesuai dengan instruksi (opcode dan mode pengalamatan) dalam urutan waktu yang tepat agar mikrooperasi-mikrooperasi yang relevan dikerjakan.
4. Kembali ke langkah 1 untuk instruksi selanjutnya.

TABEL 6.2 Langkah-langkah instruksi ADD, NOOP, HALT, dan SKIP

No	ADD	NOOP	HALT	SKIP
1	Ambil instruksi	Ambil instruksi	Ambil instruksi	Ambil instruksi
2	Dekode instruksi	Dekode instruksi	Dekode instruksi	Dekode instruksi
3	Kalkulasi alamat operand jika diperlukan	ke siklus berikutnya	1. Reset flip-flop RUN 2. ke siklus berikutnya	1. Increment PC 2. ke siklus berikutnya
4	Ambil operand	-	-	-
5	Eksekusi operasi	-	-	-
6	1.simpan hasil 2.ke siklus berikutnya	-	-	-

TABEL 6.3 Langkah-langkah instruksi SKIPIPP, BUN, BZ, dan BRAS

No	SKIP positive	BUN	BZ	Branch and save
1	Ambil instruksi	Ambil instruksi	Ambil instruksi	Ambil instruksi
2	Dekode instruksi	Dekode instruksi	Dekode instruksi	Dekode instruksi
3	1. jika tanda zero, increment PC 2. Ke siklus berikutnya	1. Alamat branch disalin ke PC 2. Ke siklus berikutnya	1. Jika akumulator zero, alamat branch disalin ke PC	Simpan PC dalam alamat operand pada memori

No	SKIP positive	BUN	BZ	Branch and save
	berikutnya		2. Ke siklus berikutnya	
4	-	-	-	Load alamat operand ke PC
5	-	-	-	1. Increment PC 2. Ke siklus berikutnya

TABEL 6.4 Langkah-langkah instruksi LDA, STA, dan AND

No	LDA	STA	AND
1	Ambil instruksi	Ambil instruksi	Ambil instruksi
2	Dekode instruksi	Dekode instruksi	Dekode instruksi
3	Ambil operand dari memori	Simpan isi akumulator dalam lokasi memori	Ambil operand
4	1. Load operand dalam akumulator 2. Ke siklus berikutnya	Ke siklus berikutnya	Lakukan operasi (AND)
5	-	-	1. Simpan hasil 2. Ke instruksi berikutnya

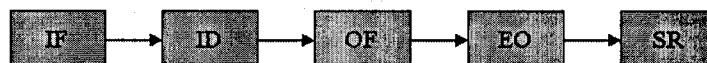
Mode pengalamatan memengaruhi seberapa cepat suatu siklus instruksi selesai. Gambar 6.4 menunjukkan tiga kasus instruksi ADD yang berbeda.

Selain pada siklus instruksi reguler, unit kontrol juga melakukan urutan-urutan tugas khusus tertentu seperti berikut ini:

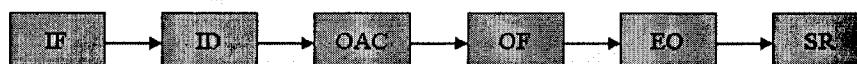
1. Urutan reset pada pengindraan sinyal reset
2. Pengenalan interupsi dan pencabangan ke ISR (*interrupt service routine*).
3. Penanganan situasi abnormal seperti pengenalan kegagalan hardware yang serius dan pengambilan aksi yang tepat seperti *shutdown* atau pengecekan mesin.



(a) Operand-operand telah tersedia pada ALU



(b) Operand pada memori dan alamat operand diberikan dalam instruksi



(c) Alamat operand tidak ditetapkan secara eksplisit

IF : Instruction fetch

ID : Instruction decode

OAC : Operand address calculation

OF : Operand fetch

EO : Execute operation

SR : Store result

Gambar 6.4 Variasi pada siklus instruksi ADD

6.2 URUTAN RESET

Sinyal reset dibangkitkan oleh sirkuit hardware pada situasi berikut:

1. Daya komputer dihidupkan. Hal ini mengaktifkan sirkuit *Power-On Reset* yang membangkitkan sinyal *power-on reset*.
2. Operator menekan tombol/saklar reset pada panel depan yang mengaktifkan sinyal *MANUAL RESET*.
3. Hardware eksternal (di luar prosesor) memberikan sinyal reset ke prosesor. Hal ini adalah suatu keadaan khusus yang ada pada prosesor dahulu yang melepaskan prosesor pada keadaan 'HALT' atau '*shutdown*'.

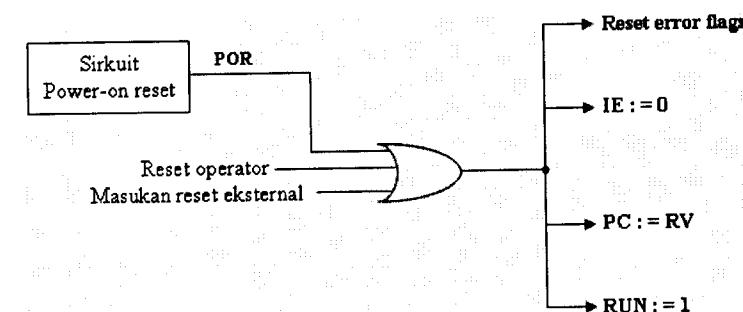
Gambar 6.5 menunjukkan aksi yang dilakukan oleh unit kontrol. Untuk sembarang kasus yang sudah dijelaskan sebelumnya, unit kontrol melakukan empat tindakan:

1. Me-reset *error flag*.
2. Me-reset *Interrupt Enable (IE)* flag; sinyal kontrol $IE := 0$ artinya dibangkitkan untuk maksud tersebut.
3. Mendorong sebuah alamat tetap ke dalam program counter (PC). Alamat ini biasa dikenal dengan '*reset vector*'. Sinyal kontrol $PC := RV$ dibangkitkan supaya alamat vektor reset dimasukkan ke dalam PC. Umumnya vektor reset adalah semua *zero-address* atau semua

one-address yang dapat dibangkitkan dengan mudah oleh hardware. Beberapa prosesor berisi program *Built-In Self-Test* (BIST) yang dieksekusi pertama sebagai suatu ukuran keyakinan agar prosesor yakin dengan keandalan yang dimilikinya. Jika BIST berhasil, maka selanjutnya dia memasukkan vektor reset ke dalam PC.

4. Men-set flip-flop RUN/HALT. Sinyal kontrol $RUN := 1$ mengerjakan hal ini.

Ketika langkah di atas selesai, maka hardware berada di bawah kendali software.



Gambar 6.5 Proses reset

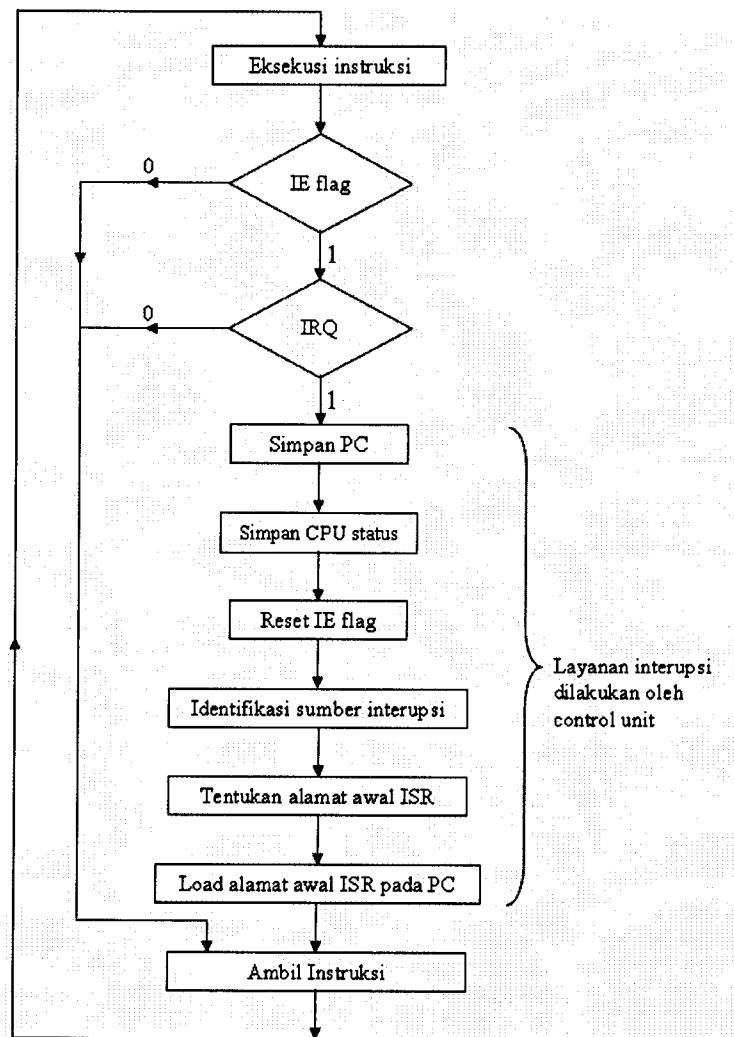
6.3 PENGENALAN DAN PELAYANAN INTERUPSI

Interupsi dapat terjadi setiap saat. Biasanya, unit kontrol memeriksa kehadiran permintaan interupsi sebelum melakukan pengambilan instruksi baru setelah instruksi yang sebelumnya diselesaikan. Jika interrupt enable flag = 0, maka unit kontrol melompati langkah ini dan interupsi tetap ditunda. Dalam mengindra permintaan interupsi, unit kontrol melakukan hal/fungsi berikut:

1. Menyimpan isi PC ke dalam lokasi stack yang alamatnya ditunjukkan oleh penunjuk stack (*Stack Pointer*, SP).
2. Menurunkan nilai (decrement) SP.
3. Menyimpan CPU status flag di dalam lokasi stack yang alamatnya ditunjukkan oleh SP dan menurunkan nilai SP.
4. Menetapkan vektor interupsi (alamat awal ISR).
5. Memasukkan alamat awal ISR ke dalam *program counter* PC

6. Me-reset IE flag supaya interupsi selanjutnya dilumpuhkan (disable)
7. Melanjutkan siklus instruksi.

Sebagai hasil dari aksi tersebut, kontrol men-switch ke ISR. Gambar 6.6 menunjukkan siklus instruksi yang dimodifikasi dan pelayanan interupsi oleh unit kontrol



Gambar 6.6 Siklus instruksi dan pelayanan interupsi

6.4 PENANGANAN SITUASI ABNORMAL

Ketika unit kontrol menghadapi beberapa situasi abnormal yang serius, setelah itu pemrosesan tak dapat berfungsi secara total. Dua kasus tersebut adalah:

1. Prosesor mengindra suatu kondisi pengecualian seperti overflow, kode operasi ilegal dan sebagainya, dan ketika dilakukan tindakan untuk hal tersebut (layanan interupsi), pengecualian yang lain terjadi, yang dihasilkan dalam situasi 'double error'. Pada kasus ini, unit kontrol melakukan 'shutdown' dengan melakukan resetting pada flip-flop RUN/HALT.
2. Ketika prosesor melakukan suatu siklus bus baca/tulis, hardware pendeksi error menemukan error yang disebabkan oleh siklus bus yang sekarang gagal, unit kontrol mengenali hal ini sebagai kondisi 'machine check'. Dilakukan scan diagnostik dengan menyimpan status CPU pada register khusus machine check dan kemudian bercabang ke rutin layanan machine check.

6.5 SIKLUS INSTRUKSI

Karena set instruksi dari suatu CPU memiliki bermacam mode pengalamatan dan format operand, maka unit kontrol bertanggung jawab untuk menjaga semua kemungkinan pada setiap tingkat dalam siklus instruksi.

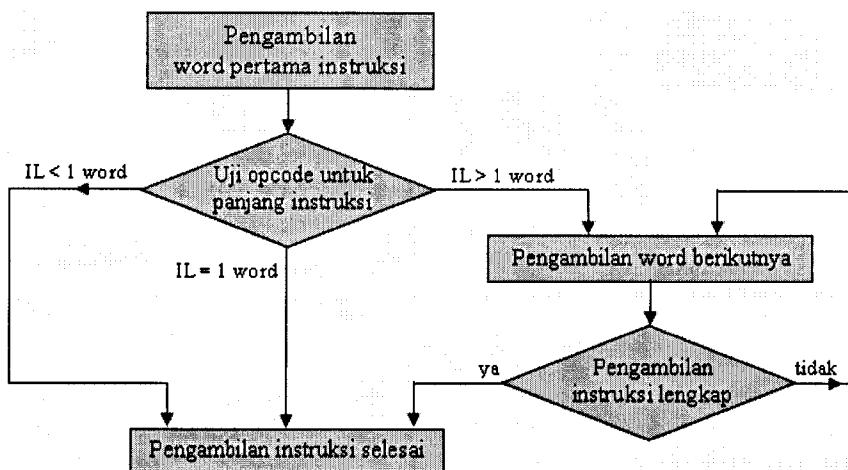
6.5.1 Pengambilan Instruksi

Sub-siklus melibatkan pengambilan instruksi dari alamat memori program yang disediakan dalam program counter. Masalah yang muncul meliputi:

1. Jika panjang-instruksi lebih besar dari panjang word, maka unit kontrol melakukan siklus baca memori lebih dari satu untuk melakukan pengambilan instruksi. Contoh: Panjang instruksi adalah 4 byte dan panjang word memori adalah 2 byte, maka dua word harus diambil/fetch.
2. Jika datapath antara CPU dan memori lebih kecil dari panjang word memori, maka unit kontrol melakukan siklus bus lebih dari satu kali untuk mendapatkan satu word memori. Contoh: Bus

data adalah 16 bit dan panjang word memori 32 bit, dua siklus bus dibutuhkan untuk mengambil satu word.

Bagaimana unit kontrol mengetahui panjang dari instruksi yang diambil? Panjang instruksi diketahui dari opcode instruksi tersebut. Karena itu, unit kontrol mengetahui panjang instruksi hanya setelah pengambilan word pertama instruksi. Berdasarkan opcode, unit kontrol memutuskan jumlah word yang akan diambil dari memori. Gambar 6.7 menunjukkan logika yang diikuti oleh unit kontrol selama pengambilan instruksi.



Gambar 6.7 Pengambilan instruksi multiword

TABEL 6.5 Instruksi dan aksi yang diperlukan pada tingkat eksekusi

No.	Opcode	Aksi pada tingkat eksekusi
1	NOOP	Nihil
2	HALT	Me-reset flip-flop RUN
3	EI	Men-set flip-flop IE
4	DI	Me-reset flip-flop IE
5	INC	Increment isi akumulator
6	CMA	Komplemenkan isi akumulator
7	LDA	1. Baca dari alamat memori yang terdapat dalam instruksi

		2. Masukkan data memori ke akumulator
8	STA	Tulis isi akumulator pada alamat memori yang terdapat dalam instruksi
9	SKIP	Increment PC (program counter)
10	BUN atau JUMP	Masukkan 'alamat branch' (diberikan oleh instruksi) dalam PC
11	BSA (branch and save return address)	1. Simpan isi PC pada alamat yang ditunjukkan oleh instruksi 2. Masukkan alamat pada PC 3. Increment PC
12	ISZ (increment and skip if zero)	1. Baca dari alamat memori yang diberikan dalam instruksi 2. Naikkan 1 (Increment 1) data memori (dalam bentuk komplemen-2) 3. Simpan hasil pada alamat memori 4. Jika hasil nol, increment PC
13	IN	Melakukan satu siklus bus baca I/O
14	OUT	Melakukan satu siklus bus tulis I/O
15	INT	1. Simpan PC dan CPU flag dalam stack 2. Baca interrupt vector 3. Baca dari alamat memori yang diberikan oleh interrupt vector 4. Masukkan data memori dalam PC
16	IRET	Baca dari stack dan simpan dalam PC dan CPU flag
17	CIR,sirkulasi kanan	Geser isi akumulator ke kanan. Flip-flop E harus disalin ke MSB
18	CIL,sirkulasi kiri	Geser isi akumulator ke kiri. Flip-flop E harus disalin ke LSB
19	AND	Lakukan operasi logika AND pada operand
20	ADD	Lakukan penjumlahan pada operand
21	EXECUTE	1. Baca dari alamat memori yang diberikan oleh instruksi 2. Masukkan data dalam IR 3. Lanjutkan fase eksekusi
22	JZ	Jika akumulator nol, masukkan alamat branch dalam PC

Pada Gambar 6.8 ditunjukkan logika yang umum diikuti oleh unit kontrol untuk beberapa instruksi.

6.5.2 Kalkulasi Alamat Operand

Unit kontrol mengetahui lokasi tepat operand apakah pada opcode atau pada medan mode pengalamatan dalam instruksi. Ada empat lokasi operand yang umum:

1. Pada instruksi itu sendiri sebagai immediate operand
2. Pada register prosesor
3. Pada memori utama
4. Pada port I/O

Alamat operand dinyatakan dalam instruksi pada salah satu mode pengalamatan seperti pengalamatan langsung, pengalamatan tak langsung, pengalamatan relatif, pengalamatan indeks, dan lain-lain. Prosedur kalkulasi alamat operand bergantung pada mode pengalamatannya masing-masing.

6.5.3 Pengambilan Operand

Jika operand berada dalam memori utama, CPU melakukan siklus baca memori. Jumlah siklus baca memori utama bergantung pada dua faktor yaitu panjang operand dan panjang word memori. Unit kontrol harus menjaga siklus memori. Jika operand berada dalam register prosesor, dia dibaca dengan cepat dibandingkan dengan operand dari memori utama.

6.5.4 Operasi Eksekusi

Unit kontrol didesain untuk melaksanakan semua mikro-operasi seperti yang ditetapkan oleh set instruksi. Unit kontrol mengetahui dari opcode urutan aksi yang tepat (mikro-operasi) yang harus dilakukan selama tingkatan ini. Hal ini bervariasi dari tanpa mikro-operasi sampai mikro-operasi jamak. Untuk beberapa instruksi, mikro-operasi merupakan aksi internal yang terjadi dalam prosesor. Sedangkan yang lainnya, mikro-operasi dilaksanakan dalam unit-unit eksternal seperti memori, pengontrol I/O, dan lain-lain. Tabel 6.5 memberikan aksi yang diperlukan dalam tingkat eksekusi pada beberapa instruksi.

6.5.5 Pemilihan Desain Unit kontrol

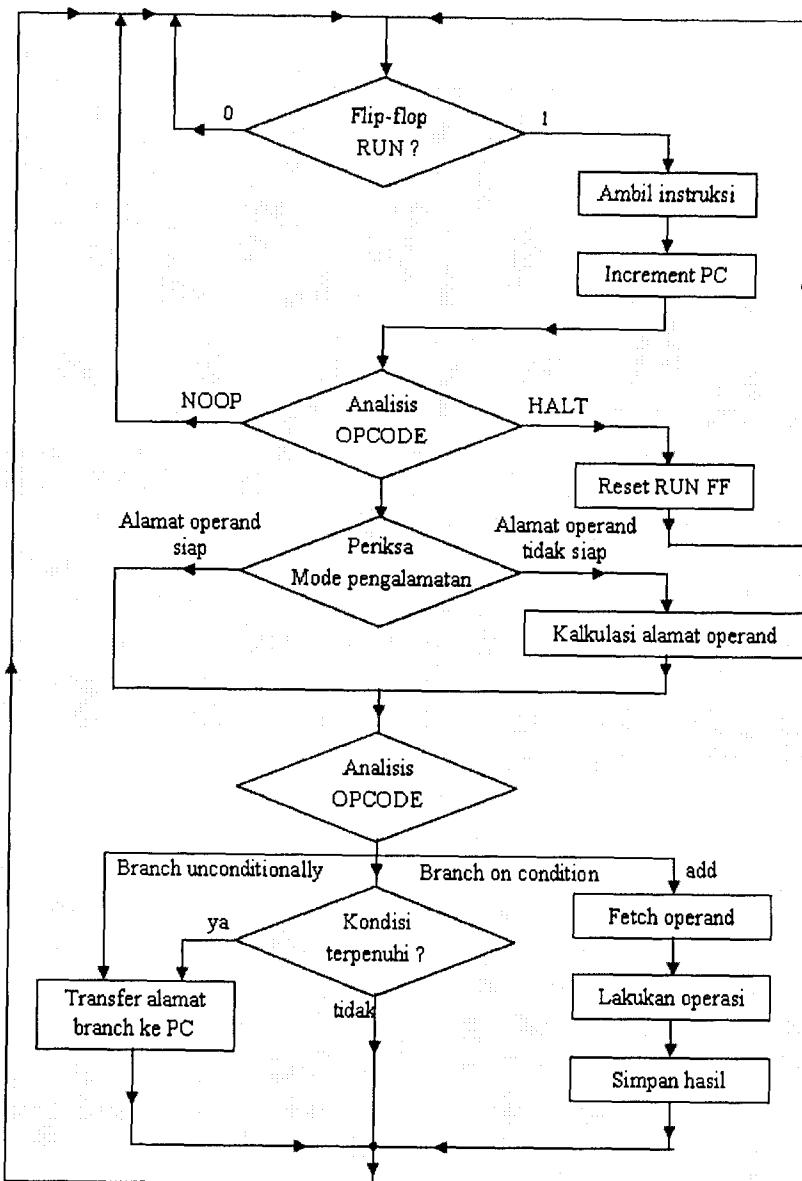
Unit kontrol didesain dengan dua macam kemampuan berbeda:

1. Kumpulan sinyal kontrol adalah unik bagi setiap instruksi. Kepintaran/intelijensi ini ‘dibangun’ atau ‘disimpan’ di dalam unit kontrol. Unit kontrol mensuplai (melalui pembangkitan atau pembacaan) sekumpulan sinyal kontrol yang sesuai dengan opcode.
2. Sinyal kontrol suatu instruksi disuplai dengan urutan yang tepat. Informasi ini juga merupakan kepintaran yang disebutkan pada poin 1 di atas.

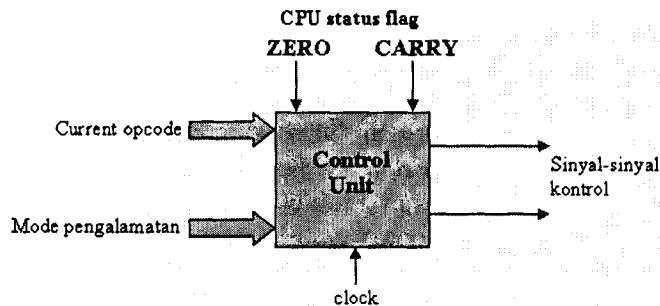
Pada dasarnya unit kontrol mengubah setiap instruksi bahasa mesin menjadi serangkaian sinyal kontrol yang mengaktifkan titik-titik kontrol dalam datapath. Gambar 6.9 menunjukkan input dan output pada unit kontrol.

Ada dua cara untuk mendesain unit kontrol yaitu *hardwired control unit* (HCU) dan *microprogrammed control unit* (MCU). Pada HCU, sirkuit-sirkuit digital membangkitkan sinyal kontrol sedangkan pada MCU, sinyal-sinyal kontrol disimpan sebagai pola-pola bit dalam sebuah ROM.

HCU merupakan teknik desain yang konvensional, sedangkan MCU merupakan teknik modern.



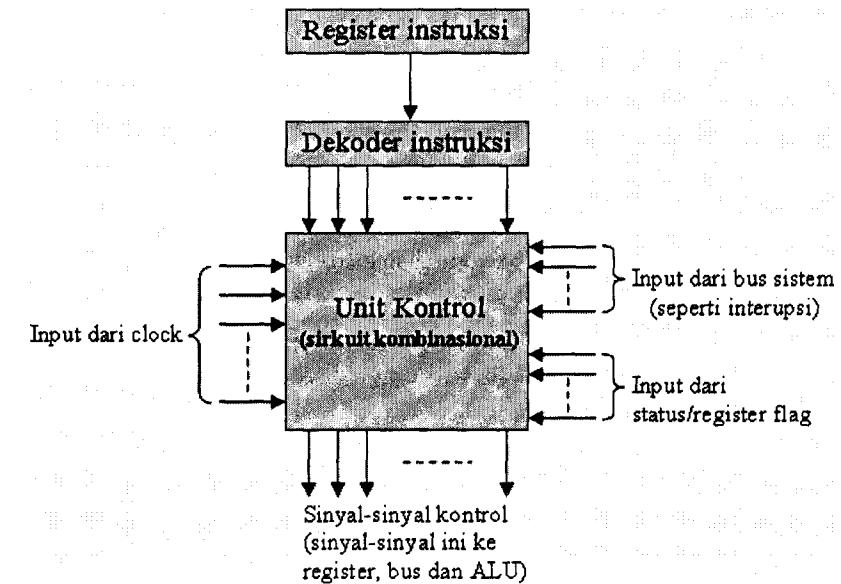
Gambar 6.8 Pengambilan keputusan secara keseluruhan dalam unit kontrol



Gambar 6.9 Masukan dan keluaran unit kontrol

6.6 HARDWIRED CONTROL UNIT

Hardwired Control Unit (HCU) terdiri atas kumpulan sirkuit kombinasional yang membangkitkan sinyal kontrol (Gambar 6.10 menunjukkan diagram blok HCU). Prinsip kerja HCU ditunjukkan pada Gambar 6.11. Setiap baris vertical merepresentasikan suatu instruksi. Setiap baris horizontal merepresentasikan suatu rangkaian pulsa perwaktuan.

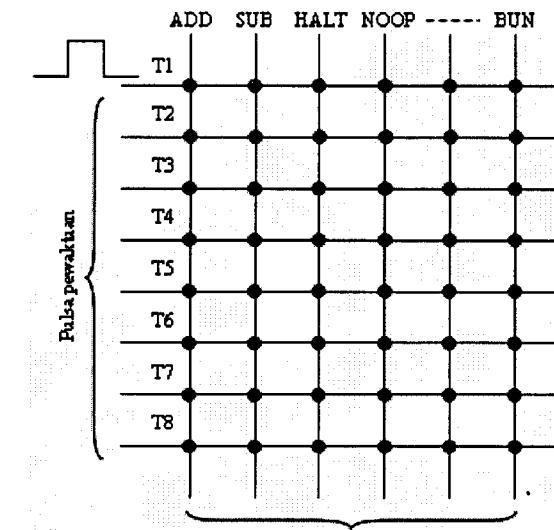


Gambar 6.10 Diagram blok HCU

Pulsa-pulsa pewaktuan T1, T2, T3, dan lain-lain mempunyai waktu tunda tertentu. Mereka digunakan sebagai sinyal referensi pewaktuan. Pada suatu instruksi, mikrooperasi-mikrooperasi yang diperlukan adalah urutan waktu yang menggunakan pulsa-pulsa pewaktuan. Misalnya untuk instruksi LDA jika dibaca dari memori, dilakukan oleh T1, penyimpanan pada memori dapat dilakukan pada T2. Kombinasi sinyal LDA dan T1 dapat digunakan sebagai sinyal kontrol untuk pembacaan dari memori. Sinyal kombinasi lainnya dari LDA dan T2 dapat digunakan sebagai sinyal kontrol untuk penyimpanan pada akumulator. Gambar 6.12 menunjukkan bagaimana dua gerbang membangkitkan dua sinyal kontrol selama eksekusi instruksi LDA. Konsep ini digunakan untuk pembangkitan semua sinyal kontrol. Untuk pembangkitan beberapa sinyal kontrol, digunakan status dari flag CPU. Misalnya, untuk instruksi JZ (lompat jika akumulator nol), zero flag digunakan bersama T1 dan JZ seperti yang ditunjukkan pada Gambar 6.13. Sinyal kontrol dibangkitkan jika zero flag = 1.

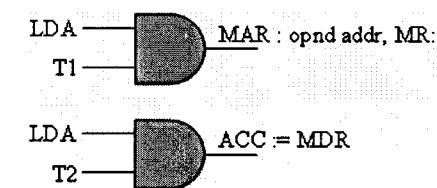
HCU mempunyai sirkuit untuk:

1. Mengidentifikasi instruksi dari opcode dan membangkitkan sinyal relevan seperti ADD, NOOP, HALT, dan lain-lain. Pada suatu waktu tertentu, ada salah satu sinyal dari opcode yang aktif dan yang lainnya tidak aktif.
2. Mempertahankan/menjaga referensi waktu dengan membangkitkan serangkaian pulsa-pulsa pewaktuan.
3. Membangkitkan sekumpulan sinyal-sinyal kontrol sesuai mikrooperasi yang akan dikerjakan untuk instruksi yang ada (saat itu) pada interval waktu yang tepat.
4. Setelah penyelesaian eksekusi dari satu instruksi, perlu pembangkitan sinyal kontrol untuk pengambilan instruksi berikutnya.
5. Penanganan special sequence seperti reset sequence, interrupt sequence, dan penanganan keadaan abnormal.

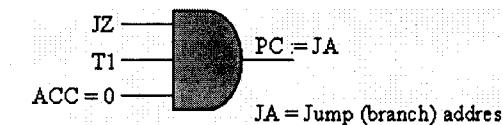


Setiap titik merepresentasikan suatu rangkaian yang membangkitkan satu atau lebih sinyal kontrol

Gambar 6.11 Prinsip dasar HCU

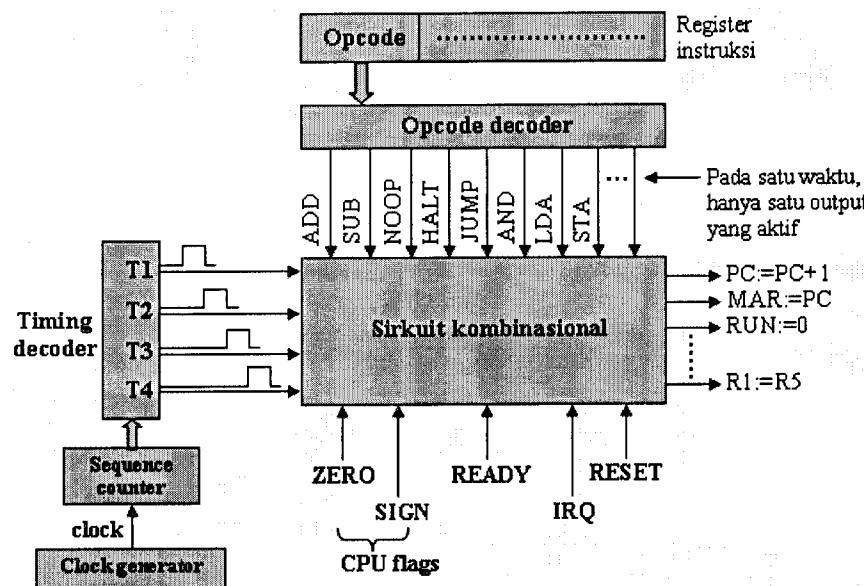


Gambar 6.12 Pembangkitan sinyal kontrol untuk instruksi LDA

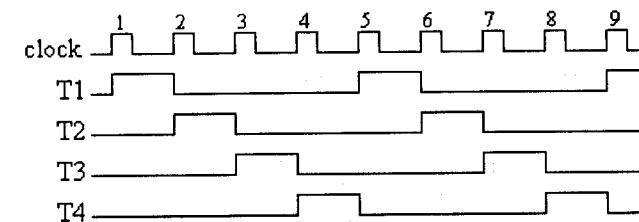


Gambar 6.13 Pembangkitan sinyal kontrol untuk instruksi JZ

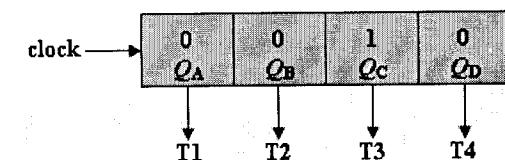
Pada Gambar 6.14 diberikan diagram blok detail HCU. Opcode merupakan masukan utama pada unit kontrol ke opcode decoder. Decoder adalah suatu sirkuit kombinasional dengan n masukan dan 2^n keluaran. Pada satu waktu tertentu, hanya ada satu keluaran yang aktif. Keluaran yang aktif menentukan kombinasi sinyal-sinyal kontrol. Opcode decoder menganalisis pola opcode dan membangkitkan satu sinyal keluaran yang berhubungan dengan opcode saat itu. Jika instruksi yang dibaca adalah ADD, maka keluaran ADD dari opcode decoder yang aktif dan yang lainnya tidak aktif. Sinyal ADD menuju ke beberapa gerbang dalam sirkuit kombinasional. Selama komputer dalam keadaan hidup (power-on), clock dibangkitkan secara kontinu apa pun yang terjadi dalam komputer. Bahkan bila CPU dalam keadaan halt, sinyal clock tetap dibangkitkan.



Gambar 6.14 Organisasi HCU



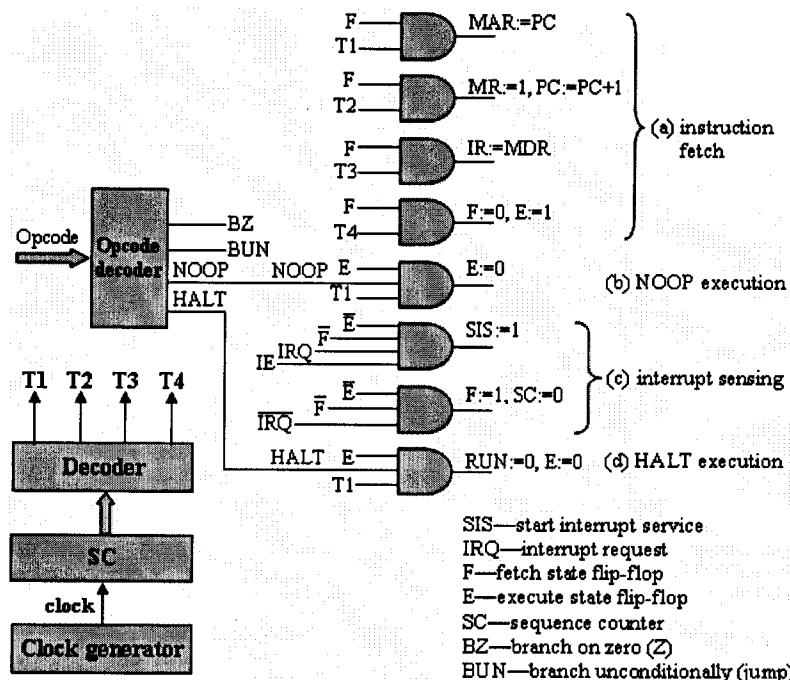
Gambar 6.15 Pembangkitan sinyal pewaktuan



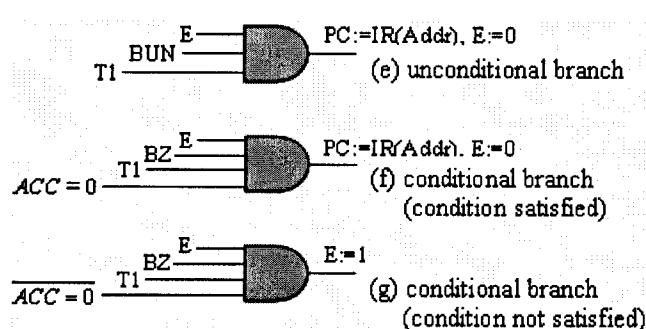
Gambar 6.16 Pencacah lingkar

Pencacah urut pada dasarnya adalah sebuah pencacah naik. Isinya dinaikkan (*increment*) pada setiap sinyal clock. Ketika cacahan mencapai nilai akhir, dia di-reset ke nol oleh clock berikutnya. Dengan pencacah urut dua bit, berarti status cacahan adalah 00, 01, 10, 11, 00 dan seterusnya. Keluaran pencacah urut diterjemahkan sandinya oleh *timing decoder*. Gambar 6.15 menunjukkan bentuk gelombang keluaran timing decoder. Setiap sinyal pewaktuan (T1, T2, T3 dan T4) adalah sebuah bentuk gelombang periodik karena bentuknya yang repetitif. Dimungkinkan menggunakan sebuah pencacah lingkar untuk menggantikan pencacah urut dan timing decoder. Sebuah pencacah lingkar (Gambar 6.16) mempunyai perotasi '1' yang bergerak dari satu posisi bit ke posisi berikutnya bila sinyal clock datang. Gambar 6.17(a)-(g) menunjukkan beberapa contoh kasus sirkuit kombinasional yang dibutuhkan untuk pembangkitan berbagai sinyal kontrol. Flip-flop F dan E menunjukkan apakah CPU berada dalam fase pengambilan instruksi atau fase eksekusi instruksi dari suatu siklus instruksi. Jika keduanya reset, artinya prosesor tidak bekerja (siklus instruksi tidak dilaksanakan). Sebuah analisis hati-hati dari Gambar 6.17 menunjukkan bahwa gerbang tertentu dapat dieliminasi dengan menggabungkan sirkuit-sirkuit dari instruksi berbeda.

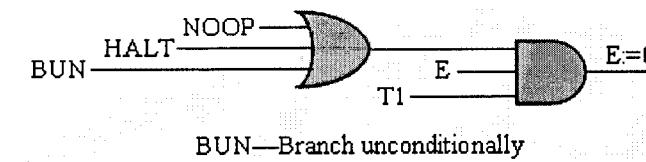
Pada Gambar 6.18 ditunjukkan sirkuit untuk kontrol program counter (PC).



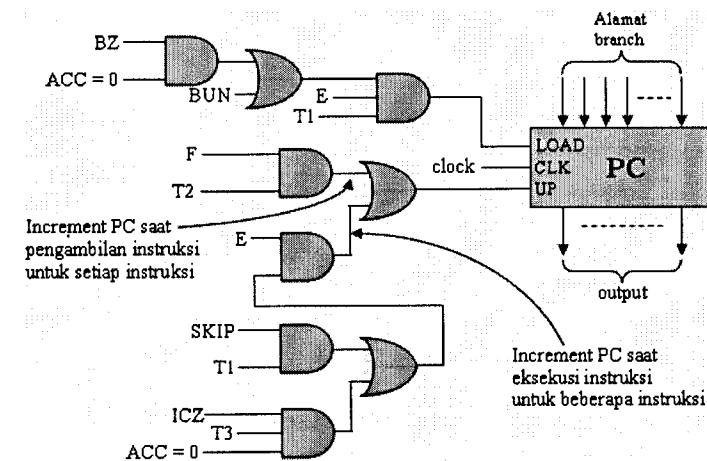
Gambar 6.17(a) HCU: pembangkitan sinyal-sinyal kontrol



Gambar 6.17(b) HCU: pembangkitan sinyal-sinyal kontrol



Gambar 6.17(c) Penyederhanaan gerbang untuk pembangkitan sinyal kontrol



Gambar 6.18 Kontrol PC (vektor reset tidak diperlihatkan)

6.6.1 Keuntungan Hardwired Control Unit

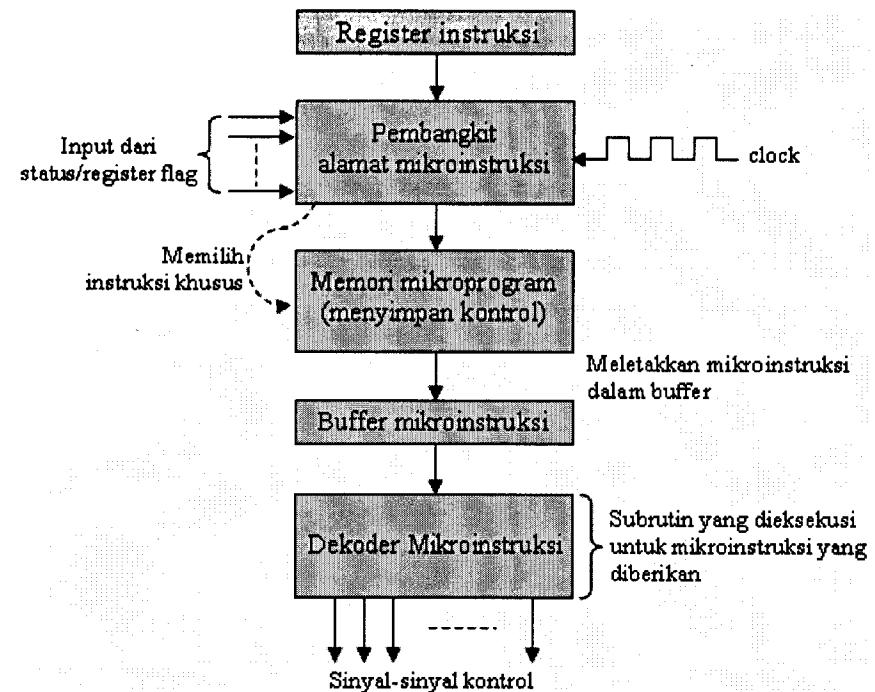
Keuntungan HCU bekerja dengan cepat daripada MCU. Sirkuit kombinasional membangkitkan sinyal kontrol berdasarkan status sinyal-sinyal input. Segara setelah kombinasi sinyal input yang diperlukan ditempatkan, dengan segera output (sinyal kontrol) dibangkitkan oleh sebuah sirkuit kombinasional. Waktu tunda antara pembangkitan output ke input yang tersedia bergantung pada banyaknya gerbang dalam lintasan dan waktu tunda propagasi setiap gerbang dalam lintasan. Jika terdapat dua level gerbang dan setiap gerbang mempunyai tunda propagasi 5 ns, waktu yang digunakan untuk pembangkitan sinyal kontrol adalah 10 ns.

6.6.2 Kekurangan Hardwired Control Unit

1. Jika CPU mempunyai titik-titik kontrol yang banyak, desain unit kontrol menjadi sangat kompleks.
2. Desain tidak fleksibel. Jika dibutuhkan modifikasi, maka sangat sulit untuk melakukan koreksi. Modifikasi desain diperlukan pada situasi berikut:
 - (a) Jika terdapat kesalahan dalam desain awal
 - (b) Jika fitur baru (misalnya instruksi baru) akan ditambahkan pada desain yang ada.
 - (c) Komponen hardware baru (misalnya memori) kecepatan yang lebih tinggi tersedia, yang akan meningkatkan performa.

6.7 MICROPROGRAMMED CONTROL UNIT

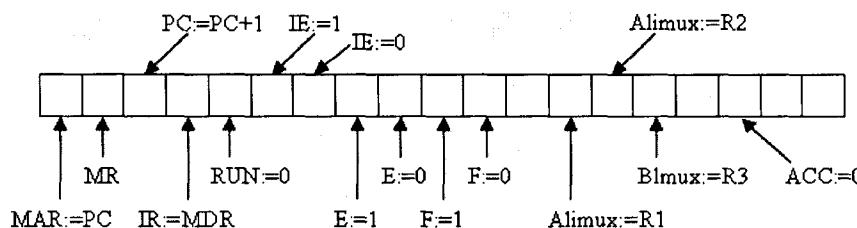
Microprogramming adalah sebuah konsep modern yang digunakan untuk perancangan unit kontrol (Gambar 6.19). Beberapa aplikasi umum adalah sebagai pengontrol I/O seperti disk controller, peripheral devices seperti printer dan hard disk drive. Filosofi microprogramming berdasarkan pada elaborasi “pembacaan pola kontrol tersimpan”.



Gambar 6.19 Diagram blok MCU

Mikrooperasi dieksekusi bila sinyal kontrol yang sesuai dibuat aktif. Setiap instruksi memerlukan suatu urutan set mikrooperasi yang spesifik. Sinyal-sinyal kontrol diperlukan untuk suatu instruksi dan urutan waktu yang dapat disimpan dalam memori yang disebut memori kontrol. Dengan pengambilan word memori kontrol satu demi satu, sinyal-sinyal kontrol dapat dibangkitkan. Memori kontrol adalah sebuah ROM. Dengan cara yang sama, aktivitas yang biasa terjadi dalam unit kontrol seperti penanganan interupsi, pengambilan instruksi, dan sebagainya dapat diterjemahkan ke dalam word memori kontrol. Unit kontrol hanya dapat dibaca dari memori kontrol tetapi isinya tidak dapat dimodifikasi. Dalam bentuk sederhana, setiap bit dalam word memori kontrol dapat dicadangkan untuk sinyal kontrol; jadi, pemindahan bit untuk semua sinyal kontrol dapat digunakan. Jika bit adalah 1, sinyal kontrol yang sesuai menjadi aktif. Jika bit adalah 0, sinyal kontrol tidak aktif. Bila suatu word dibaca/diambil dari memori kontrol, beberapa bit dapat menjadi 1 dan yang lain 0. Sinyal-sinyal kontrol

yang berkenaan dengan '1' dibangkitkan. Jadi mikrooperasi jamak dieksekusi secara simultan. Gambar 6.20 menunjukkan sebuah contoh format word memori kontrol. Setiap word memori kontrol dikenal dengan mikrooperasi.



Gambar 6.20 Konfigurasi word memori kontrol sederhana

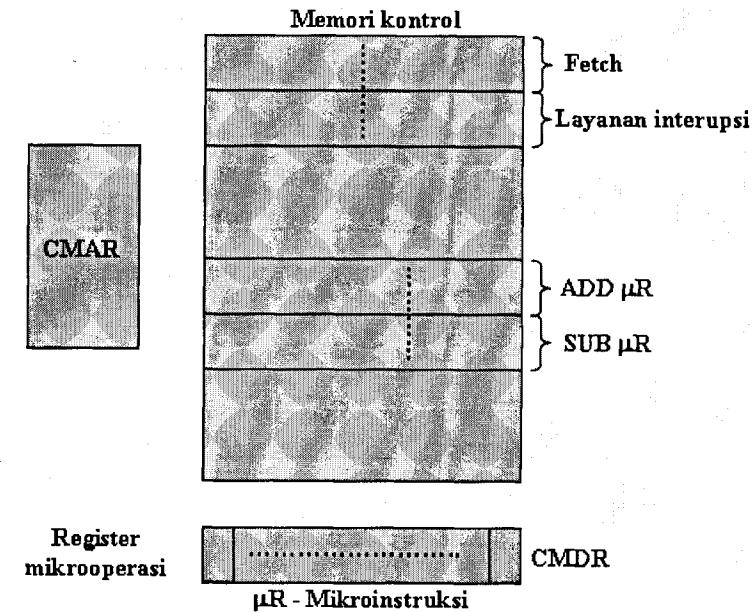
6.7.1 Permasalahan pada Microprogrammed Control Unit

Gambar 6.21 menunjukkan persyaratan dasar MCU. Terdapat *Control Memory Address Register* (CMAR) dikenal juga dengan *microprogram counter* dan CMDR (*control memory data register*) yang dikenal juga dengan register mikroinstruksi. Masalah berikut harus dipecahkan dalam suatu MCU:

1. Terdapat eksekusi mikroprogram yang terpisah pada setiap instruksi seperti ADD, SUB, LDA dan sebagainya. Setelah pengambilan instruksi mikroprogram berakhir, bagaimana sebenarnya eksekusi mikroprogram mendapatkan pengontrolan?
2. Setelah eksekusi mikroprogram selesai, bagaimana kontrol beralih mengambil mikroprogram?
3. Setelah power on, bagaimana unit kontrol memulai mikroprogram yang tepat?
4. Sering, mikroinstruksi tertentu harus dilewati (skip) dan unit kontrol harus bercabang pada basis status tertentu dalam CPU. Misalnya, status carry flag atau zero flag menentukan aksi yang tepat yang diperlukan setelah penyelesaian suatu operasi aritmetika. Dengan cara yang sama, status IE flag memutuskan

lintasan aksi yang akan diikuti dengan pengindraan permintaan interupsi. Karena itu kita memerlukan suatu mekanisme untuk pencabangan bersyarat.

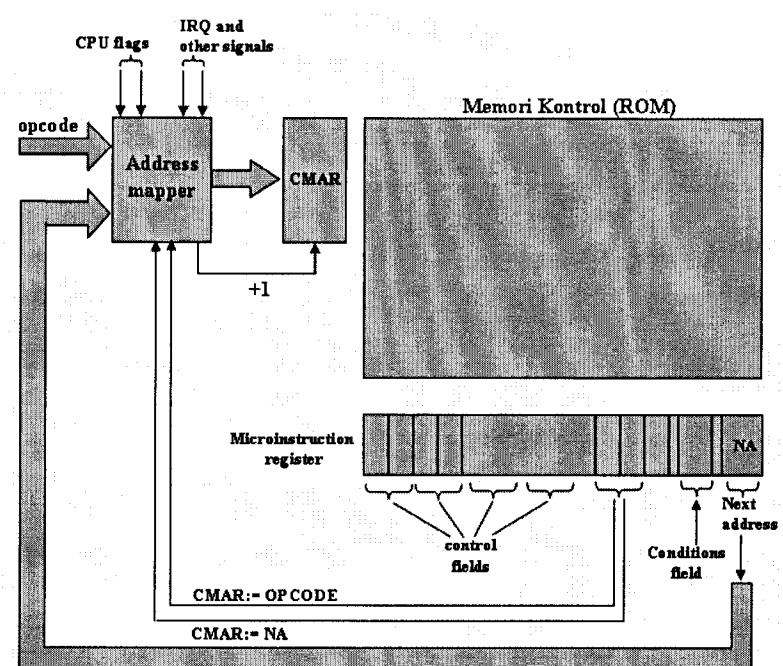
5. Penetapan sebuah bit pada setiap sinyal kontrol menimbulkan word memori kontrol yang lebar. Karena itu kita memerlukan teknik *device* untuk mengurangi panjang mikroinstruksi.



Gambar 6.21 Mikroprogram dan memori kontrol

6. Banyak mikroinstruksi dari instruksi tertentu yang sama. Misalnya, aksi yang diperlukan untuk instruksi ADDB dan SUBB hanya berbeda pada satu aspek. Instruksi ADD menggunakan B sedangkan instruksi SUB menggunakan komplemen B. Penggunaan secara keseluruhan eksekusi mikroprogram yang berbeda, menambah ukuran (jumlah lokasi) memori kontrol. Kita memerlukan beberapa teknik untuk mengombinasikan mikroinstruksi tersebut. Instruksi ADD dan SUB dapat mempunyai eksekusi mikroprogram yang sama dan akan menangani dua persyaratan ini oleh sebuah mikrorutin tunggal untuk mengurangi kapasitas memori kontrol.

Gambar 6.22(a) menunjukkan organisasi khas sebuah MCU. Gambar 6.22(b) menunjukkan pembangkitan alamat mikroinstruksi berikutnya.



Gambar 6.22(a) Operasi MCU

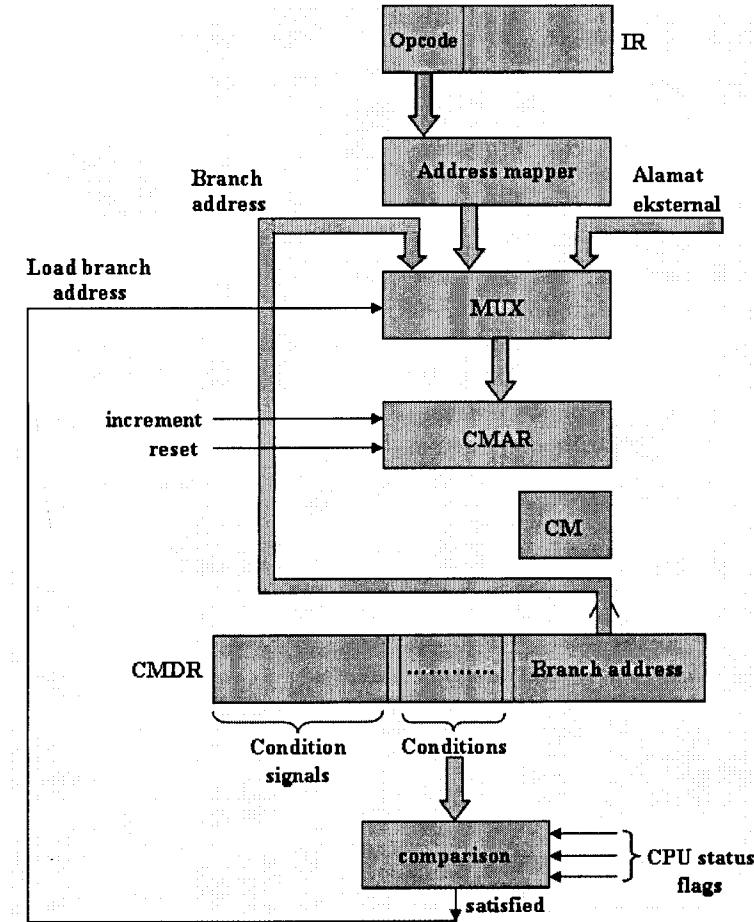
6.7.2 Pengurangan Panjang Word Mikroinstruksi

Mikrooperasi-mikrooperasi tertentu (sinyal-sinyal kontrolnya) saling eksklusif satu sama lain. Misalnya, jika kita memikirkan mikrooperasi berikut yang berhubungan dengan program counter (PC), kita akan melihat bahwa dalam suatu waktu, hanya salah satu yang dieksekusi bergantung pada kondisi siklus instruksi.

1. $PC := PC + 1$
2. $PC := MDR$
3. $PC := EA/IR$ (instruksi jump)

Sebagai pengganti penetapan 4 bit dalam word mikro-instruksi, kita dapat mempunyai 2 yang memberikan informasi yang dikodekan. Hasil ini

menghemat 2 bit. Pada waktu yang sama, dekoder 2-4 bit digunakan untuk mendekode field 2-bit dan memberikan 4 keluaran yang sesuai dengan 4 sinyal kontrol. Kita mengurangi panjang word kontrol, pada biaya penambahan sebuah sirkuit dekoder.

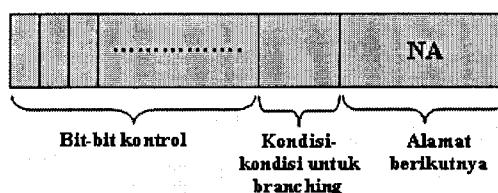


Gambar 6.22(b) Sequence logic

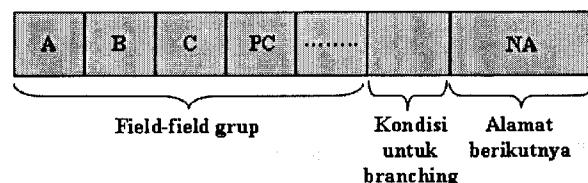
6.7.3 Horizontal dan Vertical Microprogramming

Gambar 6.23, 6.24, dan 6.25 menunjukkan tiga cara pengindikasian pola sinyal kontrol. Bit individual pada masing-masing sinyal kontrol pada

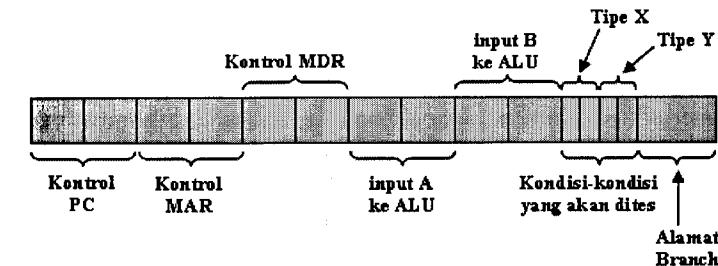
format mikroinstruksi dikenal dengan mikro-operasi horizontal seperti yang ditunjukkan pada Gambar 6.23. Setiap bit mengaktifkan satu sinyal kontrol. Beberapa sinyal kontrol dapat dibangkitkan secara simultan oleh sebuah mikroinstruksi tunggal. Panjang mikroinstruksi meningkat (30-120) bit. Pada Gambar 6.24, setiap grup menerjemahkan sinyal-sinyal kontrol (mikro-operasi) secara *mutual exclusive*. Hal ini mengurangi panjang mikroinstruksi. Tetapi, sebuah sirkuit dekoder diperlukan untuk setiap field untuk mendekode pola dan menetapkan sinyal kontrol. Jumlah total sinyal kontrol simultan (karenanya mikrooperasi) tidak bisa melebihi jumlah field. Sebagai hasilnya, jumlah mikroinstruksi yang dibutuhkan dalam sebuah rutin lebih besar daripada mikroprogram horizontal murni. Hal ini menambah delay karena jumlah mikroinstruksi yang lebih besar diambil dari memori kontrol. Pada mikroinstruksi vertikal (Gambar 6.25), field tunggal dapat menghasilkan sebuah urutan ter-encode. Teknik mikroprogram vertikal menggunakan waktu yang lebih untuk pembangkitan sinyal-sinyal kontrol karena diperlukan waktu pendekodean dan juga mikroinstruksi yang lebih. Tetapi, biaya keseluruhan lebih kecil karena ukuran mikroinstruksi kecil (12-30 bit). Mikroprogram horizontal mengeluarkan sinyal-sinyal kontrol lebih cepat, tetapi ukuran memori kontrolnya sangat besar karena panjang word meningkat.



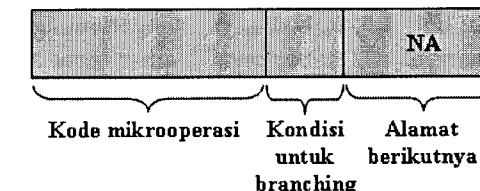
Gambar 6.23 Mikroinstruksi horizontal murni



Gambar 6.24(a) Mikroinstruksi horizontal dengan multigrup



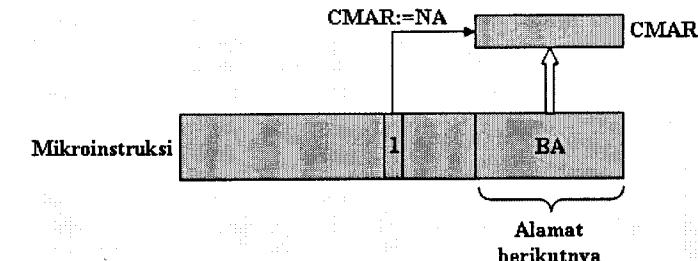
Gambar 6.24(b) Bit-bit kontrol grup pada field-field



Gambar 6.25 Mikrooperasi vertikal murni

6.7.4 Pencabangan Tidak Bersyarat

Mikroinstruksi memberikan alamat mikroinstruksi berikutnya. Salah satu dari bit-bit kontrol membangkitkan sinyal kontrol CMAR:=NA. Hal ini diilustrasikan pada Gambar 6.26, menunjukkan tiga cara pengindikasian pola sinyal.



Gambar 6.26 Pola pencabangan tak bersyarat

6.7.5 Pencabangan Bersyarat

Setiap bit pada mikroinstruksi horizontal berhubungan dengan sebuah sinyal kontrol. Field kondisi pencabangan menetapkan kondisi yang akan diuji oleh mikroinstruksi yang sekarang. Beberapa kondisi yang biasa diuji antara lain:

1. *Zero flag*
2. *Overflow flag*
3. *Carry flag*
4. *Interrupt enable flag*
5. *Sign flag*

Jika pengujian kondisi berhasil, alamat pencabangan mengisi mikroinstruksi berikutnya untuk dieksekusi, sebaliknya alamat fisik berikutnya yang mengisi mikroinstruksi.

6.7.6 Hardware, Software, dan Firmware

Fungsi berikut adalah tugas yang dikerjakan oleh sirkuit atau mikroprogram dalam CPU:

1. Menganalisa dan menginterpretasi opcode
2. Mengeksekusi mikrooperasi pada berbagai opcode
3. Pengurutan melalui mikrooperasi
4. Mengindera status flag dan mengambil keputusan berdasarkan status tersebut

Firmware merujuk pada mikroprogram. Hardware merupakan hal yang kaku karena untuk memodifikasinya sangat sulit. Software merujuk pada program (bahasa mesin/bahasa tingkat tinggi). Hal ini dapat dimodifikasi dengan mudah. Tetapi mikroprogram terletak antara hardware dan software. Tidak kaku seperti hardware dan karena itu dapat dimodifikasi. Karena itu, disebut firmware. Definisi asal firmware merujuk ke mikroprogram yang disimpan dalam memori kontrol yaitu pada ROM. Definisi ini telah digunakan sejak lama dan kini setiap software yang disimpan dalam ROM diistilahkan firmware.

6.7.7 Keuntungan Microprogramming

1. Desain MCU tidak kompleks dibandingkan dengan HCU.
2. Mikroprogram lebih fleksibel karena mudah memodifikasinya
3. Set instruksi dapat dimodifikasi dengan mudah dengan mengubah mikroprogram tanpa memengaruhi datapath.
4. Debugging dan maintenance lebih mudah.

6.7.8 Kekurangan Microprogramming

1. MCU lambat.
2. Untuk CPU kecil dengan sumber daya hardware yang sangat terbatas, MCU mahal dibandingkan dengan HCU.

RINGKASAN

Unit Kontrol merupakan ‘otak’ atau pusat syaraf hardware komputer yang mengawasi pelaksanaan siklus instruksi dan membangkitkan sinyal-sinyal kontrol relevan pada saat yang tepat supaya mikro-operasi yang tepat dapat dikerjakan pada CPU dan unit-unit eksternal lainnya seperti memori dan pengontrol I/O/perangkat.

Unit kontrol dirancang untuk melakukan fungsi-fungsi berikut:

- Reset
- Pengambilan instruksi
- Eksekusi instruksi
- Penanganan interupsi
- Mengontrol bus
- Penanganan situasi abnormal

Pengambilan instruksi merupakan pekerjaan yang dilakukan pada semua jenis instruksi dan tindakan yang diambil oleh unit kontrol sama untuk semua instruksi. Pengambilan operand dan eksekusi instruksi berbeda pada berbagai opcode. Unit kontrol menganalisis pola opcode dan mode pengalamatan dan tentunya melakukan tindakan/aksi yang sesuai (membangkitkan sinyal kontrol yang relevan).

Unit kontrol sinkron menetapkan slot waktu yang berbeda untuk berbagai mikrooperasi yang dikerjakan oleh komponen-komponen hardware logic. Dia tidak menerima umpan balik dari unit logika mengenai penyelesaian mikrooperasi-mikrooperasi. Semua unit melakukan sinkronisasi sendiri dengan sinyal clock yang diberikan oleh unit kontrol untuk informasi waktuan. Unit kontrol asinkron menunggu umpan balik/pengakuan dari unit logika individu masing-masing mikrooperasi. Unit kontrol asinkron dapat beroperasi lebih cepat daripada unit kontrol sinkron. Tetapi, unit kontrol sinkron dapat dirancang lebih mudah dan debugging kegagalan hardware dalam komputer lebih mudah.

Unit kontrol komputer pertama menggunakan jenis yang disebut hardwired control unit atau random logic. Unit kontrol ini membangkitkan sinyal-sinyal kontrol melalui sirkuit-sirkuit hardware. Unit kontrol jenis ini bekerja lebih cepat, tetapi rancangannya menjadi kompleks untuk sebuah prosesor besar.

Unit kontrol jenis yang kedua adalah microprogrammed control unit yang merupakan unit kontrol yang menggunakan konsep “penyimpanan mikroprogram” untuk pembangkitan sinyal-sinyal kontrol. Dia mempunyai sebuah memori ROM di dalam unit kontrol untuk penyimpanan pola-pola sinyal kontrol. Sinyal-sinyal kontrol dan urutan untuk setiap eksekusi instruksi dan aksi-aksi lainnya disimpan sebagai “mikroinstruksi”. Microprogrammed control unit lebih lambat karena harus dilakukan operasi pengambilan mikroinstruksi, tetapi rancangannya lebih mudah dan dapat dimodifikasi.

Mikroprogram horizontal menghasilkan eksekusi yang cepat tetapi menempati ruang yang lebih besar di dalam memori kontrol, sedangkan mikroprogram vertikal mengurangi bianya dengan mengurangi ruang memori kontrol tetapi lebih lambat.

SOAL-SOAL ULANGAN

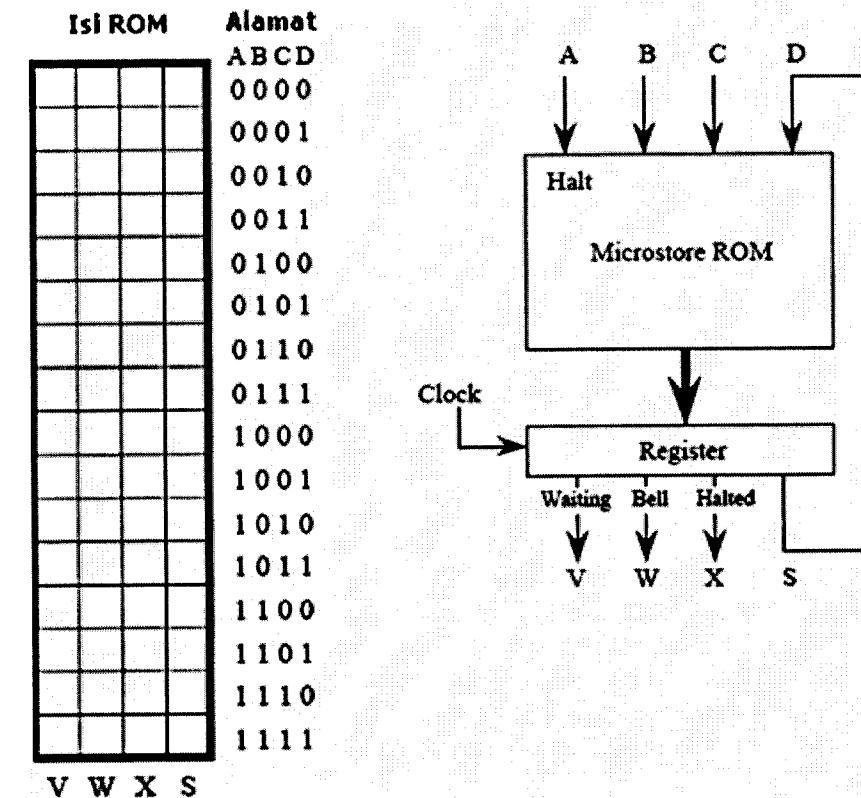
1. Unit yang mengawasi pelaksanaan siklus instruksi dan membangkitkan sinyal-sinyal kontrol relevan pada saat yang tepat supaya mikro-operasi yang tepat dapat dikerjakan pada CPU dan unit-unit eksternal lainnya seperti memori dan I/O controller/devices disebut _____.
2. Memori utama dikontrol oleh dua sinyal kontrol yaitu: (1) _____, dan (2) _____.
3. Prosesor mengeksekusi program dengan melakukan siklus-siklus instruksi yang secara garis besar siklus instruksi ini terdiri atas dua yaitu: (1) _____ dan (2) _____.
4. Siklus instruksi dapat terdiri atas enam urutan langkah yaitu (1) *instruction fetch* (IF), (2) *instruction decode* (ID), (3) *operand address calculation* (OAC), (4) *operand fetch* (OF), (5) _____, dan (6) _____.
5. Urutan langkah-langkah siklus instruksi jika operand-operand telah tersedia di ALU ada empat yaitu: (1) *instruction fetch* (IF), (2) *instruction decode* (ID), (3) _____, dan (4) *store result* (SR).
6. Urutan langkah-langkah siklus instruksi jika operand pada memori dan alamat operand diberikan dalam instruksi ada lima yaitu: (1) *instruction fetch* (IF), (2) *instruction decode* (ID), (3) _____, (4) *execute operation* (EO), dan (5) *store result* (SR).
7. Urutan langkah-langkah siklus instruksi jika alamat operand tidak ditetapkan secara eksplisit dalam instruksi ada enam yaitu: (1)

- instruction fetch (IF), (2) instruction decode (ID), (3) _____, (4) _____, (5) execute operation (EO), dan (5) store result (SR).*
8. Pada saat pengambilan instruksi, jika (1) panjang _____ lebih besar dari (2) panjang _____ maka unit kontrol melakukan siklus baca memori lebih dari satu untuk melakukan pengambilan instruksi.
 9. Jika _____ antara CPU dan memori lebih kecil dari panjang word memori, maka unit kontrol melakukan siklus bus lebih dari satu kali untuk mendapatkan satu word memori. Contoh: Bus data adalah 16 bit dan panjang word memori 32 bit, dua siklus bus dibutuhkan untuk mengambil satu word.
 10. Panjang instruksi diketahui dari _____ instruksi tersebut. Karena itu, unit kontrol mengetahui panjang instruksi hanya setelah pengambilan word pertama instruksi.
 11. Ada dua jenis desain unit kontrol yaitu: (1) _____ dan (2) _____.
 12. Jenis unit kontrol yang menyimpan sinyal-sinyal kontrol sebagai pola-pola bit dalam sebuah memori ROM disebut _____.
 13. Jenis unit kontrol yang merupakan kumpulan sirkuit kombinasional (sirkuit-sirkuit digital) disebut _____.
 14. Dua jenis teknik microprogramming yang biasa dilakukan pada unit kontrol adalah: (1) _____ dan (2) _____.
 15. Dua keuntungan utama microprogramming adalah: (1) _____, dan (2) _____.
 16. Kekurangan utama microprogramming adalah _____.
 17. Mikroprogram yang disimpan dalam memori kontrol (ROM) biasa juga disebut _____.
 18. Mikroprogram yang menghasilkan eksekusi cepat tetapi membutuhkan memori kontrol yang besar adalah (1) _____, sedangkan yang eksekusinya lambat tetapi membutuhkan memori kontrol yang kecil adalah (2) _____.

SOAL-SOAL LATIHAN

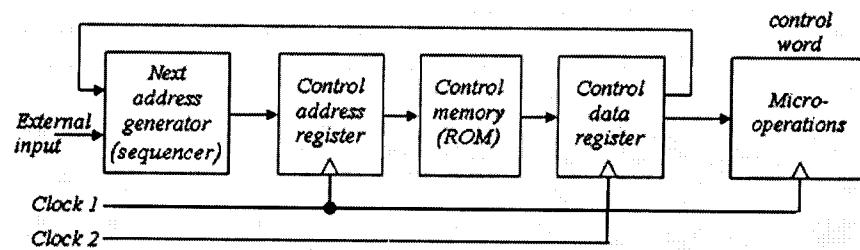
1. Jelaskan apa yang dimaksud dengan:
 - (a) Mikrooperasi
 - (b) Mikroinstruksi
 - (c) Mikroprogram
 - (d) Mikrokode

2. Sebuah mesin mempunyai unit kontrol seperti pada gambar berikut:



Buatlah microcode untuk mesin ini. Tingkah laku mesin sebagai berikut: Jika masukan Halt A set ke 1, maka keluaran mesin tetap berhenti (halt) selamanya dan keluaran pada jalur X selalu 1, dan pada V dan W selalu 0. Lampu waiting (keluaran V) diset ke 1 bila tidak ada masukan yang enable. Yaitu V menyala bila masukan A, B dan C adalah 0, dan mesin tidak halt. Bell berbunyi (W=1) pada setiap kejadian masukan (B=1 dan/atau C=1) kecuali jika mesin dalam keadaan halt. Masukan D dan keluaran S dapat digunakan untuk informasi keadaan untuk microcode Anda. Gunakan 0 untuk field yang tidak terjadi apa-apa. Arahan: Isi lebih dahulu tabel bagian paruh bawah.

3. Sebuah mikroinstruksi unit kontrol mempunyai 9 bit untuk penetapan mikrooperasi-mikrooperasi. Anggap dibagi menjadi dua field: satu mempunyai 5 bit dan yang lainnya 4 bit. Berapa banyak mikrooperasi di dalam CPU yang memungkinkan?
4. Sebuah format mikroinstruksi CPU memiliki lima control field terpisah. Jumlah mikrooperasi pada setiap field adalah sebagai berikut:
 $F_1 = 4, F_2 = 4, F_3 = 3, F_4 = 12, F_5 = 21$
 - a) Berapa total panjang mikroinstruksi yang diperlukan untuk mengakomodasi lima control field?
 - b) Jika murni dianut horizontal microprogramming tanpa encoding, berapa panjang mikroinstruksi yang akan diperoleh?
5. Sebuah prosesor memiliki konfigurasi hardware:
 - a) Jumlah register = 8
 - b) Operasi ALU: operasi aritmetika 8, operasi logika 8
 - c) Shifter: 4 operasi
 - d) Bus: bus tunggal
 Rancanglah format mikroinstruksi CPU tersebut.
6. Sebuah organisasi *microprogrammed control* (ditunjukkan pada gambar berikut) mempunyai waktu tunda propagasi masing-masing 40 ns untuk membangkitkan *next address*, 10 ns untuk mentransfer alamat ke *control address register*, 40 ns untuk mengakses *control memory (ROM)*, 10 ns untuk mentransfer mikroinstruksi ke dalam *control data register*, dan 40 ns untuk melakukan mikrooperasi yang diperlukan yang ditetapkan oleh *control word*. Berapa frekuensi clock maksimum yang dapat digunakan oleh kontrol tersebut? Berapa frekuensi clock yang akan diperlukan jika *control data register* tidak digunakan?



BAB 7

DESAIN MEMORI UTAMA SEMIKONDUKTOR

Sasaran bab ini:

1. Jenis-Jenis Memori Semikonduktor
2. Karakteristik Memori Semikonduktor
3. Memori Utama Semikonduktor

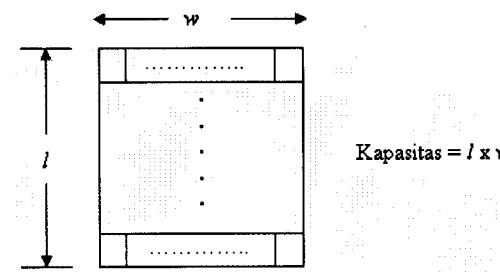
Memori digunakan untuk penyimpanan informasi. Informasi ini di dalam komputer adalah program (instruksi dan operand) dan informasi kontrol. Kapasitas memori dan kecepatan adalah dua hal penting yang merupakan pemanfaatan efektif komputer. Beberapa jenis memori hingga kini telah dikembangkan dan pada praktisnya terdapat banyak konsep manajemen memori yang ada. Pada bagian ini akan dibahas jenis-jenis memori dan difokuskan pada desain memori utama.

7.1 PARAMETER-PARAMETER MEMORI

Ada empat parameter penting yang signifikan dalam pemilihan sebuah memori komputer yaitu:

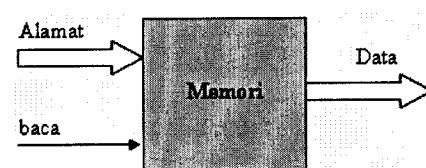
1. Kapasitas
2. Kecepatan
3. Latency
4. Bandwidth

Kapasitas: Memori dapat dipandang sebagai sebuah unit penyimpanan yang terdiri atas / jumlah lokasi yang masing-masing dapat menyimpan w jumlah bit (lihat Gambar 7.1). Dengan kata lain memori mempunyai / alamat dan panjang word w bit. Kapasitas total memori adalah / x w bit.

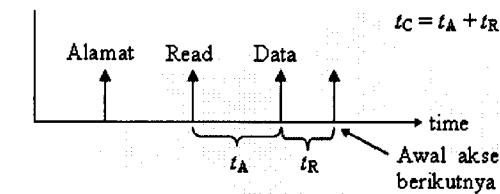


Gambar 7.1 Organisasi dan kapasitas memori

Kecepatan: Parameter penting memori adalah kecepatan operasi. Hal ini diukur dengan dua parameter: *waktu akses* t_A dan *waktu pemulihan* (*recovery time*) t_R . Untuk melakukan operasi baca, pertama, alamat lokasi dikirim ke memori yang disertai dengan sinyal kontrol "baca" (Gambar 7.2). Memori melakukan pendekodean (penerjemahan) alamat, memilih lokasi dan membaca isi yang ada dalam lokasi memori tersebut. *Waktu akses* adalah waktu yang diperlukan oleh memori untuk melengkapi operasi baca segera ketika menerima sinyal kontrol "baca" (Gambar 7.3). Umumnya waktu akses untuk operasi pembacaan dan penulisan adalah sama. Misalkan dua operasi pembacaan memori berurutan harus dilakukan. Selama operasi baca pertama, pembacaan informasi dari memori diberikan setelah waktu akses. Data ini dapat segera digunakan oleh CPU, walaupun memori masih sibuk dengan beberapa operasi internal untuk sejumlah waktu tambahan yang disebut dengan waktu pemulihan t_R . Selama waktu pemulihan t_R ini, akses memori lainnya (baca atau tulis) tidak dapat dimulai. Operasi berikutnya baru dapat dimulai hanya setelah waktu pemulihan berakhir. *Waktu siklus* adalah total waktu akses dan waktu pemulihan ($t_C = t_A + t_R$). Waktu ini adalah interval waktu minimum yang diperlukan dari awal operasi memori ke awal operasi memori berikutnya.



Gambar 7.2 Diagram blok operasi pembacaan memori



Gambar 7.3 Ilustrasi waktu akses dan waktu pemulihan memori

Latency: Beberapa jenis memori seperti hard disk, mempunyai organisasi internal yang dalam akses pertamanya ke suatu lokasi (sektor) mempunyai waktu akses yang lama, sedangkan lokasi yang berdekatan mempunyai waktu akses yang lebih singkat. *Latency* adalah waktu yang digunakan untuk mengakses lokasi (*word*) yang pertama dalam suatu rangkaian lokasi (blok lokasi).

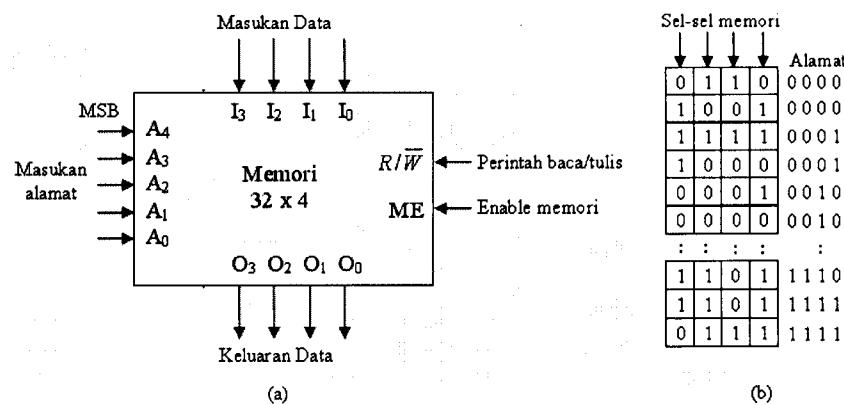
Bandwidth: *Bandwidth* adalah kecepatan transfer data memori yang dinyatakan dalam jumlah byte per detik.

7.2 OPERASI MEMORI SECARA UMUM

Walaupun setiap jenis memori berbeda cara operasi internalnya, tetapi prinsip-prinsip operasi dasar tertentu adalah sama untuk semua sistem memori. Pemahaman ide-ide dasar akan membantu dalam mempelajari perangkat memori secara individu.

Setiap sistem memori memerlukan beberapa jenis saluran input dan output untuk melaksanakan fungsi-fungsi berikut:

1. Memilih alamat memori yang akan diakses untuk operasi baca atau operasi tulis.
2. Memilih operasi baca atau operasi tulis untuk dilaksanakan.
3. Mensuplai data input untuk penyimpanan dalam memori ketika operasi penulisan.
4. Menahan data output yang berasal dari memori ketika operasi pembacaan.
5. *Enable* (atau *disable*) memori agar memori akan (tidak akan) merespons masukan alamat dan perintah baca/tulis.



Gambar 7.4 (a) Diagram memori 32 x 4, (b) Susunan sel-sel memori 32 x 4

Gambar 7.4(a) mengilustrasikan fungsi-fungsi dasar memori dalam sebuah diagram sederhana dari sebuah memori 32 x 4 yang menyimpan 32 word di mana setiap word adalah 4 bit. Karena ukuran word adalah empat bit, maka terdapat empat saluran input data I_0 sampai I_3 dan empat saluran output data O_0 sampai O_3 . Ketika suatu operasi tulis data akan disimpan dalam memori maka harus digunakan saluran input data. Ketika suatu operasi baca dilakukan, maka word yang dibaca dari memori akan tampil pada saluran output data.

7.2.1 Masukan-Masukan Alamat

Memori yang dapat menyimpan 32 word, berarti memori ini memiliki 32 lokasi penyimpanan dan karena itu terdapat alamat-alamat biner 32 macam yaitu mulai dari 00000 sampai 11111 (0 sampai 31 dalam desimal). Jadi, ada lima masukan alamat yaitu A₀ sampai A₄. Untuk mengakses salah satu lokasi memori untuk operasi baca atau tulis, maka digunakan lima bit kode alamat yang diberikan pada masukan alamat. Secara umum, masukan-masukan alamat N diperlukan untuk memori yang mempunyai kapasitas 2^N word.

Kita dapat melukiskan memori Gambar 7.4(a) sebagai sebuah susunan 32 register, di mana setiap register menyimpan sebuah word 4-bit seperti yang diilustrasikan pada Gambar 7.4(b). Setiap lokasi alamat yang ditunjukkan berisi empat sel memori yang menahan 1 dan 0. Misalnya, word data

0110 disimpan pada alamat 00000, word data 1001 disimpan pada alamat 00001, dan seterusnya.

Contoh 7.3 Sebuah memori mempunyai kapasitas 4K x 8

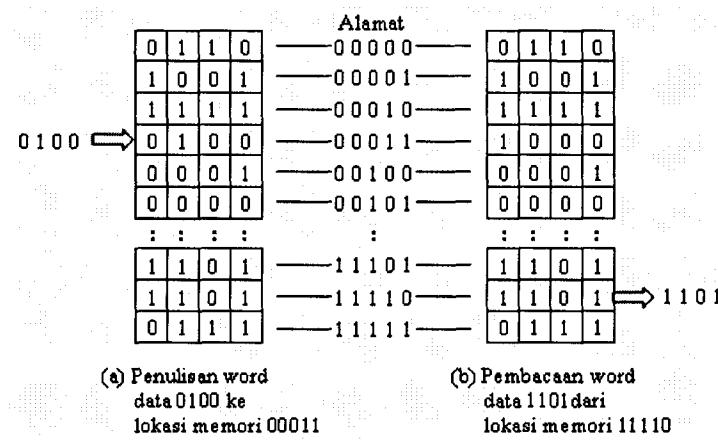
- Berapa banyak saluran masukan data dan saluran keluaran data yang dimiliki?
- Berapa banyak saluran alamat yang dimiliki?
- Berapa kapasitas dalam byte?

Solusi:

- Masing-masing 8 bit, karena ukuran word adalah 8
- Memori menyimpan $4K = 4 \times 1024 = 4096$ word. Jadi, terdapat 4096 alamat memori. Karena itu $4096 = 2^{12}$ yang artinya memerlukan saluran alamat sebanyak 12 bit yang dapat mengalami salah satu lokasi alamat dari 4096 lokasi.
- Delapan bit adalah satu byte. Jadi kapasitas memori adalah 4096 byte.

7.2.2 Masukan R/\bar{W}

Masukan R/\bar{W} ini mengontrol operasi memori apakah operasi baca (R) atau tulis (W). Input diberi label R/\bar{W} , karena tidak ada tanda garis di atas R ini mengindikasikan operasi baca terjadi jika $R/\bar{W} = 1$. Tanda garis di atas W berarti operasi tulis terjadi jika $R/\bar{W} = 0$. Demikian halnya untuk label-label yang serupa.



Gambar 7.5 Ilustrasi sederhana operasi pembacaan dan penulisan pada memori 32x4

Suatu ilustrasi sederhana operasi baca dan tulis ditunjukkan pada Gambar 7.5. Bagian (a) menunjukkan word data 0100 yang akan dituliskan ke dalam register memori pada alamat 00011. Word data yang telah dikenakan pada saluran input data menggantikan data yang sebelumnya tersimpan pada alamat 00011. Bagian (b) menunjukkan word data 1101 yang dibaca dari alamat 11110. Word data ini akan tampil pada saluran output data memori. Setelah operasi baca, word data 1101 tetap tersimpan pada alamat 11110. Dengan kata lain, operasi baca tidak mengubah data yang tersimpan.

Contoh 7.1

Dengan mengacu pada Gambar 7.5, jelaskan kondisi pada masukan dan keluaran bila isi lokasi alamat 00100 akan dibaca.

Solusi:

Masukan alamat: 00100

Masukan data: xxxx (tidak digunakan)

R/\bar{W} : HIGH

ME: HIGH

Keluaran data: 0001

Contoh 7.2

Dengan mengacu pada Gambar 7.5, jelaskan kondisi pada masukan dan keluaran bila data 1110 akan ditulis ke lokasi alamat 01101.

Solusi:

Masukan alamat: 01101

Masukan data: 1110

R/\bar{W} : LOW

ME: HIGH

Keluaran data: xxxx (tidak digunakan, biasanya keadaan impedansi tinggi)

Contoh 7.3

Sebuah memori mempunyai kapasitas $4K \times 8$

- Berapa banyak saluran masukan data dan saluran keluaran data yang dimiliki?
- Berapa banyak saluran alamat yang dimiliki?
- Berapa kapasitas dalam byte?

Solusi:

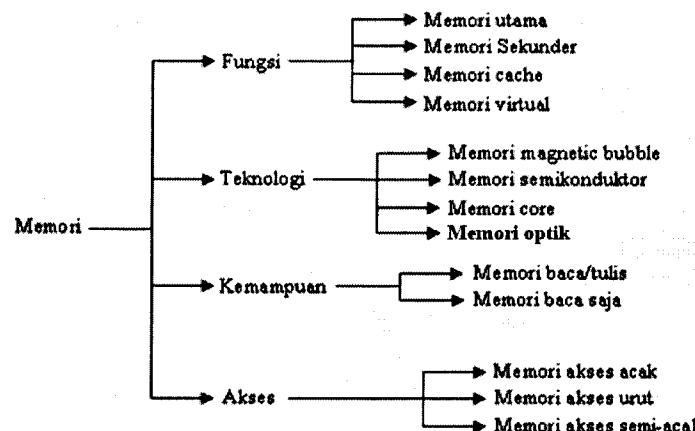
- Masing-masing 8 bit, karena ukuran word adalah 8
- Memori menyimpan $4K = 4 \times 1024 = 4096$ word. Jadi, terdapat 4096 alamat memori. Karena itu $4096 = 2^{12}$ yang artinya memerlukan saluran alamat sebanyak 12 bit yang dapat mengalami salah satu lokasi alamat dari 4096 lokasi.
- Delapan bit adalah satu byte. Jadi kapasitas memori adalah 4096 byte.

7.2.3 Enable Memori

Banyak sistem-sistem memori mempunyai cara dalam pengontrolan untuk menanggapi atau tidak menanggapi masukan-masukan dengan menggunakan saluran kontrol *enable* seperti yang ditunjukkan pada Gambar 7.4, yang disebut masukan *ENABLE MEMORI*. Penamaan ini dapat mempunyai nama yang berbeda-beda misalnya chip enable (CE) atau chip select (CS). Di sini ditunjukkan masukan aktif-HIGH yang meng-enable memori untuk beroperasi bila level HIGH dipertahankan selama pembacaan atau penulisan. Sedangkan bila diberikan masukan level LOW maka memori tersebut menjadi lumpuh/disable untuk merespons masukan *R/W*. Jenis masukan ini berguna ketika beberapa modul (bank) memori digabungkan untuk membentuk sebuah memori yang lebih besar.

7.3 KLASIFIKASI MEMORI

Klasifikasi memori yang umum adalah berdasarkan fungsi, teknologi, modus penggunaannya (kemampuan baca/tulis) serta metode akses, seperti yang ditunjukkan pada Gambar 7.6.



Gambar 7.6 Klasifikasi memori

7.3.1 Kemampuan Baca/Tulis

Memori ROM (*read only memory*) hanya boleh dibaca oleh CPU. Memori baca/tulis dapat melakukan operasi baca dan tulis. Dalam industri komputer, memori semikonduktor baca/tulis dengan istilah RAM (diganti dengan RWM) walaupun secara teknik ROM semikonduktor juga adalah memori akses acak (*random access memory*).

7.3.2 Metode Akses

Metode akses merujuk pada mekanisme perbedaan pengaksesan lokasi dalam memori. Memori akses acak adalah memori yang membolehkan pengaksesan ke suatu lokasi tanpa ada keterkaitan dengan posisi fisik dan tidak bergantung pada lokasi lain. Dengan kata lain, waktu akses sama untuk semua lokasi. Contoh memori akses acak adalah memori semikonduktor (RAM dan ROM). Pada memori akses urut (*sequential access*), pembacaan suatu lokasi dilakukan secara berurut di mana setelah operasi baca atau tulis dilakukan, head baca/tulis diletakkan di depan lokasi berikutnya. Karena itu waktu akses akan berbeda untuk setiap lokasi. Contoh memori akses urut adalah pita magnetik dan pita kertas. Pada memori akses semi-acak, pemilihan lokasi yang akan diakses menggunakan dua langkah: satu akses acak dan lainnya akses urut. Contoh memori akses semi-acak adalah memori magnetik (floppy disk dan hard disk) yang dikenal juga dengan nama memori akses langsung (*direct access memory*).

7.3.3 Mode-Mode Penggunaan dan Fungsi

Klasifikasi ini berdasarkan pada peranan memori pada komputer. Pada dasarnya ada lima peranan memori dalam hubungannya dengan CPU: register internal, memori utama, memori sekunder, memori cache, dan memori virtual. Register memiliki peran terbatas dalam penyimpanan informasi sementara ketika program berjalan. Register memiliki waktu akses yang singkat/cepat. Register tertentu hanya dapat diakses oleh CPU sedangkan yang lainnya dapat juga diakses oleh program.

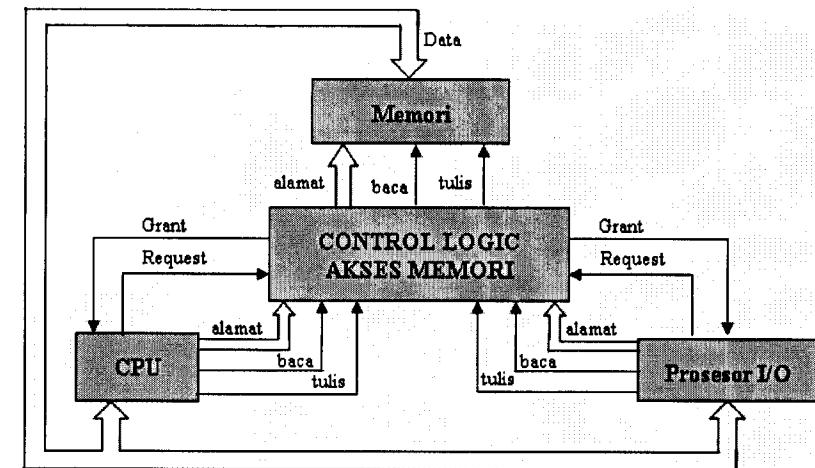
Memori Utama

CPU mengambil (*fetch*) instruksi dari memori utama ketika program dijalankan. Memori ini dikenal juga dengan nama memori program dan memori primer. Untuk mengeksekusi program, program ditransfer ke dalam

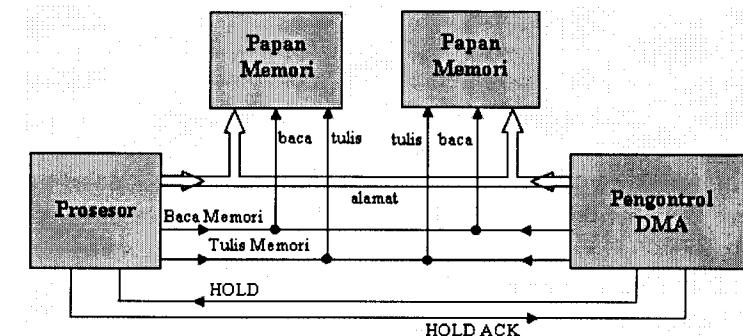
memori utama. Memori utama adalah jenis memori akses acak. Pada suatu CPU, kapasitas maksimum memori utama adalah 2^n lokasi di mana n adalah jumlah bit alamat memori. Secara praktis, komputer dapat mempunyai memori fisik lebih kecil karena pertimbangan harga. Namun, pengguna tidak perlu khawatir mengenai kapasitas memori (memori utama) fisik secara eksak karena ada cara tak langsung yang tersedia dalam penanganan program. Konsep memori virtual akan membantu menangani hal ini.

Secara fisik memori utama dapat diintegrasikan ke dalam sebuah IC prosesor dan juga ke mikrokontroler. Memori ini disebut *on-chip memory* atau memori internal. Biasanya ukurannya terbatas karena pertimbangan ruang dan harga IC. Walaupun kita berharap memori utama mempunyai kemampuan untuk baca dan tulis, namun beberapa bagian memori utama yang dimiliki komputer saat ini biasanya hanya dibaca, digunakan untuk menyimpan program permanen dan *boot strap loader*.

Pada Gambar 7.7 dan Gambar 7.8 ditunjukkan jenis CPU—antarmuka memori utama pada sistem yang besar (komputer mainframe) dan pada mikrokomputer. Pada komputer mainframe, prosesor I/O (*data channel*) dapat juga mengakses memori utama untuk melakukan operasi input/output pada mode DMA. *Memory access control logic* meng-arbitrasi CPU dan prosesor I/O, dalam hal ini keduanya mencoba memulai akses memori secara simultan. Umumnya prosesor I/O mendapat prioritas untuk menghindari waktu tunda transfer I/O. Pada mikrokomputer, pengontrol DMA menjaga akses memori untuk peripheral kecepatan tinggi. Namun, CPU merupakan master bus. Pengontrol DMA meminta bus dengan sinyal HOLD dan CPU memberikan persetujuan bus dengan sinyal HOLD ACK. Jika CPU telah menggunakan bus atau memerlukan cadangan untuk suatu permintaan yang nantinya penting (ke depan), maka CPU dapat menunda persetujuannya.



Gambar 7.7 Antarmuka CPU—memori pada komputer mainframe

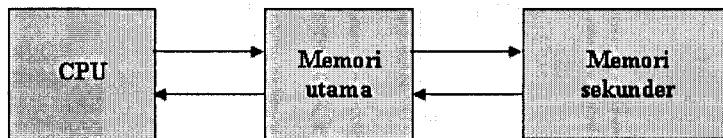


Gambar 7.8 Antarmuka CPU—memori pada mikrokomputer

Memori Sekunder

Memori sekunder dikenal juga dengan nama memori pembantu (*auxiliary memory*). Memori ini digunakan untuk menyimpan program dan data dalam volume yang besar. Memori ini lambat dan murah dibandingkan dengan memori utama. Saat ini yang biasa digunakan sebagai memori sekunder adalah floppy disk, hard disk, pita magnetik dan disk optik. *Magnetic drum*, pita kertas dan kartu kertas merupakan memori sekunder yang saat ini sudah tidak digunakan lagi. Untuk CPU (dan sistem operasi),

memori sekunder menyimpan beberapa informasi yang dapat digeser ke memori utama ketika diperlukan (Gambar 7.9). CPU mengakses memori sekunder sebagai perangkat peripheral. Namun, sistem operasi dapat memanipulasi memori sekunder sebagai perpanjangan dari memori utama seperti yang terlihat pada konsep memori virtual. Umumnya memori sekunder ada yang akses urut (pita magnetik) dan semi-acak (hard disk).



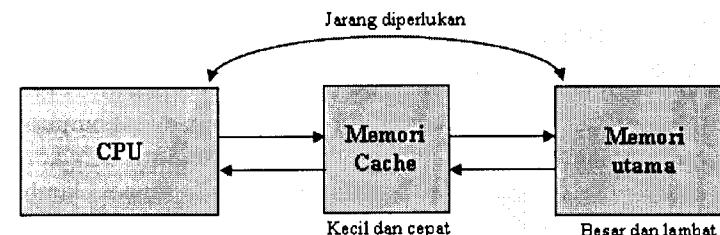
Gambar 7.9 Penggunaan memori sekunder

Memori Cache

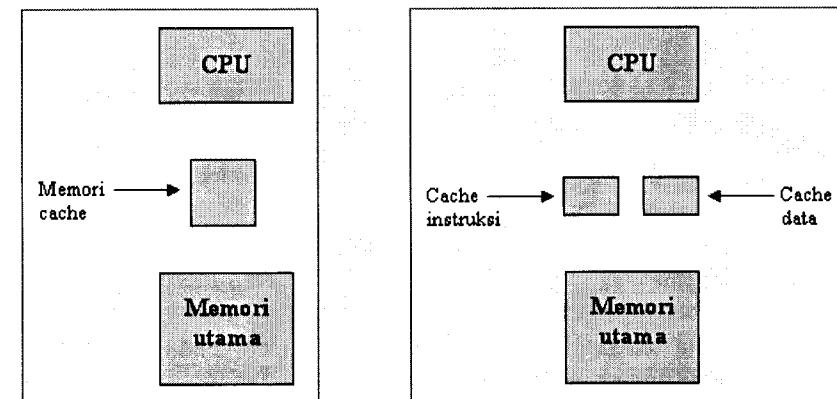
Memori cache merupakan sebuah penyangga tengah antara CPU dan memori utama. Kapasitasnya lebih kecil dibandingkan dengan memori utama karena sangat mahal. Kecepatannya beberapa kali dari memori utama. Karena itu sering diperlukan dalam menyalin data/instruksi dari memori utama ke memori cache. Ketika diperlukan oleh CPU, data/instruksi disediakan dari memori cache yang menggantikan pengaksesan memori utama. Secara efektif, waktu yang dibutuhkan dalam akses memori utama diambil alih oleh akses memori cache yang cepat. Namun jika data/instruksi yang diperlukan tidak ada dalam memori cache, maka barulah dilakukan pengaksesan ke memori utama (Gambar 7.10). Memori cache dapat dibagi dalam dua jenis (Gambar 7.11):

1. Memori cache gabung (menyimpan instruksi dan data)
2. Memori cache instruksi dan cache data terpisah.

Memori cache secara fisik dapat diintegrasikan dengan prosesor IC sebagai cache internal yang dikenal sebagai *on-chip cache*.



Gambar 7.10 Konsep memori cache



Gambar 7.11 Konsep penggabungan dan pemisahan instruksi dan data pada memori cache

Memori Virtual

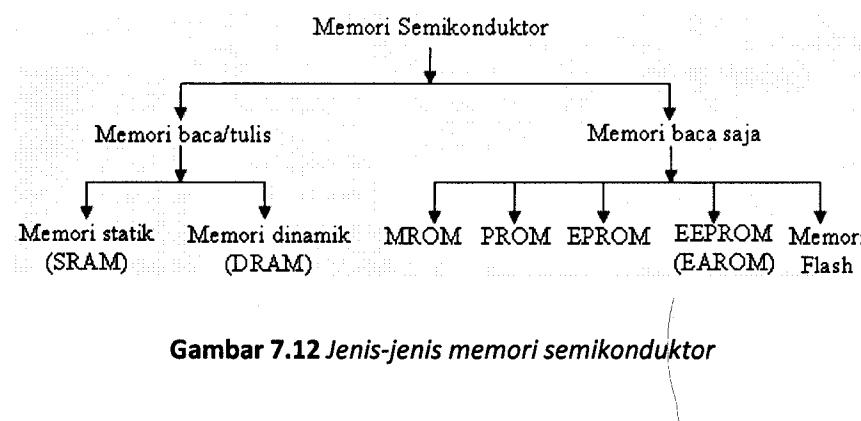
Memori virtual adalah suatu fitur/konsep yang membantu untuk menjalankan program yang panjang dalam sebuah memori fisik yang kecil. Sistem operasi mengatur program (besar) dengan hanya menyimpan sebagian di dalam memori utama dan menggunakan memori sekunder untuk menyimpan program keseluruhan. Ketika bagian program tertentu tidak terdapat di dalam memori utama, maka diambil dari memori sekunder. Perangkat keras CPU dan sistem operasi bekerja sama dalam melakukan hal tersebut pada saat program dijalankan.

7.3.4 Teknologi Memori

Dahulu dalam beberapa tahun memori inti magnetik banyak digunakan sebagai memori utama pada komputer *mainframe* dan komputer mini, tetapi sekarang secara total digantikan dengan memori semikonduktor. Memori *magnetic bubble* adalah memori serial, sangat lambat dan membutuhkan penyegaran ulang yang sirkuit kontrolnya sudah terdapat di dalamnya.

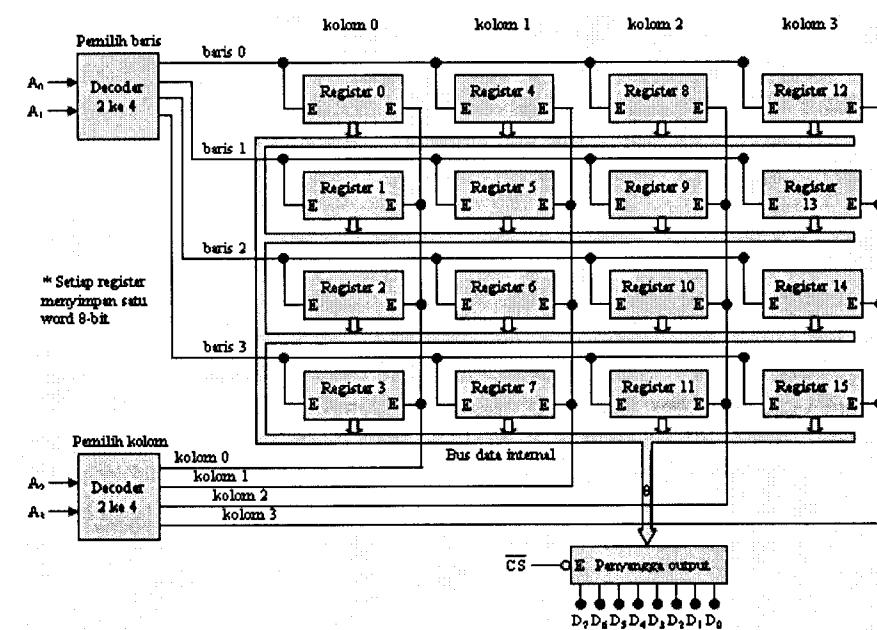
Memori inti magnetik menggunakan prinsip magnetisasi untuk menyimpan 0 atau 1 dalam inti magnetik. Saat ini sudah tidak digunakan lagi sebagai memori utama dan telah digantikan oleh memori semikonduktor karena selain lambat, juga menggunakan daya dan ruang yang lebih besar. Sifat baik yang dimiliki memori inti adalah *non-volatile* yang alami. Memori inti tetap menyimpan isinya ketika daya dilepas.

Keuntungan utama memori semikonduktor dibanding memori inti adalah konsumsi daya rendah, waktu akses singkat, *non-destructive read out* dan ukuran kecil. Ada dua jenis memori semikonduktor yaitu paralel dan serial. Peranti CCD (*charge coupled device*) adalah memori serial yang memerlukan penyegaran ulang dan mempunyai waktu latency yang panjang, Karena itu tidak populer secara komersial. Pada Gambar 7.12 ditunjukkan jenis-jenis memori semikonduktor.



7.4 ARSITEKTUR ROM

Aristektur internal (struktur) ROM sangat kompleks dan karena itu pembahasan arsitektur ROM dapat kita sederhanakan seperti yang ditunjukkan pada Gambar 7.13. Pada arsitektur ROM 16×8 ini terdapat empat bagian dasar yaitu larik register (*register array*), decoder baris, decoder kolom dan penyangga output.



Gambar 7.13 Arsitektur ROM 16×8

7.4.1 Larik Register

Larik register menyimpan data yang telah diprogram ke dalam ROM. Setiap register berisi banyak sel-sel memori yang setara dengan ukuran word. Pada kasus ini, setiap register menyimpan word 8-bit. Kita dapat menetapkan posisi setiap register pada sebuah baris dan kolom khusus. Misalnya, register 0 ada dalam baris 0, kolom 0, dan register 9 ada dalam baris 1, kolom 2.

Delapan keluaran data dari setiap register dihubungkan ke bus data internal. Setiap register mempunyai dua saluran masuk enable (E), keduanya harus dalam keadaan HIGH supaya data register dapat ditempatkan pada bus.

7.4.2 Decoder Alamat

Kode alamat $A_3A_2A_1A_0$ yang digunakan menentukan isi register (8-bit) mana dalam larik yang akan ditempatkan pada bus data (enable). Bit-bit alamat A_1A_0 diumpulkan ke decoder yang akan mengaktifkan atau memilih satu saluran baris-pilih dan bit-bit alamat A_3A_2 diumpulkan pada decoder kedua yang akan mengaktifkan atau memilih satu saluran kolom-pilih. Hanya ada satu register yang terpilih pada baris dan kolom oleh masukan alamat, dan register inilah yang di-enable.

Contoh 7.4

Register manakah yang akan terpilih (enable) oleh alamat 1101?

Solusi:

$A_3A_2 = 11$ akan menyebabkan decoder kolom mengaktifkan kolom saluran pilih 3, dan $A_1A_0 = 01$ akan menyebabkan decoder baris untuk mengaktifkan saluran pilih 1. Ini akan menempatkan level HIGH pada kedua masukan enable register 13, dengan demikian menyebabkan data keluaran ditempatkan pada bus.

Contoh 7.5

Alamat berapakah yang akan meng-enable register 7?

Solusi:

Masukan-masukan enable register ini dihubungkan ke saluran pilih baris 3 dan kolom 1. Untuk memilih baris 3, maka input yang harus diberikan ke decoder baris adalah $A_1A_0 = 11$ dan untuk memilih kolom 1 harus diberikan input ke decoder kolom $A_3A_2 = 01$. Jadi alamat yang dibutuhkan adalah $A_3A_2A_1A_0 = 0111$

7.4.3 Penyangga Output

Register yang di-enable oleh masukan alamat akan menempatkan data pada bus data. Data ini diumpulkan ke penyangga output, yang akan melewaskan data ke keluaran-keluaran data eksternal, hal ini disediakan dengan $\overline{CS} = \text{LOW}$. Jika $\overline{CS} = \text{HIGH}$, maka penyangga output berada dalam keadaan impedansi tinggi (*tristate*), dan D_7 sampai D_1 akan mengambang.

Arsitektur yang ditunjukkan pada Gambar 7.13 mirip dengan IC pada umumnya, bergantung pada jumlah word data yang disimpan. Register pada beberapa ROM tidak akan disusun dalam larik persegi. Contoh, Intel 27C64 adalah sebuah ROM CMOS yang menyimpan 8192×8 . Sebanyak 8192 register disusun dalam sebuah larik 256 baris x 32 register. Rentang kapasitas ROM mulai dari 256×4 sampai $8M \times 8$.

Contoh 7.6

Jelaskan arsitektur internal dari sebuah ROM yang menyimpan 4KB dan menggunakan sebuah larik register persegi.

Solusi:

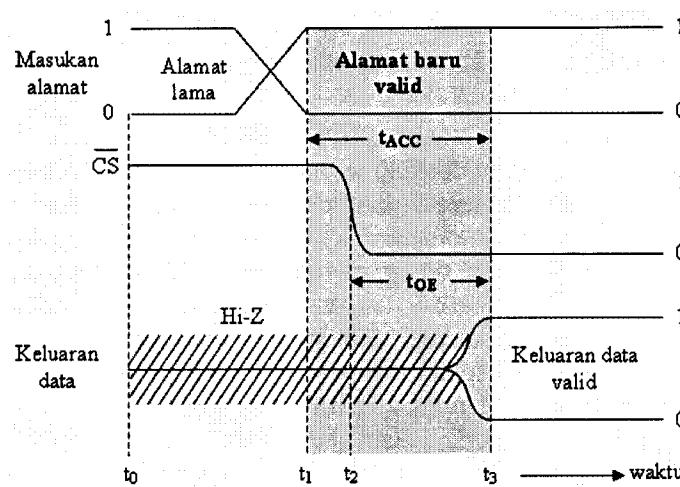
Memori 4K sebenarnya adalah $4 \times 1024 = 4096$, jadi ROM ini menyimpan 4096 byte, atau 4096×8 bit. Setiap word tersimpan pada sebuah register 8-bit, dan terdapat 4096 register yang dihubungkan secara bersama ke sebuah bus data internal ke chip. Karena $4096 = 64^2$, register disusun dalam larik 64×64 , yaitu 64 baris dan 64 kolom. Hal ini memerlukan decoder 6-ke-64 untuk mendekodekan enam masukan alamat untuk memilih baris, dan sebuah decoder 6-ke-64 untuk mendekodekan enam masukan alamat untuk memilih kolom. Jadi, total masukan alamat yang dibutuhkan adalah 12. Hal ini dapat terlihat dari $4096 = 2^{12}$, dan terdapat 4096 lokasi alamat berbeda.

7.5 PEWAKTUAN ROM

Ketika terjadi operasi pembacaan, terdapat waktu tunda perambatan (*propagation delay*) antara pemberian sebuah input ROM dan kemunculan data pada output. Waktu tunda ini disebut waktu akses t_{ACC} yang merupakan pengukuran kecepatan operasi ROM. Waktu akses digambarkan secara grafis dengan bentuk gelombang pada Gambar 7.14.

Bentuk gelombang teratas mewakili masukan alamat, bentuk gelombang yang di tengah mewakili keluaran data. Pada waktu t_0 masukan masukan alamat semua berada pada beberapa level khusus, beberapa level HIGH dan beberapa level LOW. Sinyal $\overline{CS} = \text{HIGH}$, sehingga keluaran data ROM berada dalam keadaan impedansi tinggi (Hi-Z), yang dinyatakan dengan garis miring.

Tepat sebelum t_1 , masukan alamat berubah ke alamat baru untuk operasi pembacaan. Pada saat t_1 alamat baru menjadi valid; yaitu setiap masukan alamat berada pada level logika yang valid. Pada titik ini sirkuit ROM internal mulai mendekode masukan alamat baru untuk memilih register yaitu untuk mengirim data ke penyanga output. Pada saat t_2 masukan \overline{CS} diaktifkan untuk meng-enable penyanga output. Akhirnya pada saat t_3 keluaran berubah dari keadaan Hi-Z (impedansi tinggi) ke data yang valid yang menyatakan data tersimpan pada alamat yang ditetapkan.



Gambar 7.14 Pewaktuan khas untuk operasi baca sebuah ROM

Waktu tunda antara t_1 yaitu ketika alamat baru menjadi valid dan t_3 , ketika keluaran data menjadi valid adalah waktu akses t_{ACC} . Waktu akses ROM bipolar khas berada dalam rentang 30 sampai 90 ns; waktu akses untuk perangkat-perangkat NMOS dalam rentang 35 sampai 500 ns.

Pengembangan pada teknologi CMOS telah membawa waktu akses berada dalam rentang 20 sampai 60 ns. Akibatnya perangkat-perangkat bipolar dan NMOS sudah jarang diproduksi dalam ROM yang baru (yang besar).

Parameter penting lainnya adalah waktu enable keluaran, t_{OE} yaitu waktu tunda antara masukan \overline{CS} dan keluaran data valid. Nilai khas t_{OE} untuk ROM adalah 20 ns untuk bipolar, 25 sampai 100 ns untuk NMOS, dan 12 sampai 50 ns CMOS.

7.6 JENIS-JENIS ROM

Sekarang kita telah mempunyai pemahaman umum tentang arsitektur internal dan operasi eksternal perangkat ROM. Kita akan melihat pada jenis-jenis ROM untuk mengetahui bagaimana perbedaan cara pemrogramannya, cara penghapusannya, dan cara memprogram ulang masing-masing jenis ROM.

Memori ROM dibagi menjadi lima jenis: MROM, PROM, EPROM, EEPROM (atau dikenal juga dengan EAROM) dan memori flash.

7.6.1 MROM

Memori MROM (*Mask-programmed Read Only Memory*) adalah memori yang isinya diprogram oleh pabrik secara tersembunyi di perusahaan pembuatnya, di mana isinya dipesan oleh pembeli. MROM diprogram satu kali saja dan selanjutnya hanya dapat dibaca dan tak dapat dihapus. Jika ada satu bit yang salah program, maka chip MROM tidak dapat digunakan. Sebuah negatif fotografi yang disebut *mask* digunakan untuk mengontrol interkoneksi elektrik pada chip. Suatu mask khusus diperlukan untuk setiap set informasi berbeda untuk disimpan ke dalam ROM. Karena mask ini sangat mahal harganya, maka jenis ROM hanya ekonomis jika Anda membutuhkan kuantitas yang sangat banyak pada ROM yang sama. Kelemahan utama ROM jenis ini karena tidak dapat diprogram ulang pada kasus yang memerlukan perubahan desain pada data yang tersimpan. Akibatnya ROM harus diganti dengan yang baru sesuai data yang akan diperlukan dalam ROM. Beberapa jenis *user-programmable ROM* telah dikembangkan untuk mengatasi kelemahan ini. Namun demikian MROM tetap merupakan hal yang menarik untuk pendekatan yang paling ekonomis bila diperlukan ROM yang telah diprogram dalam kuantitas yang besar.

MROM biasanya disebut saja ROM, karena hal ini dapat membingungkan karena istilah ROM sebenarnya merepresentasikan kategori besar perangkat tersebut, dalam keadaan operasi normal memori ini hanya dapat dibaca.

7.6.2 PROM

PROM (*Programmable Read Only Memory*) adalah memori yang dapat diprogram oleh pemakai (*customer*). Pemakai membeli PROM yang masih kosong dan mengisinya sesuai dengan keinginannya dengan menggunakan *PROM programmer*. Di dalamnya terdapat *fuse* (semacam sekring) kecil. Fuse ini dibuka dengan cara dibakar pada saat pemrograman. Chip PROM hanya dapat diprogram sekali saja dan isinya tak dapat dihapus. Memori ini murah dan menarik bagi pemakai untuk membeli PROM di pasaran daripada memesan MROM pada pabrik.

7.6.3 EPROM

EPROM (*Erasable Programmable Read Only Memory*) merupakan memori yang selain dapat diprogram juga dapat dihapus. *EPROM programmer* digunakan untuk menyimpan isi pada lokasi yang berbeda dari EPROM. Ketika dilakukan pemrograman, muatan listrik dijebak dalam suatu daerah gate yang terisolasi. Untuk memprogram sebuah EPROM digunakan tegangan dc yang relatif tinggi sebesar 25 volt. Pulsa high yang digunakan lebarnya 10-55 millidetik. Karena EPROM merupakan organisasi byte, maka pemrograman satu byte pada satu waktu dilakukan dengan menggunakan satu pulsa. Isi sebuah EPROM dapat dihapus dengan memaparkannya pada cahaya ultra violet dengan durasi sekitar 30 menit. Hal ini dilakukan dengan cara meletakkannya pada peralatan penghapus EPROM dan melewatkannya cahaya ultra violet melalui sebuah jendela quartz. Setelah diprogram, maka jendela quartz tersebut harus ditutup kembali dengan menggunakan pelindung (*shield/stiker*). Penghapusan EPROM dilakukan secara keseluruhan karena tidak dapat dilakukan penghapusan pada sebagian lokasi tertentu saja.

7.6.4 EEPROM

Memori EEPROM (*Electrically Erasable Programmable ROM*) atau dikenal juga dengan nama EAROM (*Electrically Alterable ROM*) diprogram dan

dihapus secara listrik (menggunakan listrik). Memori ini dapat dihapus dan diprogram ulang sekitar 10.000 kali. EAROM dapat diprogram dan dihapus pada suatu lokasi tertentu yang dipilih. Juga dapat dihapus dan diprogram ulang secara dinamis tanpa harus mengeluarkan IC EAROM dari sirkuit.

7.6.5 Memori Flash

Memori flash adalah sebuah EEPROM jenis khusus yang dapat dihapus dan diubah dalam blok pada satu operasi besar (seperti kilat). Penggunaannya terpisah dari komputer, digunakan pada sejumlah peralatan seperti *mobile phone*, kamera digital, LAN switch, dan sebagainya. Pada komputer, memori flash digunakan dalam dua cara:

1. Pengganti EPROM yang berisi BIOS. Ini dikenal dengan BIOS-flash. Membolehkan memperbarui isi BIOS dengan mudah tanpa mencabut IC dari sirkuit.
2. Pengganti hard disk. Opsi ini menarik karena beberapa keuntungan seperti tidak berat (ringan), konsumsi daya rendah, data mudah dibawa-bawa (*portable*).

Saat ini sedang marak "memori stick" yang terbuat dari memori flash yang disambungkan pada USB (*universal serial bus*) pada sebuah personal computer.

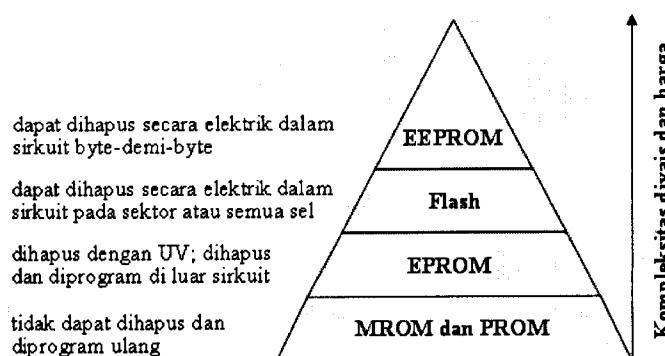
EPROM adalah memori non-volatile yang menawarkan waktu akses yang cepat (khas 120 ns) dan mempunyai kepadatan tinggi dan harga per bit rendah. Namun demikian jika ingin dihapus atau diprogram ulang, maka harus dilepaskan dari sirkuit/sistem. EEPROM juga adalah memori yang non-volatile yang menawarkan waktu akses yang cepat, dan membolehkan penghapusan dengan cepat dalam sirkuit dan pemrograman ulang byte secara individu. EEPROM kepadatannya rendah dan harganya lebih mahal dibandingkan dengan EPROM.

Tantangan bagi para insinyur semikonduktor adalah membuat memori non-volatile EEPROM yang dapat dihapus dalam sirkuit secara elektrik, tetapi dengan kepadatan dan harga yang mendekati EPROM, sementara tetap menjaga waktu akses baca yang tinggi. Respons terhadap tantangan tersebut telah direalisasikan dalam memori flash.

Secara struktur, sebuah sel memori flash adalah seperti sel EPROM transistor-tunggal sederhana (dan tidak seperti sel EEPROM dua-transistor

yang kompleks), hanya sedikit lebih besar. Memori flash mempunyai lapisan *gate-oxide* yang tipis yang membolehkan penghapusan secara elektrik tetapi dapat dibangun dengan kepadatan yang lebih tinggi daripada EEPROM. Harga memori flash sangat rendah dibanding EEPROM, walaupun masih tidak mendekati EPROM. Gambar 7.15 mengilustrasikan kecenderungan pada berbagai memori semikonduktor non-volatile. Fleksibilitas penghapusan/pemrograman meningkat (dari dasar sampai ke puncak piramida), demikian pula kompleksitas dan harga juga meningkat. MROM dan PROM merupakan perangkat yang paling sederhana dan paling murah, tetapi mereka tidak dapat dihapus dan diprogram ulang. EEPROM merupakan perangkat yang paling kompleks dan mahal karena dapat dihapus dan diprogram ulang dalam sirkuit berdasarkan byte-demi-byte.

Memori flash disebut demikian karena waktu penulisan dan penghapusannya cepat. Pada umumnya chip memori flash menggunakan operasi penghapusan besar di mana semua sel dalam chip dihapus secara simultan. Proses penghapusan besar ini secara khas membutuhkan ratusan millidetik dibandingkan dengan 20 menit untuk UV EPROM. Beberapa memori flash yang baru menawarkan mode penghapusan sektor, di mana sektor-sektor khusus dari larik memori (misalnya 512 byte) dapat dihapus dalam satu waktu.. Hal ini menghindari keharusan untuk menghapus dan memprogram ulang semua sel bila hanya satu bagian dari memori yang memerlukan pembaruan. Waktu tulis memori flash yang khas adalah 10 μ s per byte dibanding 100 μ s untuk kebanyakan EPROM terbaru dan 5 ms untuk EEPROM (termasuk waktu penghapusan byte otomatis).



Gambar 7.15 Tingkatan kompleksitas memori semikonduktor non-volatile

7.7 APLIKASI ROM

Pada umumnya perangkat ROM dapat diprogram ulang kecuali MROM dan PROM, jadi secara teknis mereka bukan memori read-only. Namun, istilah ROM dapat tetap digunakan untuk memasukkan EPROM, EEPROM dan memori flash karena ketika pada operasi normal, isi yang disimpan oleh perangkat ini hampir tidak berubah sesering dia dibaca. Jadi semua ROM dimasukkan sebagai perangkat memori semikonduktor non-volatile, dan mereka digunakan dalam aplikasi di mana penyimpanan informasi yang non-volatile, data atau kode-kode program diperlukan dan pada penyimpanan data yang jarang atau tidak pernah berubah. Di sini diberikan beberapa daerah aplikasi yang paling umum.

7.7.1 Firmware

Aplikasi ROM yang paling luas adalah dalam penyimpanan data dan kode-kode program yang harus tersedia pada sistem-sistem berbasis mikroprosesor ketika catu daya dihidupkan. Data dan kode-kode program ini disebut *firmware* karena mereka tetap tersimpan di dalam hardware (yaitu chip-chip ROM) dan tidak ada yang menghapus selama operasi sistem normal. Beberapa PC, komputer-komputer bisnis dan komputer laptop menyimpan program-program sistem operasinya dan interpreter-interpreter bahasa (misalnya BASIC, Pascal) dalam firmware ROM agar komputer dapat digunakan segera ketika catu daya dihidupkan.

Selain pada mikrokomputer yang dikenal, banyak sistem-sistem berbasis mikroprosesor yang menggunakan firmware ROM. Produk-produk konsumen seperti mobil, VCR, CD player, microwave-oven dan juga semua jenis mesin produksi mempunyai mikrokontroler yang tertanam yang terdiri atas mikroprosesor yang mengontrol dan memonitor operasi-operasi berdasarkan program-program dan data yang disimpan dalam ROM.

Pada semua sistem ini, operasi sistem bergantung pada sesuatu yang cukup besar apa yang disimpan dalam ROM, dan perubahan dalam operasi sistem biasanya dibuat dengan mengubah isi ROM. Pada EPROM, hal ini dilakukan dengan melepaskan chip dari sistem untuk menghapus dan memprogram ulang, atau mengganti chip dengan yang baru. Keharusan ini menunda operasi sistem kira-kira beberapa menit sampai satu jam.

Penggunaan EEPROM, memori flash sangat membantu karena mereka dapat dengan cepat dihapus dan diprogram ulang di dalam sistem.

7.7.2 Memori Bootstrap

Banyak mikrokomputer dan komputer-komputer besar pada umumnya tidak mempunyai sistem operasi sendiri yang tersimpan dalam ROM. Sebagai gantinya, program-program ini disimpan pada memori eksternal, biasanya cakram magnetik. Bagaimana kemudian pekerjaan komputer ini mengetahui apa yang harus dilakukan ketika catu daya mereka dihidupkan? Sebuah program yang relatif kecil yang disebut *program bootstrap* disimpan di dalam ROM. Ketika catu daya komputer dihidupkan, dia akan mengeksekusi instruksi yang ada dalam program bootstrap ini.

Kebanyakan chip-chip pemrosesan sinyal digital me-load memori program internalnya dari sebuah ROM bootstrap eksternal ketika catu dayanya dihidupkan. Beberapa menggunakan PLD (*programmable logic devices*).

7.7.3 Tabel-Tabel Data

ROM sering digunakan untuk menyimpan tabel-tabel data yang tidak berubah. Beberapa contoh tabel-tabel trigonometri (yaitu sinus, kosinus, dan lain-lain) dan tabel-tabel konversi kode. Sistem digital dapat menggunakan tabel-tabel ini untuk "mencari/look-up" nilai yang benar. Misalnya sebuah ROM dapat digunakan untuk menyimpan fungsi sinus untuk sudut-sudut dari 0° sampai 90° . Dia dapat dibuat sebagai 128×8 dengan tujuh masukan alamat dan delapan keluaran data. Masukan-masukan alamat merepresentasikan sudut dengan kenaikan kira-kira 0.7° . Misalnya, alamat 0000000 adalah 0° , alamat 0000001 adalah 0.7° , alamat 0000010 adalah 1.41° dan seterusnya hingga alamat 1111111 yang setara dengan 89.3° . Bila suatu alamat digunakan pada ROM, keluaran data akan merepresentasikan perkiraan sinus sudut. Misalnya, dengan alamat input 1000000 (merepresentasikan sekitar 45°) keluaran data akan 10110101. Karena sinus lebih kecil atau sama dengan 1, data diterjemahkan sebagai sebuah pecahan yaitu 0.10110101 yang jika dikonversikan ke desimal sama dengan 0.707 (sinus 45°).

ROM tabel *look-up* standar untuk fungsi-fungsi seperti ini pernah tersedia pada chip-chip TTL. Hanya beberapa yang masih diproduksi. Saat

ini kebanyakan sistem yang memerlukan look-up nilai-nilai ekivalen melibatkan mikroprosesor, dan data tabel "look-up" disimpan dalam ROM yang sama yang menyimpan instruksi-instruksi program.

7.7.4 Pengonversi Data

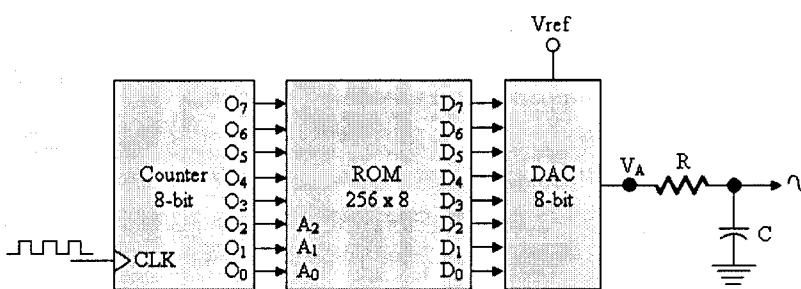
Sirkuit pengonversi data menerima data yang dinyatakan dalam satu jenis kode dan menghasilkan keluaran yang dinyatakan dalam jenis yang lain. Konversi kode diperlukan, misalnya bila suatu komputer mengeluarkan data dalam kode biner langsung dan kita akan mengubahnya menjadi BCD (*binary coded decimal*) supaya ditampilkan dalam pembacaan LED 7-semen.

Salah satu metode yang paling mudah pada pengubahan kode adalah menggunakan ROM terprogram agar aplikasi dari suatu alamat khusus (kode lama) menghasilkan keluaran data yang merepresentasikan ekivalen pada kode baru. ROM TTL 74185 menyimpan kode konversi biner-ke-BCD untuk masukan biner 6-bit. Untuk mengilustrasikannya, sebuah masukan alamat biner 100110 (38 desimal) akan menghasilkan keluaran data 00111000 yang merupakan kode BCD untuk 38 desimal.

7.7.5 Pembangkit Fungsi

Pembangkit fungsi adalah suatu sirkuit yang menghasilkan bentuk gelombang seperti gelombang sinus, gelombang gigi gergaji, gelombang segi tiga dan gelombang kotak. Gambar 7.16 menunjukkan bagaimana sebuah ROM tabel look-up dan sebuah DAC (*digital to analog converter*) digunakan untuk membangkitkan sebuah sinyal keluaran gelombang sinus.

ROM menyimpan nilai delapan-bit sebanyak 256, masing-masing sesuai dengan nilai bentuk gelombangnya. Pencacah 8-bit secara kontinu diberikan pulsa oleh sinyal clock untuk menghasilkan masukan-masukan alamat sekuensial ke ROM.



Gambar 7.16 Pembangkit fungsi/sinyal menggunakan sebuah ROM dan DAC

7.8 RAM SEMIKONDUKTOR

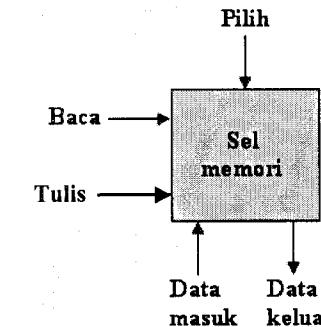
Sebutan RAM (*random access memory*) berarti lokasi alamat memori di mana saja dapat diakses semudah mengakses lokasi alamat lainnya dalam memori tersebut tanpa ada perbedaan waktu akses seperti yang terjadi pada memori sekuensial. Banyak jenis memori yang dapat diklasifikasikan sebagai memori akses acak, tetapi jika istilah RAM digunakan pada memori semikonduktor, maka umumnya berarti memori baca/tulis (RWM, *read/write memory*) sebagai lawan dari ROM. Dalam pembahasan kita selanjutnya istilah RAM digunakan untuk mewakili memori semikonduktor RWM.

Kelemahan utama RAM adalah karena data akan hilang jika catu dayanya mati, atau biasa disebut dengan istilah memori *volatile*. Namun demikian, beberapa RAM dengan teknologi CMOS menggunakan catu daya yang demikian kecil dalam mode siang (tidak ada aktivitas baca dan tulis) mereka dapat dicatuh dari baterai kapanpun catu daya utama terputus. Tentunya keuntungan utama RAM adalah karena dapat ditulisi dan dibaca dengan cepat sama mudahnya.

7.9 ARSITEKTUR SEMIKONDUKTOR

Gambar 7.17 memberikan ilustrasi sebuah diagram blok yang merepresentasikan sebuah sel memori semikonduktor RAM. Untuk melakukan operasi pembacaan atau penulisan, pertama sel memori dipilih oleh sinyal "pilih" kemudian diikuti dengan sinyal kontrol "baca". Isi sel (0 atau 1)

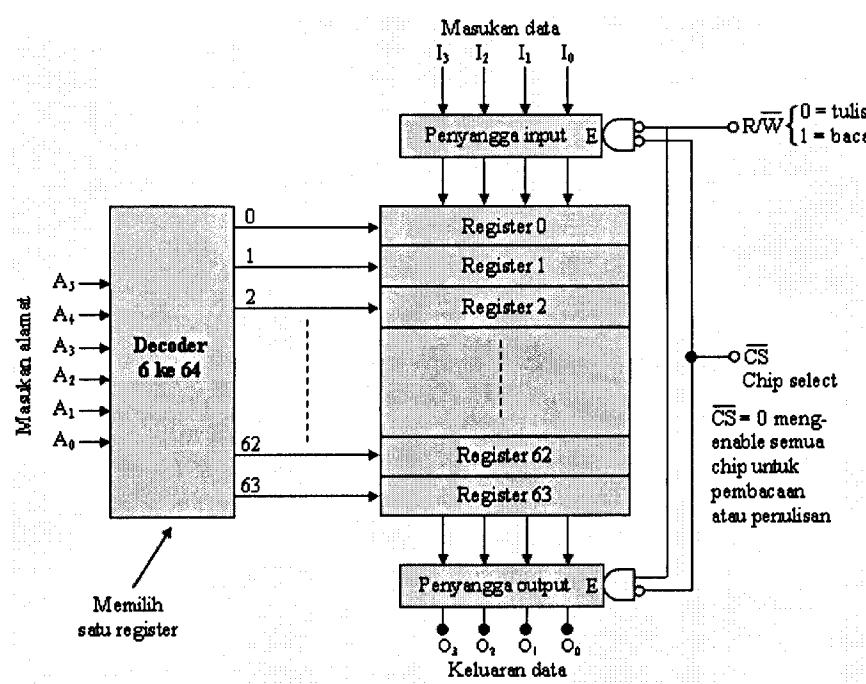
diberikan pada "data keluar" (D_o) setelah melewati waktu akses. Untuk penulisan, pertama sel dipilih oleh sinyal "pilih" kemudian diikuti dengan pengiriman data (0 atau 1) pada "data masuk" (D_i) dan mengaktifkan sinyal "tulis". Setelah itu barulah data tersimpan dalam sel.



Gambar 7.17 Sel memori semikonduktor

Ada dua jenis memori semikonduktor baca/tulis yaitu memori statik (*static random access memory*, SRAM) dan memori dinamik (*dynamic random access memory*, DRAM). Setiap sel di dalam SRAM terdapat sebuah flip-flop yang menyimpan 1 atau 0 selama ada suplai daya. Memori SRAM membutuhkan ruang lebih banyak daripada DRAM, tetapi lebih cepat. Setiap sel di dalam DRAM terdapat sebuah kapasitor yang menyimpan 1 atau 0 seperti cara penyimpanan muatan pada kapasitor, yang akan mengalami *discharge* (pembuangan muatan) selama periode waktu tertentu. Isi sel-sel DRAM biasanya hanya dapat bertahan sekitar 2 milidetik karena terjadi *discharge*. Karena itu selama 2 milidetik dilakukan penyegaran isi sel dengan cara memberikan muatan kembali (*recharge*) pada kapasitor. Hal ini disebut dengan penyegaran ulang (*refreshing*). Pada DRAM dibutuhkan sirkuit tambahan untuk melakukan penyegaran ulang secara periodik. Memori SRAM harganya lebih mahal daripada DRAM karena diperlukan sedikitnya enam buah transistor untuk membuat sebuah sel memori dan mempunyai kepadatan kemasan (jumlah bit dalam sebuah chip IC) yang rendah. Memori SRAM digunakan pada aplikasi yang membutuhkan memori yang cepat dan kecil seperti memori cache. Memori DRAM mempunyai kepadatan kemasan yang baik dan murah, karena itu digunakan untuk memori utama.

Seperti halnya arsitektur ROM, cukup membantu untuk memikirkan RAM yang terdiri atas sejumlah register yang masing-masing menyimpan word data tunggal, dan masing-masing mempunyai sebuah alamat unik. Kapasitas word dari RAM khas misalnya 1K, 4K, 8K, 16K, 64K, 128K, 256K dan 1024K, dan dengan ukuran word 1, 4, atau 8 bit. Seperti yang akan kita temui di depan, kapasitas word dan ukuran word dapat diperluas dengan mengombinasikan sejumlah chip memori.



Gambar 7.18 Organisasi internal RAM 64 x 4

Gambar 7.18 menunjukkan arsitektur RAM yang disederhanakan yang menyimpan 64 word, masing-masing word empat bit (yaikni memori 64 x 4). Word-word ini mempunyai rentang alamat dari 0 sampai 63 desimal. Untuk memilih salah satu dari 64 lokasi alamat untuk pembacaan atau penulisan, sebuah kode alamat biner dimasukkan pada sebuah sirkuit decoder. Karena $64 = 2^6$, maka decoder memerlukan kode masukan sebanyak 6 bit. Setiap kode alamat mengaktifkan satu keluaran decoder

yang pada gilirannya akan meng-enable register yang bersesuaian. Misalnya, anggaplah sebuah kode alamat yang diterapkan adalah:

$$A_5 A_4 A_3 A_2 A_1 A_0 = 011010$$

Karena $011010_2 = 26_{10}$, maka keluaran decoder 26 akan *high*, memilih register 26 untuk suatu operasi baca atau tulis.

7.9.1 Operasi Baca

Kode alamat memberikan satu register pada chip memori untuk pembacaan atau penulisan. Untuk membaca isi dari register yang terpilih, maka masukan baca/tulis (R/\overline{W}) harus *high* dan juga masukan CHIP SELECT (\overline{CS}) harus aktif (dalam hal ini *low*). Beberapa pabrik menggunakan simbol WE (write enable) atau \overline{W} yang fungsinya sama dengan R/\overline{W} . Kombinasi $R/\overline{W}=1$ (*high*) dan $\overline{CS}=0$ (*low*) meng-enable penyanga keluaran sehingga isi dari register terpilih akan berada pada keluaran data empat bit. $R/\overline{W}=1$ juga men-disable penyanga masukan sehingga masukan data tidak memengaruhi memori ketika melakukan operasi baca.

7.9.2 Operasi Tulis

Untuk menulisi sebuah word 4-bit yang baru ke dalam register yang terpilih memerlukan $R/\overline{W}=0$ dan $\overline{CS}=0$. Kombinasi ini meng-enable penyanga masukan sehingga word 4-bit yang diberikan ke masukan data akan di-load (baca) ke dalam register terpilih. $R/\overline{W}=0$ juga men-disable penyanga keluaran (*tristate*) sehingga keluaran data berada dalam keadaan Hi-Z selama operasi penulisan.

7.9.3 Pemilih Chip

Chip-chip memori pada umumnya mempunyai satu atau lebih masukan CS yang digunakan untuk meng-enable atau men-disable chip seluruhnya. Ketika mode disable, maka semua masukan data dan keluaran data dilumpuhkan (Hi-Z) sehingga operasi baca atau tulis tidak dapat dilakukan. Pada mode ini, isi memori tidak terpengaruh. Alasan untuk mempunyai masukan CS akan menjadi jelas bila kita menggabungkan chip-chip memori untuk memperoleh kapasitas memori yang lebih besar. Banyak pabrik menyebut masukan ini *CHIP ENABLE* (CE). Bila masukan CS atau CE berada dalam keadaan aktif, maka chip memori dikatakan *terpilih*; sebaliknya

dikatakan *tidak terpilih*. Banyak IC memori yang dirancang untuk digunakan pada daya yang sangat rendah jika *tidak terpilih*. Pada sistem-sistem memori yang besar, untuk pemberian operasi memori, satu atau lebih chip memori akan dipilih selama semua yang lainnya tidak terpilih.

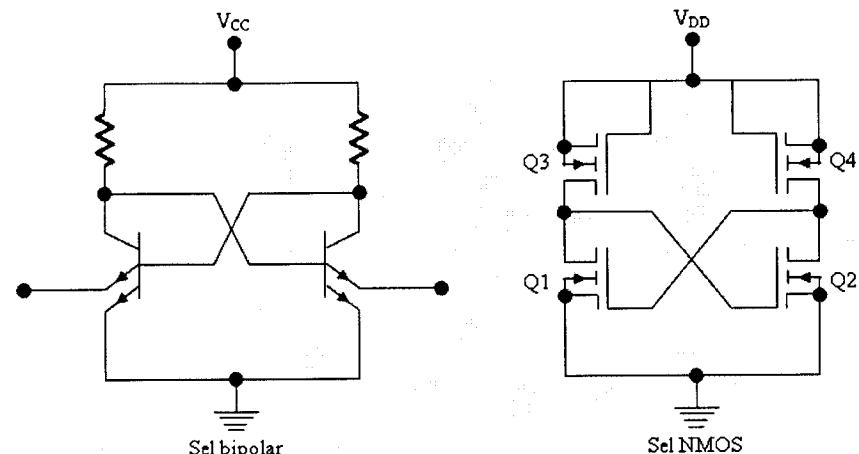
7.9.4 Penyemant Input/Output Bersama

Untuk menghemat penyemant (pin) pada suatu kemasan IC maka pabrik seringkali menggabungkan fungsi-fungsi masukan data dan keluaran data menggunakan penyemant-penyemant I/O bersama. Masukan R/\bar{W} mengontrol fungsi penyemant I/O ini. Ketika operasi pembacaan, penyemant-penyemant I/O bertindak sebagai keluaran data yang mereproduksi isi lokasi alamat yang terpilih. Ketika operasi penulisan, penyemant-penyemant I/O bertindak sebagai masukan data ke data yang akan ditulis.

Kita akan melihat mengapa hal ini dilakukan dengan mengacu pada Gambar 7.18. Dengan memisahkan penyemant-penyemant input dan output, dibutuhkan total 18 penyemant (termasuk ground dan suplai daya). Dengan empat penyemant I/O bersama, maka hanya dibutuhkan 14 penyemant. Penyemant yang dihemat menjadi lebih signifikan untuk chip-chip dengan ukuran word yang besar.

7.10 STATIC RAM (SRAM)

Operasi RAM yang telah kita bahas hingga di sini menggunakan *static RAM* (SRAM) yang dapat menyimpan data selama daya diberikan pada chip. Sel-sel memori SRAM pada dasarnya adalah sejumlah flip-flop yang akan tetap pada suatu keadaan yang diberikan (menyimpan bit) secara permanen, selama catu daya yang diberikan tidak terputus. Ke depan kita akan membahas *dynamic ram* (DRAM) yang menyimpan data sebagai muatan pada kapasitor. Dengan DRAM data yang tersimpan secara perlahan-lahan akan hilang karena adanya kehilangan muatan kapasitor, sehingga diperlukan penyegaran ulang secara periodik (mengisi kembali kapasitor).



Gambar 7.19 Sel-sel RAM-statik bipolar dan NMOS khas

Memori SRAM tersedia dalam teknologi bipolar, MOS, dan BiCMOS. Kebanyakan aplikasi-aplikasi menggunakan RAM NMOS atau CMOS. Teknologi bipolar mempunyai kelebihan dalam hal kecepatan (walaupun CMOS secara perlahan mulai mendekati kecepatan tersebut), dan teknologi MOS mempunyai kapasitas yang lebih besar dan konsumsi daya lebih rendah. Gambar 7.19 menunjukkan perbandingan sel memori statik bipolar khas dengan sel memori statik NMOS khas. Sel bipolar terdiri atas dua transistor bipolar dan dua resistor, sedangkan sel NMOS terdiri atas empat MOSFET kanal-N. Sel bipolar membutuhkan area chip yang lebih besar daripada sel MOS karena sebuah transistor bipolar lebih kompleks daripada sebuah MOSFET, dan karena sel bipolar membutuhkan resistor terpisah sedangkan sel MOS menggunakan MOSFET sebagai resistor (Q3 dan Q4). Sel memori CMOS mirip sel NMOS kecuali bahwa CMOS menggunakan MOSFET kanal-P yang ditempatkan pada Q3 dan Q4. Ini menghasilkan konsumsi daya yang lebih rendah tetapi menambah kompleksitas chip.

7.10.1 Pewaktuan SRAM

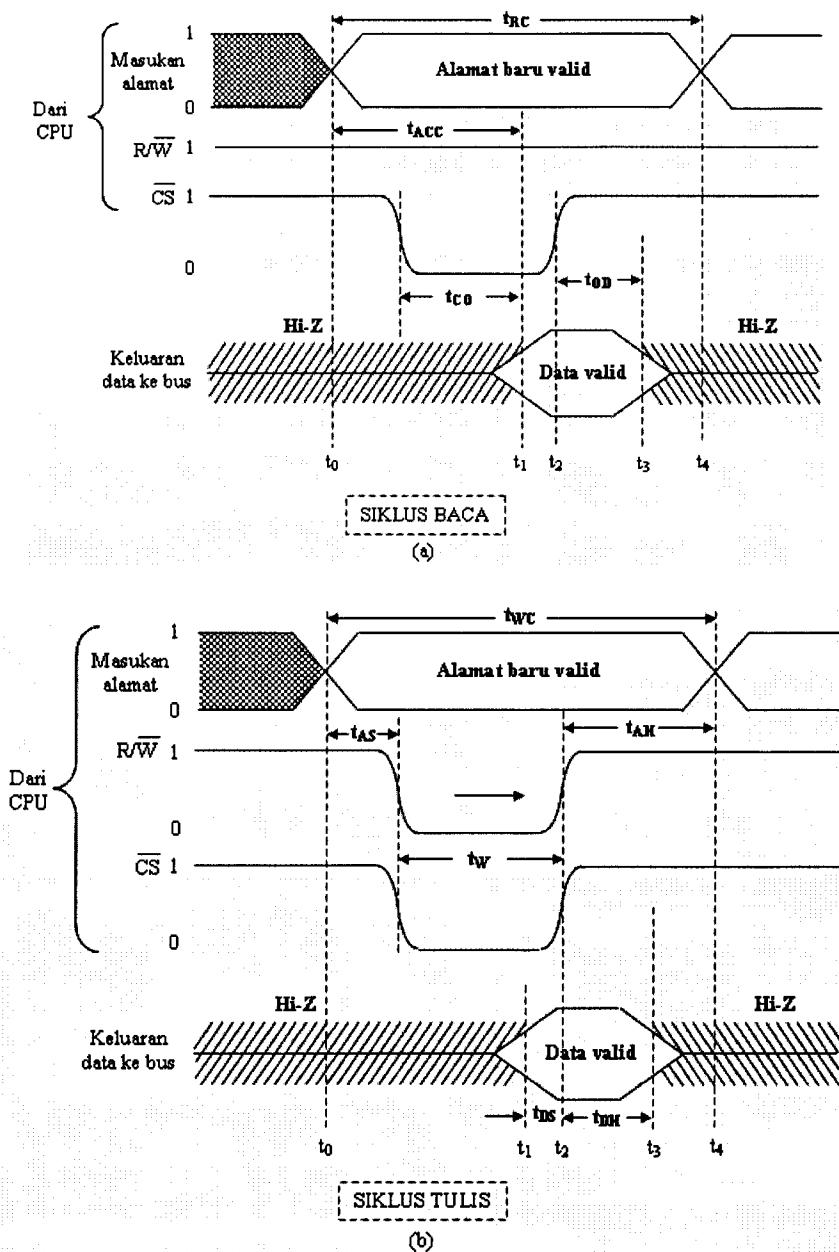
IC RAM paling sering digunakan sebagai memori internal suatu komputer. CPU melakukan operasi baca dan tulis secara kontinu pada memori ini dengan kecepatan yang sangat tinggi. Chip-chip memori yang diantar-mukakan ke CPU harus cukup cepat untuk merespons perintah-

perintah baca dan tulis CPU, dan seorang perancang komputer harus memperhatikan karakteristik pewaktuan (timing) berbagai RAM.

Tidak semua RAM mempunyai karakteristik pewaktuan yang sama, tetapi pada umumnya mirip, dan dengan demikian kita akan menggunakan set karakteristik khas untuk keperluan ilustratif. Penamaan parameter-parameter pewaktuan yang berbeda akan bervariasi dari satu pabrik ke pabrik lainnya, tetapi makna dari setiap parameter biasanya mudah ditentukan dari diagram-diagram pewaktuan memori pada lembar data (*data-sheet*) RAM. Gambar 7.20 menunjukkan diagram-diagram pewaktuan untuk suatu siklus baca secara lengkap dan suatu siklus tulis secara lengkap untuk sebuah chip RAM khas.

7.10.2 Siklus Baca

Bentuk gelombang pada Gambar 7.20(a) menunjukkan bagaimana alamat, R/\overline{W} , dan masukan pemilih chip berperilaku ketika suatu siklus baca memori. Sebagai catatan, CPU mensuplai sinyal-sinyal masukan ini ke RAM bila dia ingin membaca data dari suatu lokasi alamat RAM khusus. Walaupun sebuah RAM mempunyai banyak masukan-masukan alamat yang berasal dari bus alamat CPU, untuk kejelasan diagram hanya menunjukkan dua. Keluaran data RAM juga ditunjukkan; kita akan menganggap bahwa RAM khusus ini mempunyai satu keluaran data mengingat bahwa keluaran data RAM dihubungkan ke bus data CPU.



Gambar 7.20 Pewaktuan RAM-statik khas (a) siklus baca; (b) siklus tulis

Siklus baca dimulai pada saat t_0 . Sebelum waktu tersebut, masukan-masukan alamat bebas apa saja berada pada bus alamat dari operasi sebelumnya. Karena pemilih chip RAM tidak aktif, maka dia tidak akan merespons pada alamat lama. Perhatikan bahwa saluran R/\bar{W} adalah HIGH mendahului/sebelum t_0 dan tetap HIGH selama siklus baca. Pada kebanyakan sistem-sistem memori, R/\bar{W} umumnya dijaga dalam keadaan HIGH kecuali jika dia dikemudikan LOW ketika sebuah siklus tulis. Keluaran data RAM berada dalam keadaan Hi-Z karena $\bar{CS} = 1$.

Pada t_0 CPU menggunakan sebuah alamat baru ke masukan RAM, alamat ini merupakan lokasi yang akan dibaca. Setelah waktu berikutnya sinyal-sinyal alamat menjadi stabil, saluran \bar{CS} diaktifkan. RAM merespons dengan menempatkan data dari lokasi yang dialamat ke saluran keluaran data pada t_1 . Waktu antara t_0 dan t_1 adalah waktu akses RAM t_{ACC} dan merupakan waktu antara penggunaan alamat baru dengan munculnya data keluaran valid. Parameter pewaktuan t_{CO} adalah waktu yang digunakan oleh keluaran RAM untuk menuju dari Hi-Z ke level data sah (valid) ketika \bar{CS} diaktifkan.

Pada saat t_2 , \bar{CS} kembali HIGH, dan keluaran RAM kembali ke keadaan Hi-Z setelah suatu interval waktu t_{OD} . Jadi data RAM akan berada pada bus data antara t_1 dan t_3 . CPU dapat mengambil data dari bus data kapan pun ketika dalam interval tersebut. Kebanyakan komputer, CPU menggunakan tepi transisi positif sinyal \bar{CS} pada t_2 untuk menahan data ini ke salah satu register internal.

Waktu siklus baca t_{RC} rampung, diperluas dari t_0 ke t_4 , ketika CPU mengubah masukan-masukan alamat ke suatu alamat yang berbeda untuk siklus baca atau tulis berikutnya.

7.10.3 Siklus Tulis

Gambar 7.20(b) menunjukkan aktivitas sinyal untuk siklus tulis yang dimulai jika CPU mensuplai alamat baru ke RAM pada saat t_0 . CPU mengemudikan saluran R/\bar{W} dan \bar{CS} LOW setelah menunggu interval waktu t_{AS} , disebut *address setup time* (waktu setup alamat). Hal ini memberikan kesempatan decoder alamat RAM untuk merespons ke alamat baru. R/\bar{W} dan \bar{CS} dipertahankan LOW selama interval waktu t_w yang disebut interval waktu tulis.

Selama interval waktu tulis ini, pada saat t_1 CPU mensuplai data valid ke bus data untuk dituliskan ke dalam RAM. Data ini harus ditahan pada masukan RAM sedikitnya pada interval waktu t_{DS} sebelumnya, dan sedikitnya pada interval waktu t_{DH} sesudahnya, deaktivasi R/\bar{W} dan \bar{CS} pada t_2 . Interval t_{DS} disebut *data setup time* (waktu setup data) dan t_{DH} disebut *data hold time* (waktu menahan data). Dengan cara yang sama, masukan-masukan alamat harus tetap stabil selama interval *address hold time* (waktu menahan alamat), t_{AH} , setelah t_2 . Jika persyaratan *setup time* atau *hold time* ini tidak ditemukan, maka operasi tulis tidak akan dapat dijamin.

Waktu siklus tulis t_{WC} rampung, dari t_0 ke t_4 , ketika CPU mengubah saluran-saluran alamat ke sebuah alamat baru untuk siklus baca atau siklus tulis berikutnya.

Waktu siklus baca t_{RC} dan waktu siklus tulis t_{WC} merupakan hal yang mendasar dalam menentukan seberapa cepat suatu chip memori dapat bekerja. Misalnya, dalam aplikasi aktual, sebuah CPU akan sering membaca word-word data yang berurutan dari satu ke yang lainnya. Jika memori mempunyai t_{RC} 50 ns, maka CPU dapat membaca satu word setiap 50 ns, atau 20 juta word per detik; dengan $t_{RC} = 10$ ns, maka CPU dapat membaca 100 juta word per detik. Tabel 7.1 menunjukkan waktu siklus-baca minimum dan siklus tulis untuk beberapa chip-chip RAM statik yang representatif.

TABEL 7.1 Waktu siklus baca dan tulis (minimum) beberapa chip SRAM

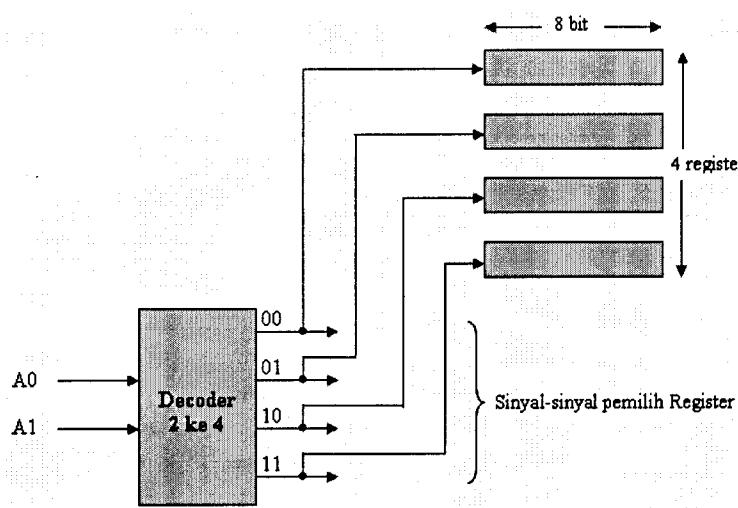
Divals	$t_{RC}(\text{min})$ (ns)	$t_{WC}(\text{min})$ (ns)
CMOS MCM6206C, 32K x 8	15	15
NMOS 2147H, 4K x 1	35	35
BiCMOS MCM6708A, 64K x 4	8	8

7.10.4 IC SRAM Aktual

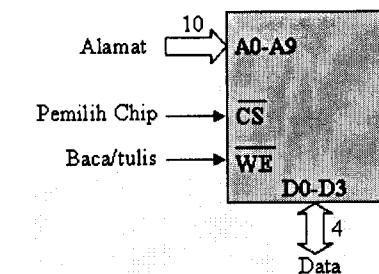
Gambar 7.21 menunjukkan memori dua dimensi menggunakan beberapa register. Jika jumlah lokasi bertambah, maka panjang decoder-alamat juga bertambah. Misalnya untuk membangun 1 KB memori, dibutuhkan 1024 register 8 bit. Dibutuhkan decoder 10-ke-1024 untuk mendekode 10

bit alamat, yang berarti diperlukan banyak gerbang multi-input. Karena itu pembuatan IC secara praktis dengan cara ini sulit. Solusi yang mungkin adalah menyusun elemen-elemen memori (sel) seperti matriks.

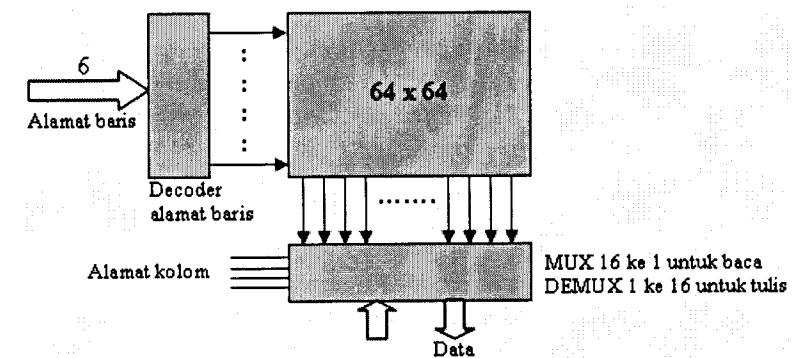
Pada Gambar 7.22 ditunjukkan diagram blok sebuah IC RAM 1K x 4. Memori ini berisi 4 bit pada setiap lokasi, karena itu dibutuhkan 10 bit untuk pengalaman chip memori ini dan kapasitasnya 4K bit. Operasi baca atau tulis hanya mungkin jika masukan CS aktif (*low*). Kepastian operasi ditentukan oleh penyematan WE (*write enable*). Ketika WE *low*, maka terjadi operasi penulisan IC. Memori mendapatkan data (4 bit) dari penyematan data dan menyimpannya dalam lokasi yang diberikan oleh bit alamat. Bila masukan WE *high*, IC melakukan operasi pembacaan lokasi dan mensuplai isi dari penyematan data. Organisasi internal SRAM ditunjukkan pada Gambar 7.22(b). Secara internal sel memori disusun sebagai matriks 64 x 64. Enam bit alamat didekode oleh decoder alamat baris untuk memilih baris. Sel-sel kolom disusun sebagai 16 set di mana setiap set mempunyai empat sel seperti yang ditunjukkan pada Gambar 7.22(c). Karena itu selama operasi pembacaan, sebuah MUX 4 bit 16-ke-1 digunakan untuk memilih set dengan menggunakan 4 bit sebagai masukan kontrol *select* ke multiplexer. Selama operasi tulis, demultiplexer melakukan routing masukan data ke set yang sesuai pada empat bit alamat.



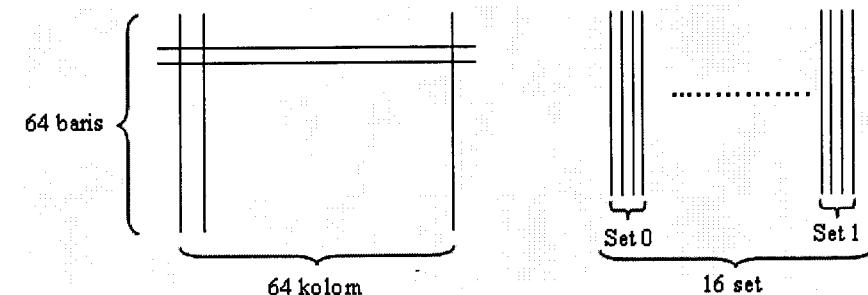
Gambar 7.21 Memori 4 x 8 menggunakan 4 register



Gambar 7.22 (a) IC SRAM 1K x 4



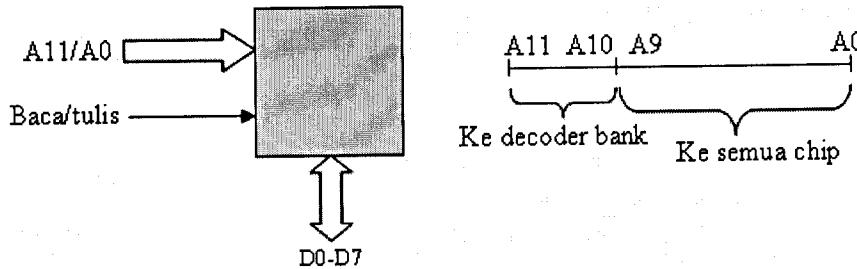
Gambar 7.22(b) Organisasi internal SRAM 2114



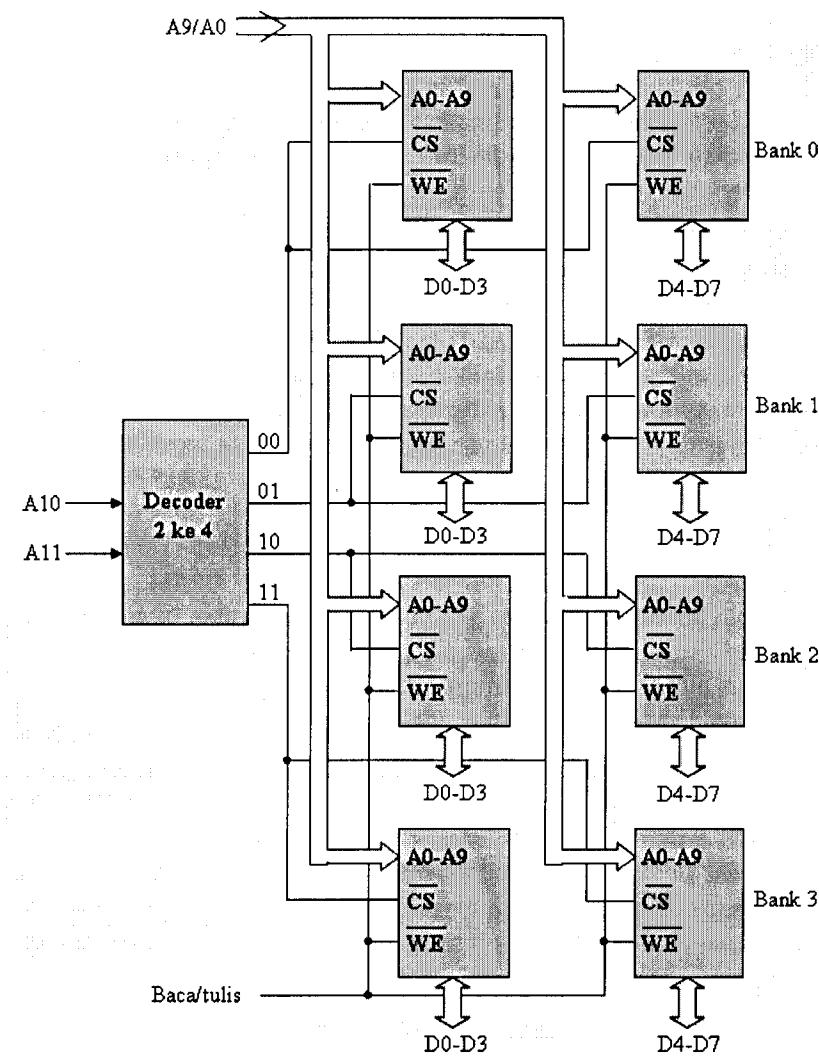
Gambar 7.22 (c) Organisasi matriks SRAM 2114

7.10.5 Desain Modul-Modul SRAM

Dengan menggunakan IC SRAM dari jenis yang sama, maka dapat dirancang sebuah modul memori SRAM kapasitas besar. Pada Gambar 7.23 ditunjukkan konstruksi RAM 4 KB dengan menggunakan IC SRAM 1K x 4. Setiap bank mempunyai dua IC yang secara bersama membentuk 1 KB. Masukan CS dari dua IC pada bank dihubungkan bersama. Empat bank secara bersama memberikan 4 KB. Pemilihan bank didasarkan pada pendekodean dari dua MSB (A10 dan A11) pada 12 bit alamat. Keluaran decoder dihubungkan ke masukan CS pada suatu bank. Masukan WE pada keseluruhan delapan IC dihubungkan bersama dan dikendalikan oleh sinyal *READ/WRITE*. Penyemat alamat A9-A0 dari keseluruhan delapan chip dihubungkan bersama. IC dibagi ke dalam dua grup pada penyemat data. Satu chip pada setiap bank menyediakan empat bit (setengah) dari sebuah byte. Data D0-D3 dibangun dari empat chip dan D4-D7 dibangun dari empat chip sisanya. Ketika CPU mentransmisikan 12 bit alamat (A11-A0), maka A11 dan A10 digunakan untuk memilih hanya satu bank sementara A9-A0 untuk memilih lokasi pada bank.



Gambar 7.23 (a) Simbol SRAM 4 KB



Gambar 7.23(b) Skema logika untuk SRAM 4 KB

7.11 DYNAMIC RAM (DRAM)

RAM dinamik dipabrikasi menggunakan teknologi MOS dan untuk kapasitas besar, kebutuhan daya rendah, dan kecepatan operasi menengah.

Telah disebutkan sebelumnya bahwa tidak seperti RAM statik yang menyimpan informasi dalam flip-flop, RAM dinamik menyimpan 1 dan 0 sebagai muatan pada sebuah kapasitor MOS kecil (khas beberapa picofarad). Karena kecenderungan muatan-muatan ini bocor setelah periode waktu tertentu, RAM dinamik memerlukan pengisian muatan ulang secara periodik terhadap sel-sel memori, hal ini disebut *refreshing* (penyegaran ulang) RAM dinamik. Pada chip DRAM modern, setiap sel memori harus disegarkan ulang khas setiap 2, 4, atau 8 ms, jika tidak data tersebut akan hilang.

Kebutuhan untuk penyegaran ulang merupakan suatu kelemahan RAM dinamik dibandingkan RAM statik karena memerlukan sirkuit pendukung tambahan eksternal. Beberapa chip DRAM mempunyai sirkuit kontrol refresh yang terpadu (*built-in*) yang tidak memerlukan hardware eksternal tambahan, tetapi memerlukan pewaktuan khusus sinyal-sinyal kontrol masukan chip. Lagi pula, seperti yang akan kita lihat, masukan-masukan alamat ke DRAM harus ditangani terus-menerus dibandingkan SRAM. Jadi secara umum, perancangan dengan menggunakan DRAM dalam suatu sistem lebih kompleks daripada SRAM. Namun, kapasitas DRAM yang lebih besar dan konsumsi dayanya yang lebih rendah membuat DRAM merupakan pilihan dalam memori sistem di mana pertimbangan-pertimbangan desain yang paling penting adalah menjaga ukuran, harga dan daya.

Untuk aplikasi di mana kecepatan dan pengurangan kompleksitas lebih kritis daripada harga, ruang, dan pertimbangan daya, maka RAM statik tetap lebih baik. RAM statik secara umum lebih cepat daripada RAM dinamik dan tidak memerlukan operasi penyegaran ulang. Mereka lebih sederhana untuk desain, tetapi mereka tidak dapat bersaing dengan RAM dinamik dalam hal kapasitas lebih tinggi dan keperluan daya lebih rendah.

Karena struktur sel DRAM lebih sederhana, maka DRAM khas mempunyai kepadatan empat kali dari SRAM. Dengan kenaikan kepadatan ini maka memungkinkan kapasitas memori empat kali lebih banyak dipasang pada suatu board tunggal, atau dengan kata lain hanya memerlukan seperempat ruang pada board untuk jumlah memori yang sama. Harga per bit memori RAM dinamik secara khas seperlima terhadap seperempat RAM statik.

Aplikasi utama SRAM adalah pada daerah di mana kebutuhan memori yang jumlahnya kecil atau di mana dibutuhkan kecepatan tinggi. Pada

umumnya instrumen-instrumen berbasis mikroprosesor dan peralatan-peralatan rumah tangga memerlukan kapasitas memori yang sangat kecil. Beberapa instrumen seperti osiloskop *digital storage* dan *logic analyzer* memerlukan memori kecepatan sangat tinggi. Untuk aplikasi-aplikasi seperti ini maka umumnya digunakan SRAM.

Memori utama (internal) pada kebanyakan mikrokomputer pribadi (misalnya PC berbasis Windows atau Macs) menggunakan DRAM karena kapasitasnya yang besar dan konsumsi dayanya rendah. Namun, komputer-komputer ini beberapa di antaranya kadang-kadang menggunakan SRAM dalam jumlah kecil untuk fungsi-fungsi yang mensyaratkan kecepatan maksimum seperti *video graphic*, *look-up table* dan memori cache. Namun DRAM kurang andal dibandingkan SRAM dan karena itu mekanisme pendektsian error seperti *parity logic* sangat perlu untuk modul DRAM. Dibutuhkan chip memori tambahan untuk penyimpanan bit paritas.

Dalam pemilihan jenis RAM secara khusus untuk suatu sistem, perancang mempunyai beberapa keputusan yang sulit. Kapasitas (sebesar mungkin), kecepatan (secepat mungkin), kebutuhan daya (sekecil mungkin), harga (semurah mungkin), dan kebaikan (semudah mungkin untuk perubahan) semuanya harus dijaga dalam kelayakan yang seimbang karena tidak ada jenis RAM yang dapat meminimalkan semua fitur-fitur yang diinginkan tersebut. Pasar RAM semikonduktor secara tetap mencoba untuk menghasilkan karakteristik-karakteristik gabungan yang ideal dalam produknya untuk berbagai kebutuhan atau aplikasi-aplikasi.

7.11.1 Penyegaran Ulang DRAM

Isi setiap bit sel dalam chip DRAM harus disegarkan sekali setiap beberapa millisecond (1,2 ms atau 4 ms). Dibutuhkan suatu mekanisme terpisah untuk melakukan penyegaran ulang memori.

Secara teoretis, pada operasi penyegaran ulang, isi tiap lokasi dibaca dari DRAM dan menulisnya kembali. Hal ini merupakan cara lama. Pada chip ini, penyegaran ulang menggunakan satu operasi baca dan satu operasi tulis pada setiap lokasi. Chip DRAM saat ini telah mempunyai kemampuan penyegaran yang terpadu. Pada chip ini tidak diperlukan membaca atau menulis data kembali. Sebagai gantinya, cukup memberi alamat lokasi dan masukan sinyal refresh secara periodik. Pembacaan dan penulisan kembali dilakukan secara internal oleh IC DRAM. Selama operasi penyegaran ulang,

penyemat DATA pada memori tidak digunakan dan penyemat ini dalam keadaan Hi-Z.

7.11.2 Penyegaran Ulang Baris

Karena IC DRAM secara internal mempunyai baris dan kolom, penyegaran ulang dapat dilakukan pada baris. Apabila satu baris disegarkan, semua bit sel di dalam baris tersebut tersegarkan. Jadi jumlah siklus penyegaran yang dibutuhkan untuk semua bit sel sama dengan jumlah baris. Selama setiap siklus, alamat baris disuplai ke IC DRAM. Umumnya, skema penyegaran ulang dibagi dua kategori:

1. *Burst refresh*
2. *Distributed refresh*

Pada skema *burst refresh*, semua baris disegarkan satu per satu dalam satu bidang tanpa terputus. Hal ini diulangi setiap 2 ms (atau 4 ms). Pada skema *distributed refresh*, penyegaran baris dilakukan antara operasi baca dan tulis.

Contoh 7.9

Sebuah IC DRAM 4416 (16K x 4) mempunyai waktu akses 150 ns. Pada organisasi internalnya terdapat 256 baris dan 256 kolom. Spesifikasi memory refresh 256 cycle/4 ms. Hitung refresh overhead.

Solusi:

Dalam 4 ms, terjadi 256 siklus refresh

Karena waktu akses 150 ns, penyegaran 256 baris membutuhkan waktu $256 \times 150 \text{ ns} = 38,4 \mu\text{s}$.

Waktu ini digunakan di dalam penyegaran setiap durasi 4 ms.

Karena itu refresh overhead = Waktu yang digunakan untuk penyegaran semua baris sekaligus: interval penyegaran

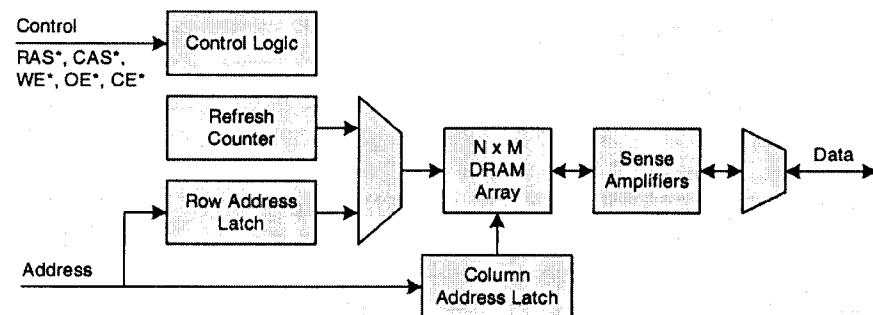
$$= (38,4 \times 10^{-6}) : (4 \times 10^{-3}) = 9,6 \times 10^{-3} \times 100\% = 0,96\%$$

refresh overhead sekitar = 1 %

7.11.3 Multiplexing Alamat Baris dan Kolom

DRAM diimplementasikan sebagai sebuah matriks bit-bit (array) dengan baris dan kolom seperti yang ditunjukkan pada Gambar 7.24. Tidak seperti SRAM, EPROM dan memori flash, DRAM secara fungsional dari perspektif luar digambarkan sebagai organisasi baris dan kolom.

SRAM diakses dengan memberikan alamat lengkap secara simultan. Alamat DRAM diberikan dalam dua bagian: alamat baris dan alamat kolom. Alamat baris dan kolom di-multiplex pada pin address yang sama untuk mengurangi ukuran kemasan dan harga. Pertama alamat baris di-load atau di-strobe ke dalam latch (penyimpan sementara), alamat baris melalui *row address strobe* atau RAS*, kemudian diikuti alamat kolom dengan *column address strobe*, atau CAS*. Data yang dibaca disalurkan ke output setelah waktu akses tertentu. Penulisan data diberikan pada waktu yang sama dengan alamat kolom, karena *column strobe* sebenarnya adalah trigger transaksi, apakah pembacaan atau penulisan.. Pada saat fase alamat kolom, maka WE dan CE akan berpengaruh.



Gambar 7.24 Arsitektur DRAM

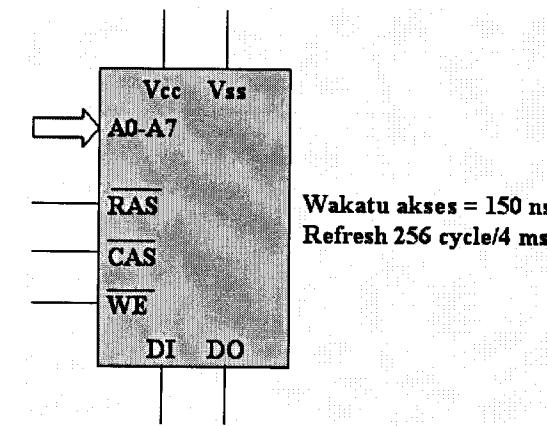
Sense amplifier pada chip diperlukan untuk mendeteksi muatan-muatan sementara yang digenggam dalam kapasitor DRAM. Amplifier ini digunakan juga untuk membantu dalam operasi penyegaran ulang. Sebuah *memory control* bertanggung jawab menjaga refresh timer dan memulai operasi refresh dengan frekuensi yang tepat agar data terjamin tidak hilang. Semua baris di-refresh dalam waktu yang sama. Sebuah refresh counter internal melakukan increment setelah setiap operasi refresh agar semua

baris (semua bit) akan mendapat giliran. Bila refresh dimulai, maka refresh counter akan meng-enable satu baris memori khusus. Isi baris dideteksi oleh sense amplifier dan dikembalikan lagi ke dalam array untuk recharge semua kapasitor di dalam baris. DRAM modern secara khas memerlukan refresh lengkap setiap 30 sampai 64 ms. Sebuah DRAM 64 Mb disusun sebagai 8.388.608 word x 8 (8 MB) dengan ukuran array/matriks internal 4.096 x 2.048 byte yang memerlukan 4.096 siklus refresh setiap 64 ms.

Kombinasi skema pengalamatan baris dan kolom plus array memori yang besar dengan sense amplifier yang kompleks dan decode logic membuat DRAM secara signifikan lebih lambat dari SRAM.

Teknik standar yang diikuti oleh pabrik IC DRAM dalam melakukan pengurangan jumlah penyemant (efeknya pada ukuran IC) adalah multiplexing alamat baris dan kolom pada salah satu kumpulan penyemant. Gambar 7.25 mengilustrasikannya untuk IC 4164, IC DRAM 64K bit dengan organisasi 64K x 1. Memori IC 4164 mempunyai sel-sel memori dalam sebuah array/matriks 256 x 256. Ada dua sel matriks yang sama dalam sebuah IC. Efektifnya, kelihatannya seperti sebuah matriks dengan 256 baris dan 256 kolom. Ada alamat yang diberikan pada chip memori yang selama pembacaan atau penulisan mengidentifikasi sel tertentu di dalam matriks. Chip memori mempunyai decoder untuk alamat baris dan alamat kolom. Karena itu ada 64K lokasi, diperlukan 16 bit untuk pengalamatan. 16 bit alamat mengandung 8 bit alamat baris dan 8 alamat kolom. Seperti yang terlihat pada Gambar 7.25, hanya ada 8 penyemant yang tampak menggantikan 16 penyemant alamat. Penyemant yang sama digunakan untuk pemberian alamat baris dan alamat kolom. Sinyal RAS dan CAS berturut-turut digunakan oleh chip untuk menahan (latch) alamat baris dan alamat kolom. Bila alamat baris diberikan ke chip, masukan RAS dibuat aktif (low). Berikutnya, bila alamat kolom dikirim, masukan CAS yang dibuat aktif. Secara internal IC DRAM mempunyai latch alamat baris dan latch alamat kolom. Bila sinyal RAS beralih dari high ke low, chip akan menahan masukan pada penyemant alamat ke dalam latch alamat baris. Bila sinyal CAS beralih dari high ke low, chip akan menahan masukan pada penyemant alamat ke dalam latch alamat kolom. Operasi baca atau tulis ditunjukkan pada chip oleh sinyal pada penyemant masukan WE. Untuk operasi tulis, penyemant WE dibuat low dan operasi baca dibuat high. Urutan operasi untuk proses baca/tulis adalah sebagai berikut:

1. Gunakan alamat baris
2. Aktifkan RAS
3. Gunakan bit data pada keadaan operasi tulis
4. Buat WE low untuk penulisan dan high untuk pembacaan
5. Gunakan alamat kolom
6. Aktifkan CAS
7. Terima bit data pada keadaan operasi baca



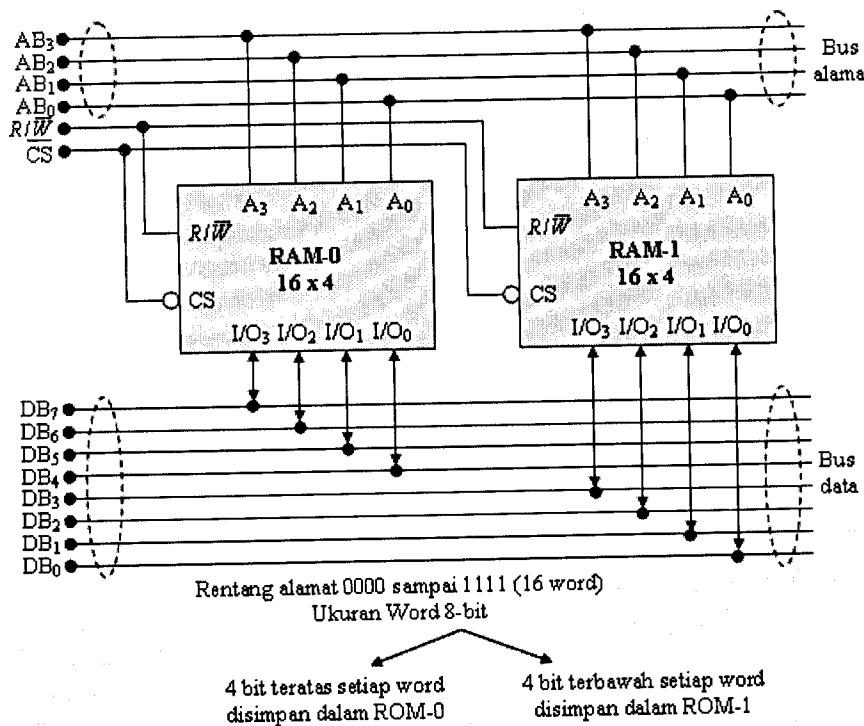
Gambar 7.25 Konfigurasi fungsi penyemant IC DRAM 4164

7.12 MEMPERBESAR UKURAN WORD DAN KAPASITAS

Dalam banyak aplikasi, kapasitas memori RAM atau ROM yang dibutuhkan atau ukuran word tidak dapat dicapai dengan hanya satu chip, beberapa chip memori digabungkan untuk mendapatkan kapasitas dan/atau ukuran word. Kita akan melihat bagaimana hal ini dilakukan melalui beberapa contoh yang mengilustrasikan gagasan-gagasan penting yang digunakan bila chip-chip memori diantar-mukakan ke sebuah mikroprosesor. Contoh-contoh berikut dimaksudkan untuk menjadi pelajaran yang baik, dan ukuran-ukuran chip memori yang digunakan untuk menghemat ruang.

7.12.1 Memperbesar Ukuran Word

Anggap kita memerlukan sebuah memori yang dapat menyimpan word 16×8 bit dan kita mempunyai chip-chip RAM yang semuanya disusun sebagai memori 16×4 dengan saluran I/O bersama. Kita menggabungkan dua chip 16×4 ini untuk menghasilkan memori yang kita inginkan. Konfigurasi untuk melakukan hal ini ditunjukkan pada Gambar 7.26. Periksa diagram ini secara seksama.



Gambar 7.26 Gabungan dua RAM 16×4 yang membentuk modul 16×8

Karena setiap chip dapat menyimpan word 16×4 bit dan kita akan menyimpan word 16×8 bit, maka kita menggunakan setiap chip untuk menyimpan setengah dari masing-masing word. Dengan kata lain, RAM-0 menyimpan empat bit bagian atas dari setiap 16 word, dan RAM-1 menyimpan empat bit bagian bawah dari setiap 16 word.

menyimpan empat bit bagian bawah dari setiap 16 word. Word delapan-bit penuh tersedia pada keluaran RAM yang dihubungkan ke bus data.

Salah satu dari 16 word dipilih dengan menggunakan kode alamat yang tepat ke bus alamat empat-saluran (AB_3, AB_2, AB_1, AB_0). Saluran alamat secara khas berasal pada CPU. Perhatikan bahwa setiap saluran bus alamat dihubungkan ke masukan alamat yang bersesuaian pada setiap chip. Artinya bahwa ketika suatu kode alamat ditempatkan pada bus alamat, kode alamat ini sama dengan yang diberikan pada chip-chip agar lokasi yang sama dalam setiap chip diakses pada saat yang sama.

Ketika alamat terpilih, kita dapat membaca atau menulis pada alamat ini di bawah kontrol saluran R/W dan \overline{CS} . Untuk membaca, R/W harus HIGH dan \overline{CS} harus LOW, hal ini menyebabkan saluran I/O RAM bertindak sebagai keluaran. RAM-0 menempatkan word empat-bit yang terpilih pada empat saluran bus data bagian atas dan RAM-1 menempatkan word empat-bit yang terpilih pada empat saluran bus data bagian bawah. Bus data kemudian berisi word delapan-bit penuh, yang sekarang dapat ditransmisikan ke sejumlah perangkat lain (biasanya pada sebuah register dalam CPU).

Untuk menulis, R/W harus LOW dan \overline{CS} harus LOW, yang menyebabkan saluran I/O RAM bertindak sebagai masukan. Word delapan-bit yang akan ditulis ditempatkan pada bus data (biasanya oleh CPU). Empat-bit bagian atas akan ditulisi ke dalam lokasi RAM-0 yang terpilih, dan empat-bit bagian bawah akan ditulisi ke dalam RAM-1.

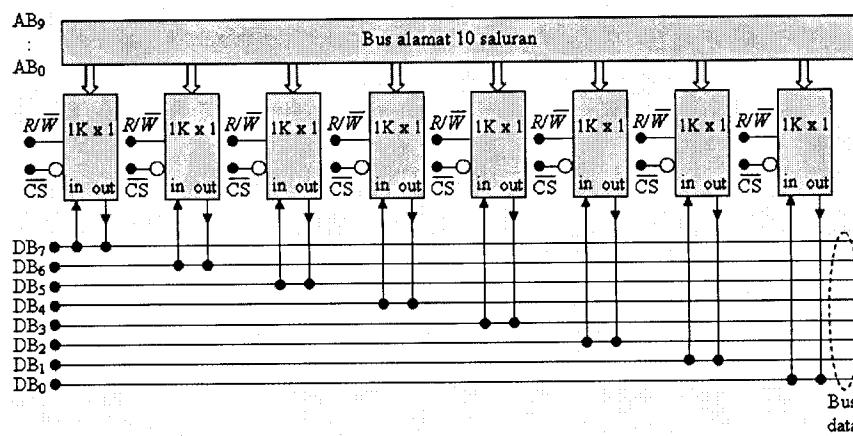
Pada dasarnya gabungan dua chip RAM bertindak sebagai sebuah chip memori tunggal 16×8 . Kita akan merujuk gabungan ini sebagai modul memori 16×8 . Gagasan dasar yang sama untuk memperluas ukuran word akan berlaku untuk banyak keadaan.

Contoh 7.10

Sebuah IC RAM statik jenis 2125A mempunyai kapasitas $1K \times 1$, satu masukan chip select aktif-LOW, dan masukan dan keluaran data yang terpisah. Tunjukkan bagaimana menggabungkan beberapa IC 2125A untuk membentuk modul memori $1K \times 8$.

Solusi:

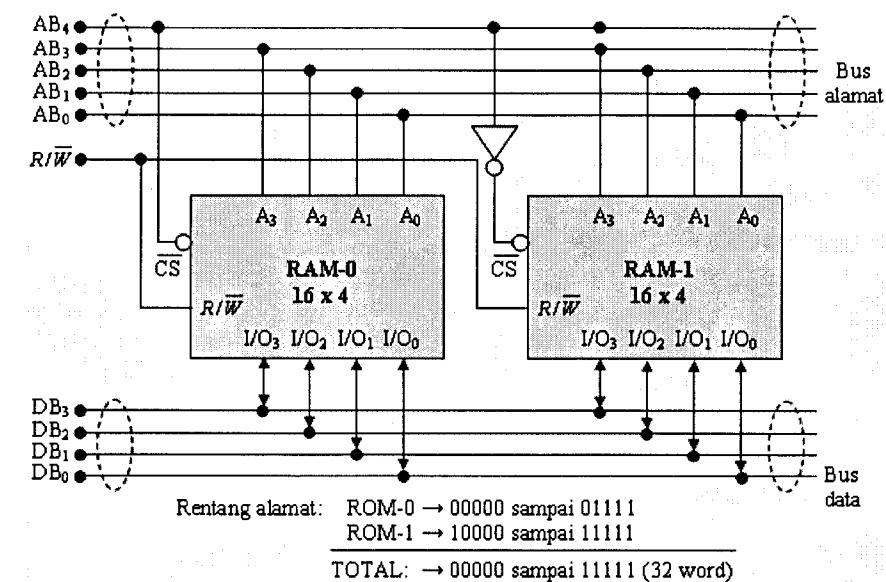
Susunan memori ditunjukkan dalam Gambar 7.27, di mana delapan chip 2125A digunakan untuk membentuk modul $1K \times 8$. Setiap chip menyimpan salah satu bit dari masing-masing 1024 word delapan-bit. Perhatikan bahwa semua masukan R/W dan \overline{CS} diikat bersama, dan 10-saluran bus alamat dihubungkan ke masukan-masukan alamat dari setiap chip. Dan juga karena 2125A mempunyai penyemat-penyemat data in dan data out yang terpisah, kedua penyemat tersebut dari masing-masing chip diikat ke saluran bus data yang sama.



Gambar 7.27 Delapan chip 2125A $1K \times 1$ yang disusun sebagai memori $1K \times 8$

7.12.2 Memperbesar Kapasitas

Anggaplah kita membutuhkan memori yang dapat menyimpan 32 word empat-bit dan semua chip yang kita miliki adalah chip 16×4 . Dengan menggabungkan dua 16×4 seperti yang ditunjukkan pada Gambar 7.28, kita dapat menghasilkan memori yang diinginkan. Sekali lagi, periksa diagram ini dan lihat apa yang Anda dapat tentukan dari gambar tersebut sebelum membacanya.



Gambar 7.28 Gabungan dua chip 16×4 yang membentuk modul memori 32×4

Setiap RAM digunakan untuk menyimpan 16 word empat-bit. Penyemat-penyemat I/O dari setiap RAM dihubungkan ke bus data empat-bit secara bersama. Hanya satu dari chip-chip RAM dapat dipilih (enable) pada satu waktu sehingga tidak ada masalah pada pertentangan bus. Hal ini dijamin dengan pengemudian masukan-masukan \overline{CS} dari sinyal-sinyal logika berbeda.

Karena kapasitas total modul memori ini adalah 32×4 , maka harus ada 32 alamat-alamat berbeda. Berarti hal ini membutuhkan lima saluran bus alamat. Saluran alamat atas AB_4 digunakan untuk memilih satu RAM atau yang lainnya (via masukan-masukan \overline{CS}) sebagai satu dari yang akan dibaca atau ditulisi. Empat saluran alamat lainnya AB_0 hingga AB_3 digunakan untuk memilih satu lokasi memori dari 16 dari chip RAM yang terpilih.

Sebagai ilustrasi, bila $AB_4 = 0$, maka \overline{CS} RAM-0 meng-enable chip ini untuk membaca atau menulis. Kemudian sembarang lokasi alamat dalam RAM-0 dapat diakses oleh AB_3 hingga AB_0 . Empat saluran alamat terakhir

dapat berada dalam rentang 0000 sampai 1111 untuk memilih lokasi yang diinginkan. Jadi, rentang alamat yang mewakili lokasi dalam RAM-0 adalah

$$AB_4 AB_3 AB_2 AB_1 AB_0 = 00000 \text{ sampai } 01111$$

Perhatikan bila $AB_0 = 0$, maka \overline{CS} RAM-1 adalah high, sehingga saluran I/O di-disable (Hi-Z) dan tidak dapat berkomunikasi dengan (memberi data ke atau menerima data dari) bus data.

Hal itu akan jelas bahwa bila $AB_4 = 1$, fungsi RAM-0 dan RAM-1 berlawanan. RAM-1 sekarang di-enable, dan saluran AB_3 hingga AB_0 memilih salah satu lokasinya. Jadi rentang alamat yang berada dalam RAM-1 adalah

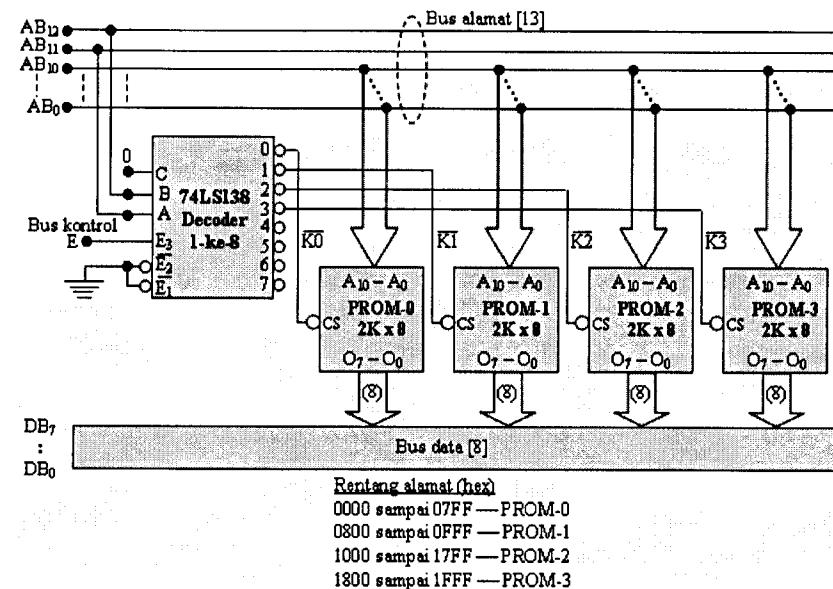
$$AB_4 AB_3 AB_2 AB_1 AB_0 = 10000 \text{ sampai } 11111$$

Contoh 7.11

Diinginkan membentuk gabungan beberapa PROM $2K \times 8$ untuk menghasilkan kapasitas total $8K \times 8$. Berapa banyak chip PROM yang dibutuhkan? Berapa banyak saluran bus alamat yang dibutuhkan?

Solusi:

Dibutuhkan empat chip PROM, dengan masing-masing menyimpan $2K$ dari $8K$ word. Karena $8K = 8 \times 1024 = 8192 = 2^{13}$, maka dibutuhkan saluran alamat tigabelas.



Gambar 7.29 Empat buah PROM $2K \times 8$ disusun menjadi kapasitas total $8K \times 8$

Konfigurasi memori pada Contoh 7.11 mirip dengan memori $32K \times 4$ Gambar 7.28. Namun, sedikit lebih kompleks karena memerlukan sirkuit decoder untuk pembangkitan sinyal masukan \overline{CS} . Diagram lengkap untuk memori 8192×8 ini ditunjukkan pada Gambar 7.29.

Karena kapasitas total adalah 8192 word, maka dibutuhkan 13 saluran bus alamat. Dua saluran urutan tertinggi AB_{11} dan AB_{12} digunakan untuk memilih salah satu chip PROM, sebelas saluran bus alamat lainnya menuju ke setiap PROM untuk memilih lokasi yang diinginkan dalam PROM yang terpilih. Pemilihan PROM diselesaikan dengan mengumpulkan AB_{11} dan AB_{12} ke dalam decoder 74LS138. Empat kombinasi yang mungkin didekodekan untuk membangkitkan sinyal aktif-LOW yang dikenakan pada masukan-masukan \overline{CS} . Misalnya, jika $AB_{11} = AB_{12} = 0$, keluaran K0 dari decoder menjadi LOW (yang lainnya HIGH) dan meng-enable PROM-0. Hal ini menyebabkan PROM-0 membangkitkan word data yang tersimpan secara internal pada alamat yang ditentukan oleh AB_0 sampai AB_{10} . Semua PROM lainnya disable.

Selama $AB_{11} = AB_{12} = 0$, nilai AB_{10} sampai AB_0 dapat berada dalam rentang semua 0 atau semua 1. Jadi, PROM-0 akan merespons ke rentang berikut dari 13-bit alamat:

$$AB_{12} - AB_0 = 00000000000000 \text{ sampai } 001111111111$$

Untuk praktisnya, alamat ini dapat dinyatakan lebih mudah dalam kode heksadesimal sebagai 0000 sampai 07FF.

Dengan cara yang sama, bila $AB_{12} = 0$ dan $AB_{11} = 1$, decoder memilih PROM-1, yang kemudian merespons dengan mengeluarkan word data yang tersimpan secara internal pada alamat AB_{10} sampai AB_0 . Jadi, PROM-1 merespons ke rentang berikut dari 13-bit alamat:

$$AB_{12} - AB_0 = 01000000000000 \text{ sampai } 011111111111 = 0800 \text{ sampai } 0FFF \text{ (hex)}$$

Kita harus memeriksa rentang alamat PROM-2 dan PROM-3 yang diberikan pada Gambar 7.28.

Jadi jelaslah bahwa saluran alamat AB_{12} dan AB_{11} digunakan untuk memilih salah satu dari empat buah chip PROM, sementara AB_{10} sampai AB_0 memilih word yang tersimpan dalam PROM yang terpilih.

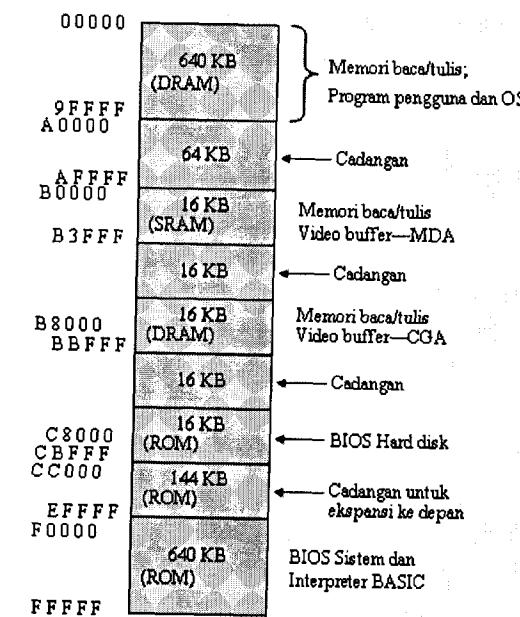
7.13 ALOKASI MEMORI UTAMA

Ketika arsitektur komputer telah rampung, salah satu pekerjaan perancangan yang paling penting adalah pemanfaatan ruang memori utama. Keseluruhan ruang memori utama tidak dapat diperuntukkan bagi penggunaan program saja karena alasan-alasan berikut:

1. Beberapa ruang memori diperlukan untuk memberikan jatah alamat bagi I/O port dalam sebuah skema *memory mapped I/O*. Walaupun jatah I/O port dalam suatu komputer dalam orde 10–100, suatu ruang besar 64 K dicadangkan untuk I/O port. Bagian ini tidak dapat digunakan untuk program. Karena itu hardware bertanggung jawab untuk pendekodean pada ruang alamat ini dan melakukan disable pada memori utama selama operasi baca/tulis untuk alamat pada ruang ini.
2. Pada beberapa sistem, sebagian dari memori utama dicadangkan sebagai video buffer (dikenal juga sebagai memori layar). Secara fisik, sebuah memori baca/tulis tetap ada dalam ruang alamat ini.

Ruang ini digunakan bersama (*share*) antara CPU dan pengontrol layar (CRT). CPU menyimpan “halaman memori” untuk menampilkan kannya pada layar CRT. Oleh karena itu bagian ini tidak disediakan untuk program biasa.

Gambar 7.30 menunjukkan bagaimana ruang memori utama 1MB pada sebuah IBM PC berbasis INTEL 8088 dijatah untuk melakukan berbagai fungsi. Alokasi ini dikenal dengan peta memori dan merupakan suatu spesifikasi penting yang diikuti oleh perancang hardware dan pengembang software. Karena IBM PC mengikuti metode *I/O mapped I/O* sebagai pengganti dari *memory mapped I/O*, maka tidak ada reservasi untuk I/O port.



Gambar 7.30 Peta memori pada PC

Penyediaan memori untuk program digunakan dalam dua cara:

1. Memori baca/tulis 640 KB adalah untuk program pemakai dan sistem operasi
2. Memori ROM diutamakan untuk BIOS.

BIOS menyimpan juga *self-test routine* dan “boot strap loader” terpisah dengan I/O driver. IBM PC orisinal juga mempunyai interpreter bahasa BASIC dalam ROM.

RINGKASAN

Memori utama adalah memori di mana CPU mengambil instruksi selama program dieksekusi. Untuk pengeksekusian program, dia harus ditransfer ke memori utama. Memori ini disebut juga dengan memori program atau memori primer. Sebuah memori RAM (random access memory) membolehkan mengakses suatu lokasi tanpa ada hubungan dengan posisi fisik dan tidak bergantung pada lokasi lain. Dengan kata lain, waktu akses sama untuk semua lokasi. Memori ini membolehkan operasi baca dan tulis. Memori inti magnetik paling banyak digunakan sebagai memori utama untuk beberapa tahun pada mainframe dan microcomputer. Saat ini secara total telah digantikan dengan memori semikonduktor. Keuntungan memori semikonduktor dibanding memori inti magnetik adalah konsumsi daya rendah, waktu akses cepat, *non-destructive readout* dan ukurannya kecil. Memori baca/tulis semikonduktor ada dua jenis yaitu static RAM (SRAM) dan dynamic ram (DRAM). Setiap sel dalam memori statik serupa dengan sebuah flip-flop. Saat ini chip DRAM mempunyai kemampuan built-in refresh. Dengan chip ini, maka tidak perlu dilakukan pembacaan atau penulisan data kembali. Sebagai gantinya, cukup menyediakan alamat lokasi dan masukan sinyal refresh secara periodik. Pembacaan dan penulisan ulang dilakukan secara internal oleh IC DRAM. Bila suatu baris di-refresh, semua sel-sel bit pada baris tersebut tercakup. Selama setiap siklus, alamat baris harus disuplai ke IC DRAM.

RAM digunakan untuk menyimpan sistem operasi dan program-program user. Read only memory (ROM) menyimpan secara permanen program-program kontrol. Memori ini dikelompokkan menjadi MROM, PROM, EPROM, EEPROM dan memori flash.

IC memori tersedia dalam berbagai kapasitas. Ruang alamat memori utama yang disediakan di dalam komputer terdiri dari RAM dan ROM. Sirkuit RAM dan ROM didesain dengan melakukan kaskade beberapa IC memori. Decoding ruang alamat dan pemilihan modul-modul memori yang relevan dilakukan oleh *memory control logic*.

SOAL-SOAL ULANGAN

1. Ada empat parameter penting yang signifikan dalam pemilihan sebuah memori komputer yaitu: (1) kapasitas, (2) kecepatan, (3) latency, dan (4) _____.
2. Kapasitas memori adalah jumlah lokasi memori dikalikan dengan _____.
3. Salah satu parameter penting memori adalah kecepatan operasi. Parameter-parameter kecepatan adalah _____, dan _____.
4. _____ adalah waktu yang diperlukan oleh memori untuk melengkapi operasi baca segera ketika menerima sinyal kontrol "baca"
5. Waktu yang dibutuhkan oleh memori untuk mempersiapkan akses word berikutnya disebut _____.
6. Kecepatan transfer data memori yang dinyatakan dalam jumlah byte per detik, disebut _____.
7. Beberapa jenis memori seperti hard disk, mempunyai organisasi internal yang dalam akses pertamanya ke suatu lokasi (sektor) mempunyai waktu akses yang lama sedangkan lokasi yang berdekatan mempunyai waktu akses yang lebih singkat. Waktu yang digunakan untuk mengakses lokasi (word) yang pertama dalam suatu rangkaian lokasi (blok lokasi) disebut _____.
8. Memori yang dapat mengakses 32 word, harus mempunyai saluran alamat minimal sebanyak _____ bit.
9. Secara umum, jika terdapat masukan saluran alamat N maka kapasitas/jumlah lokasi word yang dapat diakses sebanyak _____ word.
10. Sinyal kontrol yang umum untuk mengakses memori adalah (1) sinyal baca/tulis (R/W), dan (2) sinyal _____.
11. Klasifikasi memori yang umum adalah berdasarkan (1) fungsi, (2) teknologi, (3) modus penggunaannya (kemampuan baca/tulis), dan (4) _____.
12. Metode akses memori yang membolehkan pengaksesan ke suatu lokasi tanpa ada keterkaitan dengan posisi fisik dan tidak bergantung pada lokasi lain disebut metode _____.
13. Instruksi/data yang dibaca oleh prosesor (CPU) disimpan di dalam _____.

14. Untuk mempercepat proses pembacaan instruksi/data digunakan buffer tengah antara prosesor dan memori utama, memori ini disebut _____.
15. _____ adalah suatu fitur/konsep yang membantu untuk menjalankan program yang panjang dalam sebuah memori fisik yang kecil.
16. Contoh jenis memori akses urut/serial adalah (1) pita magnetik, dan (2) _____.
17. Memori MROM (*Mask-programmed Read Only Memory*) atau umumnya disebut ROM saja adalah memori yang isinya diprogram oleh _____.
18. Memori ROM yang dapat diprogram oleh user tetapi hanya sekali saja disebut memori _____.
19. Jenis memori ROM yang dapat diprogram oleh user beberapa kali adalah (1) _____ dan (2) _____.
20. Jenis memori ROM yang cara penghapusannya menggunakan sinar ultra violet disebut _____.
21. Jenis memori ROM yang cara penghapusannya menggunakan sinyal listrik disebut _____.
22. Memori _____ adalah sebuah EEPROM jenis khusus yang dapat dihapus dan diubah dalam blok pada satu operasi besar (seperti kilat).
23. Kelemahan utama memori _____ adalah karena data akan hilang jika catu dayanya mati, atau biasa disebut dengan istilah memori *volatile*.
24. Ada dua jenis memori semikonduktor baca/tulis yaitu: (1) memori _____ dan (2) memori _____.
25. Setiap sel di dalam memori _____ terdapat sebuah flip-flop yang menyimpan 1 atau 0 selama ada suplai daya.
26. Memori _____ membutuhkan ruang lebih banyak daripada DRAM tetapi lebih cepat.
27. Setiap sel di dalam memori _____ terdapat sebuah kapasitor yang menyimpan 1 atau 0 seperti cara penyimpanan muatan pada kapasitor, yang akan mengalami discharge (pembuangan muatan) selama periode waktu tertentu.
28. Memori yang isinya hanya dapat bertahan dalam beberapa saat karena terjadi discharge sehingga membutuhkan penyegaran ulang secara periodik disebut memori _____.
29. Memori _____ adalah jenis memori yang digunakan sebagai memori cache.

30. Memori _____ adalah jenis memori yang digunakan sebagai memori utama.
31. Word memori yang mempunyai rentang alamat dari 0 sampai 63 desimal dibutuhkan saluran alamat minimal _____ bit
32. Jika pada saluran alamat terdapat sinyal 011010_2 berarti word yang diakses berada pada nomor lokasi _____ desimal.
33. Memori SRAM tersedia dalam teknologi (1) bipolar, (2) MOS, dan (3) _____.
34. Keunggulan teknologi bipolar dibanding teknologi CMOS adalah _____.
35. Keunggulan teknologi CMOS dibanding teknologi bipolar adalah (1) _____, dan (2) _____.

SOAL-SOAL LATIHAN

1. Jelaskan arsitektur internal dari sebuah ROM yang menyimpan 64KB dan menggunakan sebuah larik register persegi.
2. IC 4164 adalah sebuah DRAM 64K (64K x 1) yang mempunyai waktu akses 100 ns. Organisasi internalnya adalah matriks 256 x 256 (secara internal memiliki array dua 32K bit). Jika refresh dilakukan hanya dengan prinsip RAS, hitung *refresh overhead*.
Catatan: setiap sel harus di-refresh sedikitnya sekali dalam 2 ms.
3. IC 814400 adalah sebuah DRAM 1M x 4. Parameter-parameternya adalah:
 - a) Waktu siklus 80 ns
 - b) Waktu refresh 60 ns
 - c) Siklus refresh 1024 siklus, 16.4 ms
 Hitung refresh overhead.
4. Sebuah komputer yang memorinya dibangun menggunakan IC 81C4256 DRAM 256K x 4. Hitung jumlah IC (chip) yang dibutuhkan untuk memperoleh kapasitas 4MB .
5. Sebuah komputer mempunyai bus alamat 26 bit. Jika IC (chip) DRAM yang tersedia adalah 4M x 4, tentukan:
 - a) Kapasitas memori DRAM maksimum komputer yang dapat diakses
 - b) Jumlah IC yang dibutuhkan untuk membangun modul DRAM komputer secara kaskade

- c) Kapasitas decoder alamat yang digunakan untuk mengakses bank memori
- d) Rentang alamat (dalam hex) pada masing-masing bank memori
- e) Gambarkan organisasi modul DRAM tersebut dilengkapi hubungan bus-alamat dengan decoder alamatnya.
6. Sebuah 2167 SRAM 16K x 1. Berapa banyak IC seperti itu yang diperlukan untuk membangun RAM 64 KB?
7. Sebuah RAM 32 KB dibentuk dari 16 buah jenis IC SRAM. Jika setiap IC memerlukan 14 bit alamat, tentukan:
 - a) Kapasitas IC
 - b) Organisasi memori
 - c) Rentang alamat untuk setiap bank.

BAB 8

TEKNIK MANAJEMEN MEMORI

Sasaran bab ini:

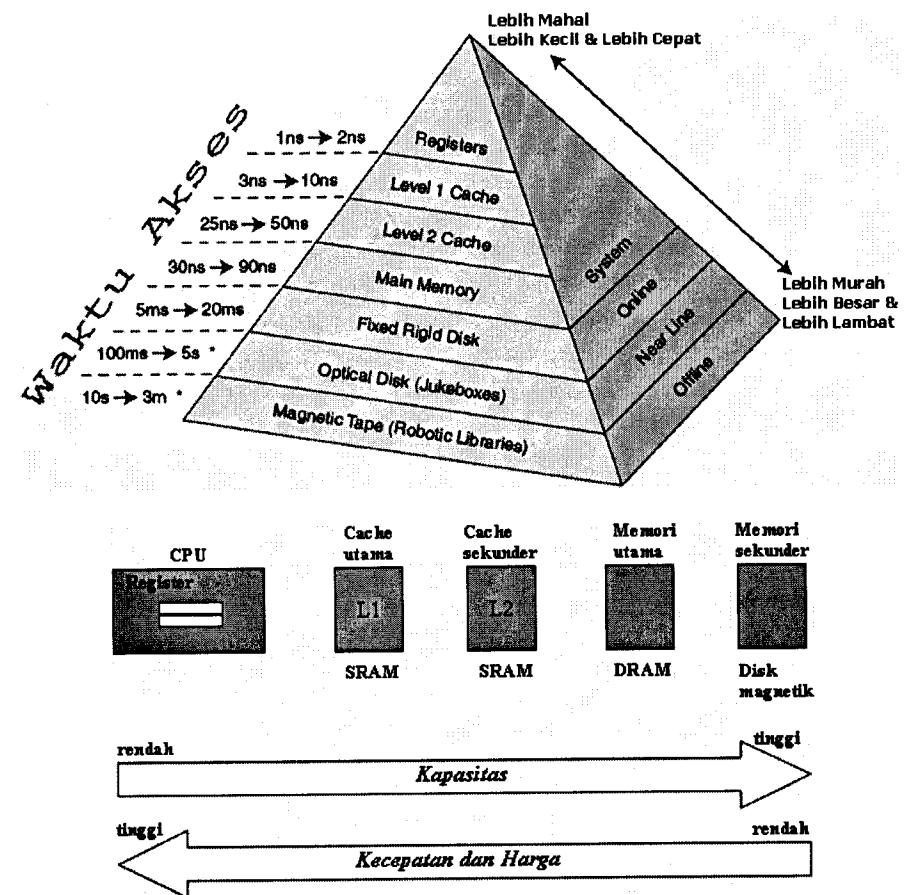
1. Prefetch instruksi
2. Interleave Memori
3. Buffer Tulis
4. Memori Cache
5. Memori Virtual

Kecepatan dan kapasitas memori utama merupakan hal yang kritis terhadap kinerja dan harga suatu sistem. Sehingga seorang arsitek komputer perlu mencapai dua persyaratan yang saling konflik: kecepatan dan memori besar yang harganya murah. Pengembangan teknologi dalam perjalanan waktu secara tetap menawarkan memori utama yang besar dan lebih cepat dengan harga yang rendah. Untuk melampaui kemungkinan teknologi memori yang ada, perancang komputer telah menemukan beberapa teknik desain yang mengompensasi memori utama yang kecil dan lambat yang disebut teknik manajemen memori. Beberapa dari teknik ini adalah *interleave* memori (*memory interleaving*), *prefetch* instruksi (*instruction prefetch*), buffer tulis (*write buffer*), memori cache dan memori virtual. Penggunaan memori cache adalah sebuah teknik pengembangan kinerja sistem. Memori cache telah digunakan dalam komputer mainframe sejak adanya memori *magnetic core*. Mikrokomputer telah memunculkan kembali memori cache karena adanya penurunan harga memori cache. Memori virtual adalah merupakan konsep lain yang juga merupakan fitur dari mainframe yang mengalami pemunculan kembali dalam mikrokomputer. Tujuannya adalah untuk menfasilitasi pengeksekusian program-program besar tanpa peningkatan memori secara fisik. Walaupun memori cache dan memori virtual merupakan dua konsep yang berbeda dengan tujuan yang berbeda dalam suatu sistem komputer, beberapa

teknik implementasinya serupa. Memori cache adalah suatu fitur hardware yang tidak dikenal pada software, sedangkan memori virtual merupakan fitur hardware-software. Bab ini membahas prinsip-prinsip dan teknik desain yang digunakan untuk menghasilkan kinerja yang lebih baik dengan biaya per bit yang rendah.

8.1 HIERARKI MEMORI

Idealnya memori utama haruslah cepat dan besar agar program yang besar dapat disimpan dan dieksekusi. Ada beberapa teknologi memori yang berbeda seperti memori semikonduktor, hard disk magnetik, pita magnetik, disk optik, dan sebagainya. Masing-masing mempunyai keunikan dan rentang waktu akses yang berbeda. Harga per bit-nya pun bervariasi. Tidak semua teknologi memori dapat digunakan sebagai memori utama. RAM merupakan pilihan yang baik sebagai memori utama (memori program). Memori akses serial (pita magnetik) atau memori akses semi acak (hard disk magnetik) tidak tepat digunakan sebagai memori utama walaupun menawarkan bandwidth yang besar. Memori ini paling baik digunakan sebagai memori cadangan atau memori sekunder untuk menyimpan program volume besar. Walaupun lambat, memori ini begitu murah dalam kapasitas besar yang dapat disimpan dalam sistem komputer. Hardware dan sistem operasi secara bersama mengeluarkan program yang benar dari memori sekunder ke memori utama dengan waktu yang relevan agar CPU tidak menahan data yang diinginkan instruksi selama pengaksesan ke memori utama. Komputer praktis, menggunakan hierarki memori seperti yang ditunjukkan pada Gambar 8.1.



Gambar 8.1 Hierarki memori

Memori yang terdekat ke CPU dalam Gambar 8.1 mempunyai kecepatan yang tertinggi (waktu akses singkat) dan juga harga per bit tertinggi. Karena itu kapasitasnya terkecil dalam komputer. Bila menjauh dari CPU yaitu dari register ke memori sekunder, kecepatan dan harga menurun dari satu level ke level berikutnya. Karena itu kapasitas meningkat dari level ke level. Kita memaksimalkan kapasitas ketika menjauhi CPU. Tujuannya ada tiga:

1. Kapasitas total sebesar mungkin yang kita perlukan.
2. Harga total sekecil mungkin

3. Waktu tunggu rata-rata CPU untuk mendapatkan informasi yang dibutuhkan harus sekecil mungkin.

Saat ini, memori sekunder yang populer digunakan adalah hard disk magnetik. Hard disk drive tunggal menawarkan kapasitas penyimpanan yang besar dari 80 GB dan lebih. Komputer dapat mempunyai hard disk drive lebih dari satu/jamak. Dahulu, komputer menggunakan drum magnetik yang saat ini telah usang dengan ditemukannya hard disk magnetik. Hard disk lebih cepat beberapa kali daripada drum magnetik. Teknologi disk optik menawarkan keandalan yang lebih baik dibandingkan dengan hard disk magnetik, tetapi lebih lambat. Karena itu, tak dapat digantikan dengan hard disk. Memori ini digunakan sebagai penyimpan cadangan di samping hard disk.

Saat ini, DRAM digunakan sebagai memori utama baca/tulis pada sebagian besar komputer untuk tambahan pada ROM semikonduktor yang jumlahnya kecil dalam penyimpanan program-program kontrol permanen. SRAM lebih cepat dari DRAM, tetapi harga per bit SRAM lebih tinggi. Sehingga SRAM yang besar tidak diberikan pada komputer secara umum kecuali komputer kinerja tinggi seperti supercomputer yang dapat mempunyai SRAM yang besar.

SRAM dipilih sebagai memori cache karena kecepatannya yang tinggi. Memori cache dapat ditempatkan lebih dari satu level. Kebanyakan mikroprosesor saat ini, dimulai dari Intel 80486, mempunyai memori cache yang terdapat dalam prosesor (*on-chip*) dan disebut memori cache internal. Mikroprosesor kinerja tinggi seperti Pentium pro dan berikutnya mempunyai dua level. Level tersebut dikenal dengan cache L1 dan cache L2. Cache on-chip lebih cepat daripada cache off-chip dengan teknologi yang sama. Hal ini karena waktu tunda propagasinya hampir nol di dalam lintasan antara prosesor dan cache on-chip.

Register yang terdapat dalam prosesor diakses oleh CPU dengan segera. Karena itu menggiurkan untuk menggunakan register sebanyak mungkin, tetapi register mahal dan ukuran chip prosesor meningkat tajam karena jumlah register yang besar.

8.2 KEKURANGAN MEMORI UTAMA

Ada dua isu yang berhubungan dengan fungsi memori utama dalam komputer: kecepatan dan kapasitas memori utama. Kecepatan memori utama tidak sama dengan prosesor dan karena itu menjadi beban bagi CPU. Lagi pula kapasitas fisik memori utama tidak dapat ditingkatkan untuk pemenuhan program-program panjang karena pengaruh harga. Perancang telah menemukan solusi dari dua kendala memori utama ini dengan mengadopsi perubahan arsitektur. Hal ini akan dibahas berikut ini.

Kecepatan Memori Utama

Kinerja sistem utamanya tergantung waktu siklus instruksi. Kecepatan clock prosesor dan waktu akses memori utama merupakan dua faktor penting yang berkontribusi terhadap waktu siklus instruksi. Sedikitnya satu akses memori utama diperlukan setiap instruksi untuk pengambilan instruksi dari memori utama. Dan juga, selama pengeksekusian beberapa instruksi, akses memori utama diperlukan untuk pengambilan operand dan penyimpanan hasil. Waktu prosesor terbuang jika waktu akses memori besar (dianggap tidak ada pipelining dalam prosesor). Hal ini merupakan masalah waktu tunda keseluruhan yang terdapat pada komputer. Empat teknik berikut menyelesaikan masalah ini dan membolehkan prosesor untuk bekerja dengan baik pada memori utama yang lambat:

1. Prefetch instruksi
2. Interleave memori
3. Buffer tulis
4. Memori cache

Masing-masing keempat teknik tersebut berbeda dalam kompleksitas hardware dan efisiensi. Saat ini mikroprosesor menggabungkan semua teknik ini dalam satu prosesor. Karena itu, secara teoretis memungkinkan untuk mempunyai memori fisik sebanyak yang dibutuhkan oleh program. Tetapi ada beberapa masalah praktis yang berhubungan dengan hal ini:

1. Harga memori utama meningkat dan karena itu harga sistem meningkat.
2. Kebutuhan catu daya meningkat karena harga sistem juga meningkat.

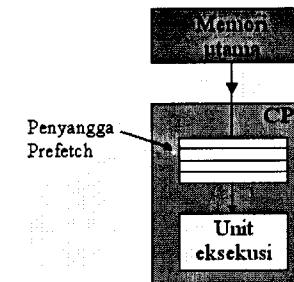
- Karena hardware meningkat (memori dan catu daya), panas yang dihasilkan meningkat melampaui persyaratan pendingin tambahan (kipas pendingin/pengondisi udara).

Dari pengamatan ini, jelas dengan meningkatnya ukuran memori utama yang melampaui batas menyebabkan biaya kumulatif meningkat. Dengan tidak adanya memori virtual, teknik *program folding (overlay)* digunakan untuk mengeksekusi program besar. Pada teknik ini, program besar dibagi menjadi beberapa bagian oleh pemrogram. Setiap bagian di-load ke dalam memori utama sebelum eksekusi dan penyimpanan pada hard disk setelah eksekusi. Fitur memori virtual merupakan solusi yang tepat untuk mengeksekusi program yang panjang dengan memori utama yang terbatas, tetapi mempunyai dua penalti:

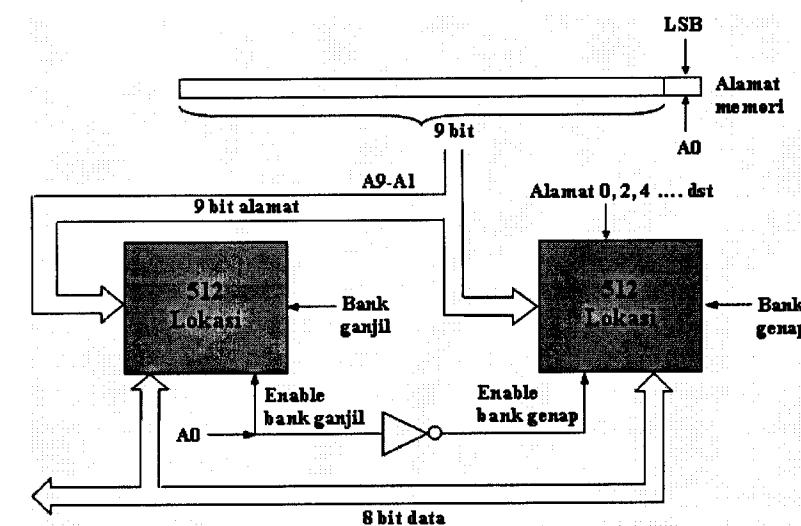
- Program harus menggunakan alamat logika dan bukan alamat fisik untuk instruksi dan data.
- Waktu eksekusi program meningkat karena translasi alamat (alamat logika ke fisik) oleh prosesor.

8.3 PREFETCH INSTRUKSI

Tujuan prefetch instruksi adalah untuk menuju ke instruksi berikutnya dari memori utama, mendahului, ketika instruksi yang ada (*current instruction*) dieksekusi oleh prosesor. Hal ini dicapai melalui sekumpulan “penyangga prefetch” di mana instruksi-instruksi yang diambil sebelumnya disimpan. Gambar 8.2 menunjukkan konsep prefetch instruksi. Penyangga prefetch menyimpan suatu antrian instruksi karena beberapa instruksi diambil sebelumnya dari memori utama. Ketika prosesor menyelesaikan suatu eksekusi instruksi, dia hanya mengambil instruksi berikutnya dari antrian instruksi. Karena itu prosesor tidak membuang waktu untuk pengambilan instruksi. Hal ini dimungkinkan dengan melakukan tumpang tindih fase pengambilan dan fase eksekusi instruksi. Terlepas apakah mempunyai hardware “antrian” atau register untuk penyimpanan instruksi prefetch, unit kontrol harus lebih pintar dalam pelacakan status antrian dan prefetch instruksi.



Gambar 8.2 Penyangga prefetch



Gambar 8.3 (a) Interleaving 2-way

Prefetch instruksi adalah suatu fitur hardware dan program yang bukan menyangkut keberadaan antrian instruksi. Unit kontrol harus menangani situasi khusus tersebut. Anggap instruksi sekarang yang dieksekusi adalah suatu instruksi JUMP, maka instruksi berikutnya yang diperlukan tidak berada dalam antrian instruksi. Karena itu, antrian kosong dan set instruksi yang baru diambil dari alamat lompatan. Keberadaan instruksi JUMP dalam program adalah terbatas dan bervariasi dari program ke program. Karena itu, prosesor dapat menghasilkan antrian instruksi yang kosong dan memulai pengambilan instruksi baru setelah instruksi JUMP. Sebagai

hasilnya, prosesor harus menunggu waktu pengambilan instruksi setelah setiap instruksi JUMP.

Contoh 8.1

Sebuah CPU memerlukan 50 ns untuk pengambilan instruksi. Hitung peningkatan kinerja jika fitur prefetch instruksi termasuk dalam prosesor. Anggap bahwa 20% dari instruksi adalah instruksi pencabangan.

- a) Kasus 1: tidak ada fitur prefetch instruksi

$$\begin{aligned} \text{Waktu siklus instruksi} &= \text{waktu pengambilan} + \text{waktu eksekusi} \\ &= 50 \text{ ns} + t_e \text{ di mana } t_e \text{ adalah waktu} \\ &\quad \text{eksekusi instruksi} \end{aligned}$$

$$\begin{aligned} \text{Anggap program mempunyai 100 instruksi, total waktu eksekusi:} \\ &= 100 \times (t_e + 50) \text{ ns} = (100 t_e + 5000) \text{ ns} \end{aligned}$$

- b) Kasus 2: ada prefetch instruksi

Dalam 100 instruksi, 20 adalah instruksi cabang dan lainnya adalah instruksi biasa. Selama eksekusi program, pengambilan instruksi oleh prosesor hanya dibutuhkan 20 kali yaitu setelah setiap instruksi JUMP. Untuk sisanya 80 instruksi, prosesor tidak mempunyai waktu pengambilan (karena antrean instruksi) dan hanya mempunyai waktu eksekusi. Total waktu eksekusi = waktu eksekusi selama 80 instruksi tanpa lompatan + waktu siklus instruksi untuk 20 instruksi lompatan.

$$\begin{aligned} &= 80 t_e + 20 \times (t_e + 50) \text{ ns} \\ &= 100 t_e + 1000 \text{ ns} \end{aligned}$$

Perbedaan dalam waktu eksekusi total = 4000 ns terhadap 100 instruksi

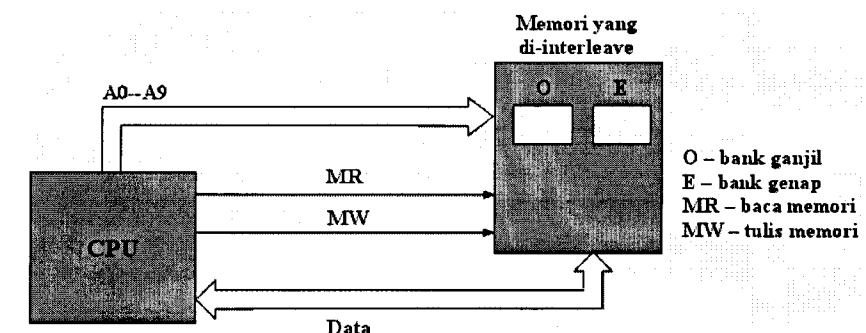
Perolehan rata-rata per instruksi karena prefetch instruksi = 40 ns

8.4 INTERLEAVE MEMORI

Bandwidth memori yaitu jumlah byte yang ditransfer per detik. CPU mampu menangani bandwidth yang lebih tinggi daripada memori. Operasi internal CPU dikontrol oleh clock berkecepatan tinggi. Karena kecepatan antara CPU dan bandwidth memori tidak sesuai, maka CPU harus selalu istirahat/menunggu instruksi atau data dari memori. Interleave memori adalah suatu teknik reorganisasi (pembagian) memori utama

menjadi modul-modul independen yang jamak agar bandwidth ditingkatkan menjadi beberapa kali.

Gambar 8.3(a) dan (b) mengilustrasikan dua cara interleave untuk CPU 8 bit. CPU mempunyai alamat memori 10 bit. Ruang memori fisik sebanyak 1024 byte (1 KB). Memori 1024 x 8 bit, diganti dengan dua modul yang masing-masing 512 lokasi x 8 bit. Masing-masing modul memori berdiri sendiri dan mempunyai dekoder alamat sendiri dan kontrol baca/tulis. Karena kapasitas setiap modul adalah 512 lokasi, maka dibutuhkan 9 bit untuk alamat setiap modul. Alamat memori dari CPU mempunyai 10 bit, A0-A9. Sembilan bit MSB yang keluar A9-A1 diberikan sebagai alamat untuk kedua modul tersebut. LSB A0 mengontrol *enable/disable* modul memori. Bila A0 = 0, (alamatnya adalah alamat genap), bank genap di-enable dan bank ganjil di-disable. Bila A0 = 1 (alamatnya adalah alamat ganjil) bank ganjil di-enable dan bank genap di-disable. Karena itu, bila CPU mengirimkan sebuah alamat 10 bit, hanya salah satu bank yang di-enable sesuai dengan status A0. CPU hanya memandang satu unit memori dengan 1024 lokasi. Sekarang CPU dapat memulai pembacaan dua memori untuk dua alamat berbeda, satu ganjil dan satu genap, kemudian satu setelah yang lainnya. Marilah kita menganggap bahwa CPU mengirimkan alamat pertama 1000 dan alamat 1001 berikutnya. Bila alamat 1000 diterima, bank genap terpilih dan dia mulai operasi baca. Bila alamat 1001 diterima, bank ganjil terpilih dan dia juga mulai membaca. Jadi, bank-bank terpilih satu demi satu dan kedua bank melakukan operasi baca secara paralel. CPU menerima data dari dua bank dengan urutan yang sama.



Gambar 8.3 (b) Antarmuka CPU – memori untuk interleaving 2-way

Jika operasi non-interleaving, CPU melakukan operasi baca memori untuk alamat 1000 pertama dan setelah waktu siklus berakhir, barulah kemudian dilakukan operasi baca untuk alamat 1001 berikutnya. Jika waktu siklus memori 50 ns, maka total waktu akses yang digunakan untuk dua pengaksesan adalah 100 ns. Karena itu bandwidth-nya adalah 2 byte/100 ns = 20 MB/sec. Dalam interleaving 2-way, dua pengaksesan selesai dalam 50 ns. Karena itu bandwidth memori = 2 byte/50 ns = 40 MB/sec. Jadi, interleaving 2-way menggandakan bandwidth memori. Secara umum, interleave memori n -way memberikan bandwidth sebesar n -kali bandwidth kasus non-interleaving.

Contoh 8.2

Sebuah CPU mempunyai ruang memori 1KB. Rancang interleaving memori 4-way dengan IC memori dengan waktu siklus 50 ns dan hitung bandwidth efektif.

$$\text{Ruang alamat} = 1024 \text{ byte}$$

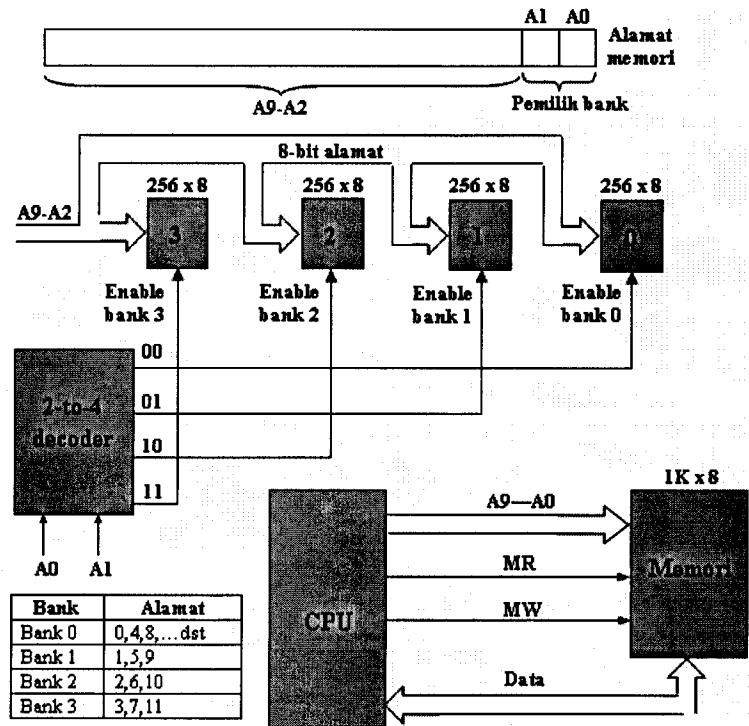
Karena itu, jumlah bit alamat = 10

Karena interleaving 4-way yang digunakan, setiap modul akan mempunyai 256 lokasi. Alamat memori 10 bit dari CPU digunakan sebagai:

A9—A2 : lokasi memori dalam modul

A0—A1 : pemilihan bank

Dua LSB (A0 dan A1) menunjuk bank. Dekoder 2-ke-4 digunakan untuk mendekodekan bit-bit ini dan memilih bank. 8 MSB (A9—A2) diberikan ke semua bank. Gambar 8.4 menunjukkan rancangan pada level fungsional.



Gambar 8.4 Interleaving memori 4-way

Pengaruh Interleave Memori

Interleaving n -way melipatgandakan bandwidth n -kali. Kekurangannya adalah:

1. Diperlukan pendekodean bank dan sirkuit enable. Hal ini meningkatkan biaya.
2. Waktu tunda nominal didahului oleh sirkuit logika pendekode bank yang harus diperhitungkan ketika penghitungan waktu akses efektif dan bandwidth.
3. CPU harus mempunyai sirkuit logika tambahan untuk memulai dan menangani serangkaian siklus baca/tulis memori.

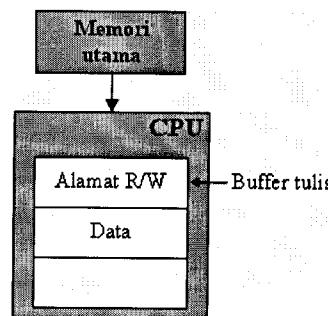
Ketiga kekurangan ini tidak terlalu serius dalam peningkatan kinerja yang disebabkan interleave memori.

8.5 BUFFER TULIS

Buffer tulis secara fungsional adalah kebalikan dari prefetch instruksi. Dia digunakan untuk melakukan operasi penulisan memori untuk kepentingan CPU sehingga membebaskan CPU dari operasi penulisan memori. Buffer tulis berisi informasi berikut:

1. Alamat memori di mana operasi tulis harus dilakukan.
2. Data yang akan ditulis

Umumnya buffer tulis dapat menyimpan informasi untuk lebih dari satu operasi tulis. Gambar 8.5 menunjukkan konsep buffer tulis. Umumnya bila CPU menulis data dalam memori utama, maka dia menyimpan hasil dari suatu instruksi. Jika CPU harus terlibat secara penuh dalam siklus penulisan memori, maka dia memperlambat permulaan siklus instruksi berikutnya.



Gambar 8.5 Buffer tulis

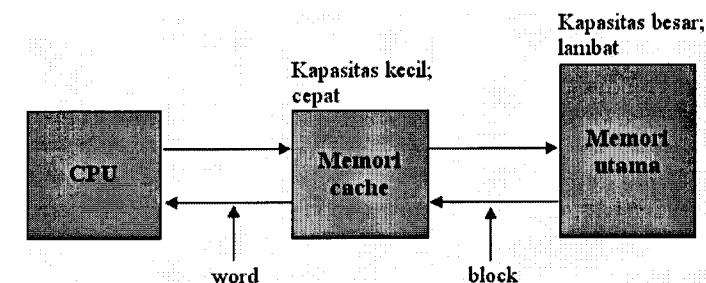
Jika memori sedang sibuk mengerjakan beberapa operasi yang dimulai lebih dulu (misalnya oleh pengontrol DMA), maka dia selanjutnya mengajukan waktu tunggu pada CPU dimana hanya operasi tulis yang dapat dimulai. Namun, instruksi berikutnya biasanya sudah diambil sebelumnya dan disediakan dalam CPU, karenanya waktu yang berharga dari CPU terbuang karena harus melakukan operasi tulis memori. Ini menguntungkan jika CPU terbebas dari operasi tulis memori sehingga CPU dapat memulai siklus instruksi berikutnya. *Buffer tulis* digunakan oleh CPU untuk mengeluarkan informasi tentang siklus tulis memori yang akan dikerjakan dan CPU selanjutnya beralih untuk memulai siklus instruksi berikutnya. *Buffer tulis* menunggu kesempatan yang tepat untuk melakukan operasi

tulis yaitu sampai memori bebas. Kemudian melakukan siklus tulis memori dengan mengambil informasi dari *buffer tulis*.

Apa yang terjadi jika instruksi yang sekarang membutuhkan data memori dari suatu lokasi memori untuk operasi tulis yang tertunda dalam *buffer tulis*? Pada kasus demikian, kesempatan CPU untuk membaca/mengambil data dari memori utama tidak berguna (sia-sia). Karena itu sebelum memulai operasi baca memori, *buffer tulis* diperiksa untuk melihat jika ada operasi tulis yang ditunda pada alamat memori yang sudah ada. Jika ya, CPU tidak membaca dari memori utama. Sebagai gantinya, dia menyalin data dari *buffer tulis*. Hal ini merupakan keuntungan tambahan yang diberikan oleh *buffer tulis* walaupun hanya terjadi kadang-kadang.

8.6 MEMORI CACHE

Memori cache adalah suatu buffer tengah antara CPU dan memori utama (lihat Gambar 8.6). Tujuan memori cache adalah untuk mengurangi waktu tunggu CPU selama pengaksesan memori utama. Pada sistem yang tanpa memori cache, setiap akses memori utama menghasilkan sejumlah waktu tunda dalam pemrosesan instruksi karena waktu akses memori utama yang lebih besar dari periode clock prosesor. Kecepatan CPU yang tinggi terbuang selama akses memori (ketika pengambilan instruksi, pengambilan operand dan penyimpanan hasil) karena waktu akses memori utama yang besar. Untuk mengurangi waktu tunggu CPU, diperkenalkan sebuah memori yang kecil tetapi cepat sebagai memori cache yang berada di antara memori utama dan CPU.



Gambar 8.6 Penggunaan memori cache

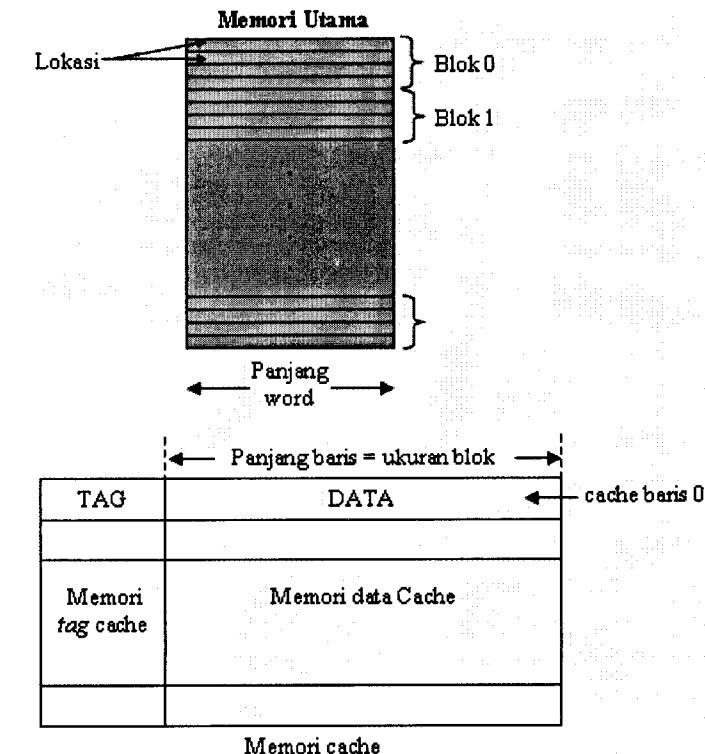
Sebagian program dan data dibawa sebelumnya ke dalam memori cache. Pengontrol cache menjaga lokasi memori utama yang akan disalin pada memori cache pada waktu yang diberikan. Bila CPU memerlukan suatu instruksi atau operand, dia menerimanya dari memori cache (jika tersedia) sebagai pengganti dari pengaksesan memori utama. Jadi, akses memori selesai dalam waktu yang singkat sehingga memori utama tampak seperti memori yang cepat. Namun, jika item yang diperlukan tidak terdapat di dalam memori cache, maka akses ke memori utama tidak dapat dihindari sehingga item dibaca dari memori utama dan diletakkan di memori cache. Kapasitas memori cache sangat kecil (dibandingkan dengan memori utama) karena mahal. Jika memori cache penuh, maka beberapa item dihapus. Keberadaan memori cache tidak dikenal pada program pemakai (user program) dan bahkan oleh prosesor. Hardware “cache controller” mengatur semua persyaratan dengan melakukan operasi-operasi yang diperlukan. Seperti yang tampak pada Gambar 8.6, transfer antara memori utama dan memori cache adalah dalam bentuk block. Transfer antara CPU dan memori cache biasanya satu word dalam satu waktu. Memori cache secara fisik dapat diintegrasikan dengan IC prosesor sebagai cache internal, yang juga dikenal dengan nama *on-chip cache*.

8.6.1 Prinsip Cache

Operasi cache berdasarkan pada “*locality of reference*” yang merupakan sifat yang melekat dalam program. Hampir semua waktu, pemrosesan instruksi dan data yang diperlukan tersedia dalam lokasi memori utama tersebut yang secara fisik dekat dengan lokasi memori utama yang sedang diakses. Ada dua macam pola tingkah laku:

1. *Temporal locality*: Instruksi yang sedang diambil dapat diperlukan kembali dengan segera.
2. *Spatial locality*: Instruksi-instruksi yang berdekatan dengan instruksi sekarang dapat diperlukan segera.

Dari pandangan dua sifat ini, bila pengaksesan memori utama untuk suatu instruksi, sebagai pengganti dari pengambilan satu instruksi saja dari memori utama, beberapa instruksi berurutan diambil bersama dan disimpan dalam memori cache.



Gambar 8.7(a) Struktur memori utama dan memori cache

Gambar 8.7(a) menunjukkan organisasi memori utama dan memori cache. Secara konseptual memori utama dibagi menjadi banyak blok, setiap blok berisi sejumlah lokasi memori berurutan yang tetap. Selama pembacaan suatu lokasi dari memori utama, isi seluruh blok ditransfer dan disimpan dalam memori cache. Memori cache disusun atas sejumlah baris-baris (disebut juga blok) dan ukuran setiap baris sama seperti kapasitas blok memori utama. Jumlah blok-blok memori utama lebih banyak daripada jumlah baris dalam memori cache. Karena itu, suatu formula (atau fungsi) digunakan untuk memetakan secara simetris suatu blok memori utama ke salah satu baris cache. Bila suatu blok memori utama disimpan dalam memori cache, maka baris cache (nomor baris) di mana blok harus ditulisi ditentukan dari fungsi pemetaan (*mapping function*). Dengan cara yang sama, bila CPU membaca dari suatu lokasi memori utama, maka pengontrol

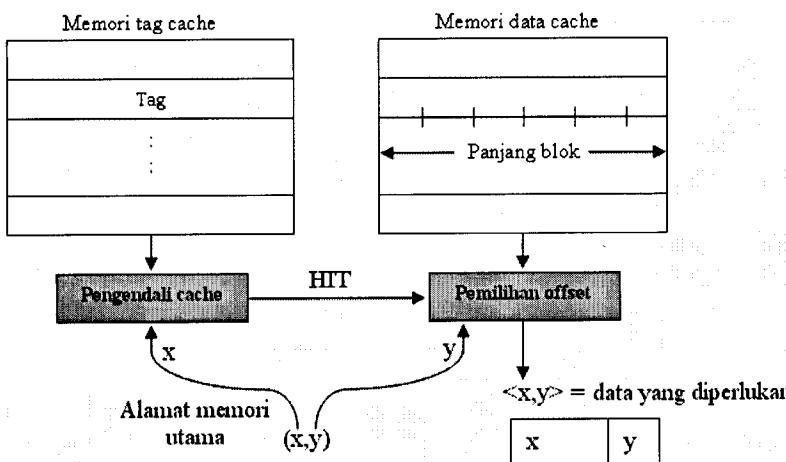
cache menggunakan fungsi pemetaan yang mengidentifikasi baris cache di mana blok memori utama disimpan (jika tersedia). Ada tiga metode popular untuk pemetaan memori cache:

1. Pemetaan langsung (*direct mapping*)
2. Pemetaan asosiatif penuh (*fully associative mapping*)
3. Pemetaan asosiatif set (*set associative mapping*)

Secara fungsional, memori cache mempunyai dua bagian seperti yang ditunjukkan pada Gambar 8.7(b):

1. Memori data cache
2. Memori tag cache

Setiap baris dari memori data cache berisi blok memori. Baris tag yang bersesuaian dalam memori tag cache menunjukkan jumlah blok dari blok memori ini. Pola tag adalah suatu bagian alamat memori utama dari blok. Ini berbeda untuk setiap teknik pemetaan (langsung, asosiatif penuh, dan asosiatif set) yang akan dibahas pada bagian berikutnya.



Gambar 8.7(b) Prinsip dasar operasi cache

8.6.2 Hit dan Miss

Bila CPU mengakses memori utama, pengontrol cache memeriksa memori cache untuk melihat apakah alamat memori sekarang yang dilepaskan oleh CPU dipetakan ke dalam cache. Jika alamat memori yang sekarang dipetakan, item yang diperlukan tersedia dalam memori cache. Keadaan ini disebut "*cache hit*". Informasi yang diperlukan dibaca dari memori cache. Pada sisi lain, jika alamat memori utama yang sekarang tidak dipetakan dalam memori cache, informasi yang diperlukan tidak tersedia dalam memori cache. Keadaan ini disebut "*cache miss*". Karena itu semua blok yang berisi alamat memori utama dikirim ke dalam memori cache. Sekarang ada dua cara suplai informasi ke prosesor:

1. Simpan semua blok dalam memori cache pertama dan kemudian baca item yang diperlukan dari memori cache dan suplai ke prosesor.
2. Item yang segera diperlukan dibaca dari memori utama, disuplai ke prosesor dan kemudian disimpan dalam memori cache.

Cache hit yang lebih sering terjadi lebih baik karena setiap terjadi '*miss*' maka harus dilakukan pengaksesan memori utama. Waktu yang digunakan untuk membawa item yang diperlukan dari memori utama dan menyuplainya ke prosesor disebut '*miss penalty*'. Hit rate (disebut juga *hit ratio*) menyediakan bilangan jumlah akses yang dihadapi '*cache hit*' terhadap jumlah akses yang terjadi. Hit rate bergantung pada faktor-faktor berikut:

1. Ukuran baris memori cache.
2. Kapasitas total memori cache.
3. Fungsi pemetaan yang digunakan.
4. Algoritma penggantian yang digunakan oleh sistem operasi yang memutuskan apakah akan berada dalam memori cache.
5. Jenis program yang dijalankan.

Algoritma penggantian universal yang dapat mengurangi hit rate untuk semua situasi adalah mustahil.

8.6.3 Pemetaan Langsung

Pada pemetaan langsung, blok memori utama yang ada dipetakan ke suatu baris khusus dalam memori cache. Dengan kata lain, pengontrol

cache tidaklah fleksibel dalam penempatan blok memori utama dalam baris-baris memori cache. Gambar 8.8 menunjukkan konsep pemetaan langsung. Alamat memori utama dikelompokkan ke dalam tiga field: TAG, LINE, WORD. Bit-bit dalam field WORD menunjukkan banyaknya WORD dalam blok. Field TAG dan LINE bersama-sama menentukan jumlah blok. Anggap memori cache mempunyai c baris (blok) dan memori utama mempunyai m blok. Pemetaan langsung ditetapkan sebagai:

$$l_{cm} = b_{mm} \text{ modulo } c$$

di mana l_{cm} = nomor baris cache

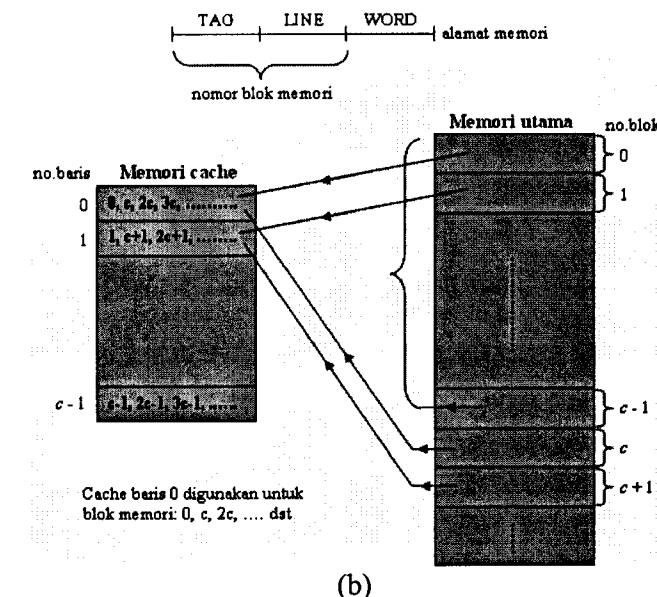
b_{mm} = nomor blok memori utama

c = jumlah total baris cache

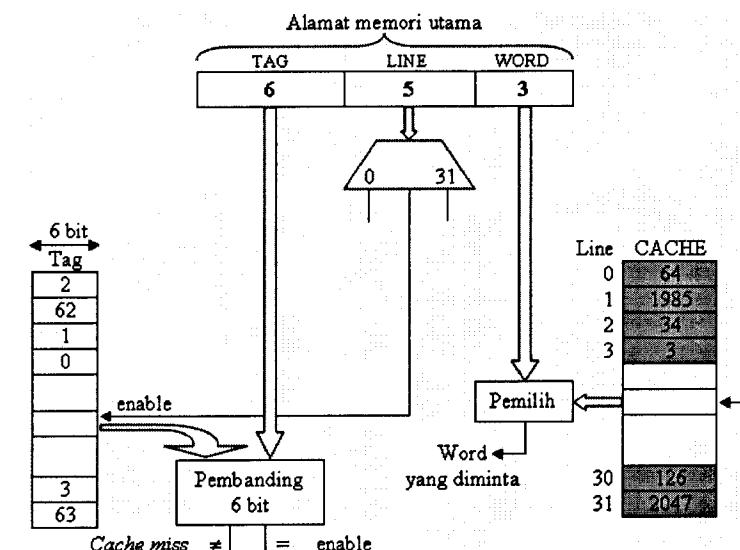
Gambar 8.8 juga menunjukkan alokasi blok-blok memori yang berbeda untuk membedakan baris-baris cache pada setiap pemetaan langsung. Bila prosesor melakukan operasi pembacaan memori utama, pengontrol cache mendapatkan nomor baris cache dari field LINE alamat memori utama. Tag pada baris cache tersebut cocok (match) dengan field TAG pada alamat memori. Sekarang ada dua kemungkinan:

1. Jika dia cocok, maka dia adalah HIT. Data yang diperlukan tersedia dalam cache data pada baris ini. Pembacaan dari baris tersebut dimulai. Word yang diperlukan dalam blok tersebut diseleksi menggunakan field WORD dalam alamat memori.
2. Jika tag dalam baris cache tidak cocok dengan field TAG dalam alamat memori, maka dia adalah MISS. Karena itu, pengontrol cache mulai melakukan operasi pada memori utama. Semua blok ditransfer dan disimpan dalam baris cache tersebut mengantikan blok lama. Memori tag juga diperbarui dengan field TAG dari alamat memori. Word yang diperlukan dikirim ke prosesor.

Ilustrasi protokol yang mengatur permintaan word oleh prosesor pada pemetaan langsung ditunjukkan pada Gambar 8.9.



Gambar 8.8 Pemetaan langsung



Gambar 8.9 Protokol translasi alamat pemetaan langsung

Berdasarkan teknik pemetaan langsung, MMU (*memory management unit*) melakukan translasi alamat yang diberikan oleh prosesor dengan melakukan pembagian alamat menjadi tiga field seperti yang ditunjukkan pada Gambar 8.8. Panjang bit pada setiap field tersebut adalah:

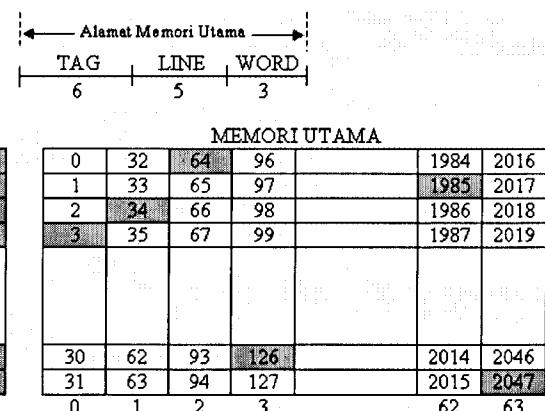
- field untuk word = $\log_2 B$ di mana B adalah jumlah word dalam blok
- field untuk blok = $\log_2 N$, di mana N adalah jumlah baris dalam memori cache
- field untuk tag = $\log_2 (M/N)$, di mana M adalah jumlah blok dalam memori utama
- Jumlah bit keseluruhan di dalam memori utama adalah = $\log_2 (B \times M)$

Contoh 8.3

Misalkan sebuah memori utama mempunyai 2K blok, memori cache 32 baris, dan setiap blok terdiri atas 8 word. Tunjukkan contoh pemetaannya dalam pemetaan langsung.

Solusi:

Pada Gambar 8.10 ditunjukkan pembagian memori utama dan cache yang menganut teknik pemetaan langsung. Seperti yang terlihat, terdapat total 32 blok memori utama yang dipetakan pada baris memori cache. Dengan menggunakan persamaan $l_{cm} = b_{mm} \text{ modulo } c$, maka nomor blok 0, 32, 64, ..., 2016 dipetakan pada baris 0 di memori cache; nomor blok 31, 63, 94, ... 2047 dipetakan pada baris 31 di memori cache.



Gambar 8.10 Pemetaan langsung blok memori utama ke baris memori cache

Panjang bit-bit field dan jumlah bit alamat memori utama adalah:

- field untuk word = $\log_2 B = \log_2 8 = \log_2 2^3 = 3$ bit
- field untuk blok = $\log_2 N = \log_2 32 = \log_2 2^5 = 5$ bit
- field untuk tag = $\log_2 (M/N) = \log_2 (2^{11}/2^5) = 6$ bit
- Jumlah bit pada alamat memori utama adalah = $\log_2 (B \times M) = \log_2 (2^3 \times 2^{11}) = 14$ bit

Tiga bit LSB digunakan untuk mengakses word (A2 – A0); 5 bit untuk mengakses baris cache (A7—A3) dan sisanya 6 bit (A13—A8) memberikan field TAG nomor blok.

Contoh 8.4

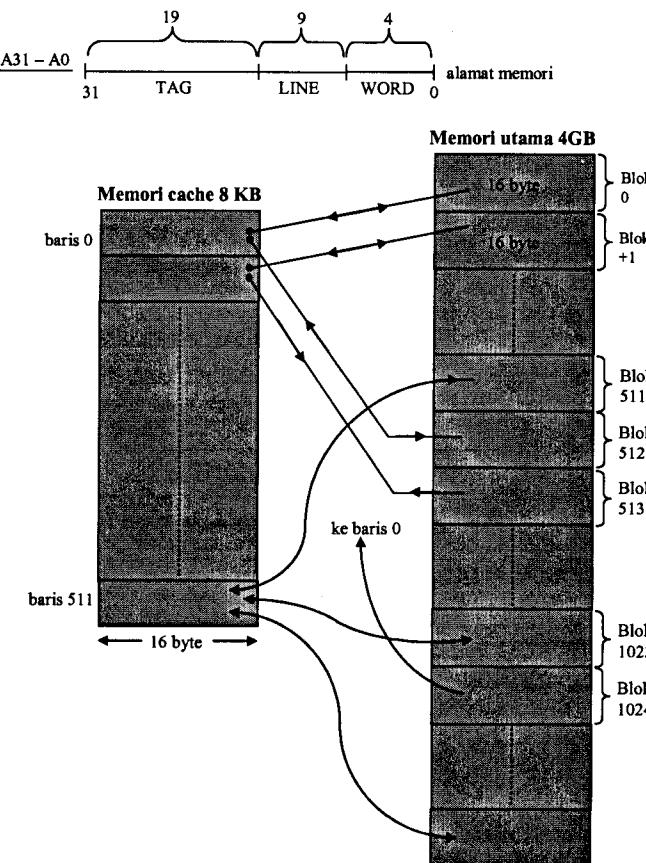
Sebuah komputer 32 bit mempunyai alamat memori 32 bit. Komputer tersebut mempunyai memori cache 8 KB. Komputer mengikuti pemetaan langsung. Setiap baris berukuran 16 byte. Tunjukkan format alamat memori utama dan organisasi memori cache.

Solusi:

Dengan alamat memori 32-bit, prosesor dapat mempunyai memori utama 4GB. Secara logika dibagi menjadi blok-blok 16 byte. Empat LSB dari alamat memori (A0—A3) memberikan nomor word dalam blok, 28 MSB alamat (A31—A4) harus dibagi menjadi field TAG dan field LINE. Panjang field LINE dapat diperoleh pertama karena ukuran memori cache diberikan. Memori cache 8 KB, dibagi dengan ukuran baris (ukuran baris 16 byte),

$$8192/16 = 512$$

Karena jumlah baris dalam memori cache adalah 512, kita memerlukan 9 bit untuk field LINE. Sisanya 19 bit (32 – 4 – 9 = 19) disediakan untuk field TAG. Gambar 8.11 menunjukkan detail pemetaan.



Gambar 8.11 Pemetaan langsung dengan memori utama 4GB dan cache 8 KB

Keuntungan Pemetaan Langsung

Pemetaan langsung merupakan jenis pemetaan yang paling sederhana. Hardware yang diperlukan sangat sederhana karena tag dari baris cache yang hanya satu yang cocok dengan field TAG dalam alamat memori yang diberikan. Pengontrol cache dengan sangat cepat mengeluarkan informasi hit atau miss.

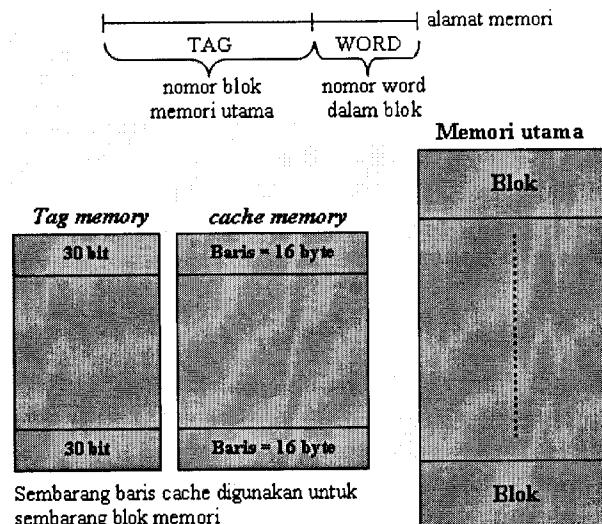
Kekurangan Pemetaan Langsung

Tidak fleksibel dalam sistem pemetaan. Blok memori tertentu dipetakan pada suatu baris cache yang tetap. Jika dua blok yang sering diakses terjadi untuk dipetakan pada baris cache yang sama (karena alamat memorinya), maka hit ratio yang dihasilkan pada penggantian baris cache yang sering adalah kecil. Hal ini memberikan eksekusi program yang lambat. Namun, masalah ini banyak terjadi hanya pada beberapa jenis program.

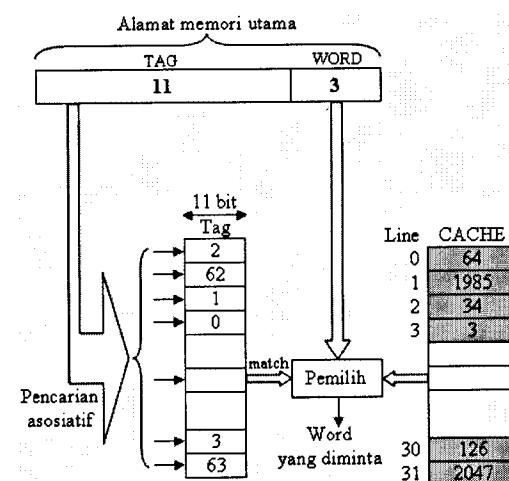
8.6.4 Pemetaan Asosiatif Penuh

Pemetaan asosiatif penuh memperbaiki masalahkekakuan yang terdapat pada pemetaan langsung. Pada pemetaan asosiatif penuh, blok memori dapat dipetakan pada sembarang baris cache. Gambar 8.12 menunjukkan prinsip pemetaan asosiatif penuh. Alamat memori terdiri atas dua field: TAG dan WORD. Field TAG menunjukkan nomor blok memori. Bila prosesor melakukan operasi pembacaan memori, pengontrol cache harus mencocokkan field TAG dalam alamat dengan isi TAG pada semua baris dalam cache. Jika cocok (hit) dengan suatu baris tag, maka blok yang tersimpan dalam baris tersebut dibaca dari cache. Jika tidak cocok (miss) dengan semua baris, pengontrol cache mulai operasi pembacaan memori utama. Blok yang dibaca dari memori utama disimpan pada suatu baris cache jika cache tidak penuh. Jika cache penuh, maka hanya dilakukan penggantian blok yang diperlukan dalam cache. Blok yang harus diganti ditetapkan oleh algoritma penggantian yang digunakan oleh pengontrol cache.

Ilustrasi protokol yang mengatur permintaan word oleh prosesor pada pemetaan asosiatif penuh ditunjukkan pada Gambar 8.13.



Gambar 8.12 Pemetaan asosiatif penuh



Gambar 8.13 Protokol translasi alamat pemetaan asosiatif penuh

Berdasarkan teknik pemetaan asosiatif penuh, MMU (*memory management unit*) melakukan translasi alamat yang diberikan oleh prosesor dengan melakukan pembagian alamat menjadi dua field seperti yang

ditunjukkan pada Gambar 8.12. Panjang bit pada setiap field tersebut adalah:

- field untuk word = $\log_2 B$ di mana B adalah jumlah word dalam blok
- field untuk tag = $\log_2 M$, di mana M adalah jumlah blok dalam memori utama
- Jumlah bit keseluruhan di dalam memori utama adalah = $\log_2 (B \times M)$

Keuntungan Pemetaan Asosiatif Penuh

Pemetaan asosiatif penuh menawarkan fleksibilitas yang tinggi. Suatu blok memori dapat dipindahkan ke sembarang baris cache. Karena itu, penggantian blok cache diperlukan hanya jika cache penuh secara total. Hal ini memberikan kinerja yang lebih baik karena waktu yang digunakan dalam penggantian berkurang. Tentunya ukuran cache juga memengaruhi frekuensi penggantian. Dalam pemetaan asosiatif penuh, jumlah baris dalam cache tidaklah tetap.

Kekurangan Pemetaan Asosiatif Penuh

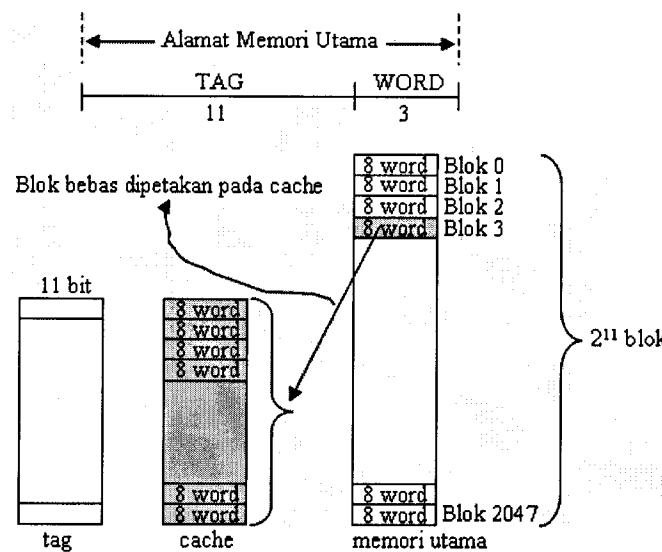
Dalam implementasi, pemetaan asosiatif penuh merupakan sistem yang mahal. Pengontrol cache menjadi kompleks karena tag semua baris cache dicocokkan secara simultan untuk mengurangi delay pengontrol cache. Dengan kata lain, pencarian paralel semua baris cache dilakukan untuk suatu pola TAG khusus. Hal ini disebut pencarian asosiatif penuh. Suatu memori khusus disebut 'memori asosiatif' atau *Content Addressable Memory* (CAM) digunakan untuk tujuan ini.

Contoh 8.5

Misalkan sebuah memori utama mempunyai 2K blok, memori cache 32 baris, dan setiap blok terdiri atas 8 word. Tunjukkan contoh pemetaannya dalam pemetaan asosiatif penuh.

Solusi:

Terdapat total 32 blok memori utama yang dipetakan pada baris memori cache. Pemetaan word dari salah satu blok di memori utama ke baris memori cache dilakukan secara bebas.



Gambar 8.14 Pemetaan asosiatif penuh blok memori utama ke baris memori cache

Panjang bit-bit field dan jumlah bit alamat memori utama adalah:

- field untuk word = $\log_2 B = \log_2 8 = \log_2 2^3 = 3$ bit
- field untuk tag = $\log_2 M = \log_2 2^{11} = 11$ bit
- Jumlah bit pada alamat memori utama adalah = $\log_2 (B \times M) = \log_2 (2^3 \times 2^{11}) = 14$ bit

Tiga bit LSB digunakan untuk mengakses word (A₂ – A₀) dan sisanya 11 bit (A₁₃–A₃) memberikan field TAG nomor blok.

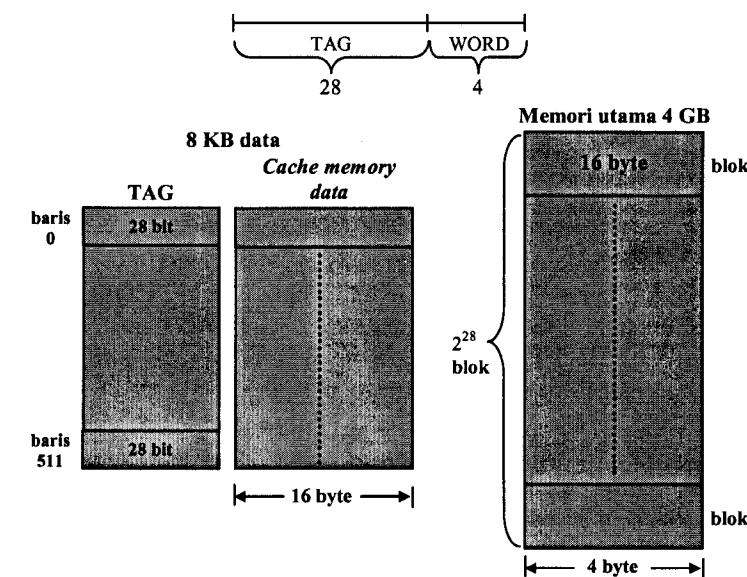
Contoh 8.6

Sebuah komputer 32 bit mempunyai alamat memori 32 bit. Komputer tersebut mempunyai memori cache 8 KB dan mengikuti pemetaan asosiatif penuh. Setiap baris berukuran 16 byte. Tunjukkan format alamat memori utama dan organisasi memori cache.

Solusi:

Empat LSB (A₀–A₃) menetapkan nomor word dalam blok. Sisanya 28-bit (A₃₁–A₄) memberikan field TAG nomor blok. Ada 2²⁸ blok dalam memori. Karena ukuran baris adalah 16 byte, maka memori cache

mempunyai 512 baris (8KB/16B). Gambar 8.15 menunjukkan format alamat memori.



Gambar 8.15 Pemetaan asosiatif penuh—memori utama 4 GB dan memori cache 8 KB

8.6.5 Pemetaan Asosiatif Set

Pemetaan asosiatif set (*set associative mapping*) menggabungkan konsep pemetaan langsung dengan pemetaan asosiatif, untuk memberikan biaya efektif dan kelayakan skema pemetaan yang fleksibel. Jumlah total baris cache dikelompokkan ke dalam set yang banyak. Setiap set mempunyai baris jamak (*multiple line*) dari nomor yang sama. Jika ada i baris dalam seti set, maka sistem pemetaan dikenal dengan pemetaan asosiatif set i -way. Dalam suatu set, blok memori dapat ditempatkan di mana saja dalam i baris. Format alamat memori ditunjukkan pada Gambar 8.16. Format ini mempunyai tiga field: TAG, SET, dan WORD. Khusus field SET menyediakan nomor set.

Formula-formula untuk pemetaan asosiatif set adalah:

$$\begin{aligned} r &= s \times l \\ g &= b \text{ modulo } s \end{aligned}$$

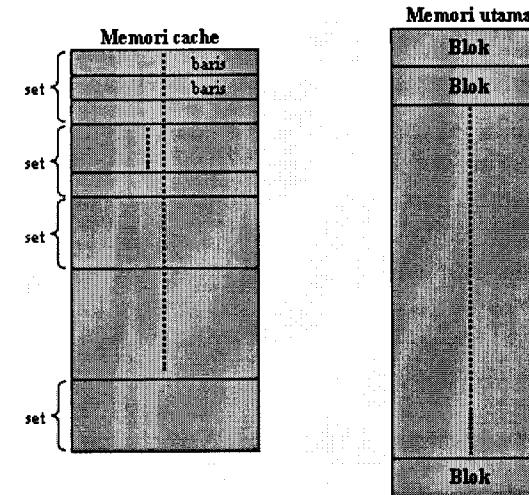
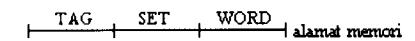
di mana

- r = jumlah total baris memori cache
- s = jumlah set memori cache
- l = jumlah baris di dalam set
- g = nomor set memori cache
- b = nomor blok memori utama

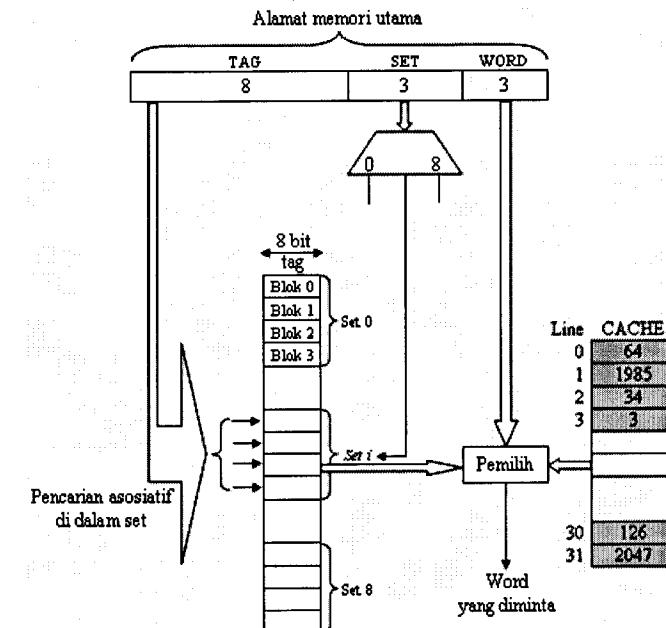
Bila prosesor melakukan operasi pembacaan memori, pengontrol cache menggunakan field SET untuk mengakses cache. Dalam set, ada l jumlah baris. Karena itu field TAG dalam alamat dicocokkan dengan isi tag dalam semua l baris. Pada penerimaan suatu HIT, blok yang bersesuaian dibaca dan word yang diperlukan dipilih. Pada penerimaan MISS, blok dibaca dari memori utama dan disimpan dalam set yang ditentukan sebelumnya. Jika cache (set) penuh, maka penggantian salah satu baris dilakukan seperti pada setiap algoritma penggantian.

Setiap set seperti sebuah memori cache kecil. Jika hanya ada satu set, maka dia sama seperti pemetaan asosiatif. Di sisi lain, jika hanya terdapat satu baris/set, maka dia sama seperti pemetaan langsung.

Ilustrasi protokol yang mengatur permintaan word oleh prosesor pada pemetaan asosiatif set ditunjukkan pada Gambar 8.17.



Gambar 8.16 Pemetaan asosiatif set



Gambar 8.17 Protokol translasi alamat pemetaan asosiatif set

Berdasarkan teknik pemetaan asosiatif set, MMU (*memory management unit*) melakukan translasi alamat yang diberikan oleh prosesor dengan melakukan pembagian alamat menjadi tiga field seperti yang ditunjukkan pada Gambar 8.16. Panjang bit pada setiap field tersebut adalah:

- field untuk word = $\log_2 B$ di mana B adalah jumlah word dalam blok
- field untuk set = $\log_2 S = \log_2 (R/L)$, di mana S adalah jumlah set dalam memori cache, R adalah jumlah baris dalam memori cache dan L adalah jumlah baris setiap set
- field untuk tag = $\log_2 (M/S)$, di mana M adalah jumlah blok dalam memori utama
- Jumlah bit keseluruhan di dalam memori utama adalah = $\log_2 (B \times M)$

Keuntungan Pemetaan Asosiatif Set

- Dibandingkan dengan pemetaan langsung, terdapat pilihan yang banyak dalam pemetaan suatu blok memori. Banyaknya pilihan/opsi bergantung pada ukuran l . Biasanya dalam nomor kecil: 2, 4, 8 dan lain-lain. Jadi menyediakan fleksibilitas yang lebih baik.
- Selama pembacaan, penyesuaian tag (pencarian) terbatas pada jumlah baris dalam set. Karena itu, pencarian hanya dalam suatu set, tidak seperti dalam pemetaan asosiatif, di mana pencarian dilakukan pada semua memori cache.

Kekurangan Pemetaan Asosiatif Set

Dalam implementasinya, biaya lebih mahal daripada pemetaan langsung dan lebih murah daripada pemetaan asosiatif. Namun, dibandingkan terhadap keuntungan pemetaan asosiatif set, biaya ini meningkat tidak relevan.

Contoh 8.7

Misalkan sebuah memori utama mempunyai 2K blok, memori cache 32 baris, dan setiap blok terdiri atas 8 word. Tunjukkan contoh pemetaannya dalam pemetaan asosiatif set dengan jumlah blok dalam setiap set adalah 4.

Solusi:

Seperti yang terlihat pada Gambar 8.17, terdapat total 32 blok memori utama yang dipetakan pada baris memori cache. Pemetaan word dari salah satu blok di memori utama ke salah satu baris di dalam set tertentu di memori cache.

Panjang bit-bit field dan jumlah bit alamat memori utama adalah:

$$S = \frac{32}{4} = 8 \text{ set}$$

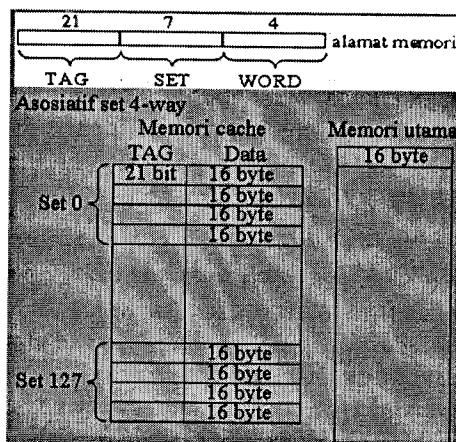
- field untuk word = $\log_2 B = \log_2 8 = \log_2 2^3 = 3$ bit
- field untuk set = $\log_2 S = \log_2 8 = \log_2 2^3 = 3$ bit
- field untuk tag = $\log_2 (M/S) = \log_2 (2^{11}/2^3) = 8$ bit
- Jumlah bit pada alamat memori utama adalah = $\log_2 (B \times M) = \log_2 (2^3 \times 2^{11}) = 14$ bit

Contoh 8.8

Sebuah komputer 32-bit mempunyai alamat memori 32-bit. Komputer tersebut mempunyai memori cache 8 KB dan mengikuti pemetaan asosiatif set 4-way. Setiap baris berukuran 16 byte. Tunjukkan format alamat memori dan organisasi memori cache.

Solusi:

Empat bit alamat A0—A3 untuk nomor word karena ukuran blok adalah 16 byte. Total jumlah baris dalam memori cache adalah $8 \text{ KB}/16 = 512 = 2^9$. Karena kita mempunyai pemetaan asosiatif set 4-way, maka cache dibagi menjadi 128 set ($=2^9/4$). Setiap set mempunyai empat baris. Karena ada 128 set, maka 7 bit dibutuhkan untuk nomor set. Karena itu file SET mempunyai 7 bit. Sisanya 21 bit membentuk field TAG. Gambar 8.18 menunjukkan format alamat dan pemetaan.



Gambar 8.18 Pemetaan asosiatif set 4-way

8.6.6 Penggantian Cache

Bila suatu alamat yang diakses oleh CPU tidak dipetakan dalam memori cache, maka pengaksesan dilakukan langsung ke memori utama. Bersama dengan word yang diperlukan, semua blok ditransfer ke memori cache. Anggap memori cache sudah penuh, maka beberapa isi memori cache dihapus untuk membuat ruang bagi entry baru. Porsi memori cache yang dihapus untuk pembuatan ruang bagi entry baru merupakan hal yang penting. Terdapat sejumlah algoritma khusus untuk memaksimalkan hit rate.

Algoritma Penggantian

Masalah penggantian adalah suatu hal yang penting bila memori cache penuh dan suatu blok baru dari memori utama harus disimpan dalam memori cache. Jelas keberadaan suatu blok dari memori cache harus diganti dengan blok baru. Pada kasus cache pemetaan langsung, kita tidak punya pilihan. Blok baru harus disimpan hanya dalam suatu baris cache yang ditetapkan per aturan pemetaan untuk cache pemetaan langsung. Untuk pemetaan asosiatif dan pemetaan asosiatif set kita memerlukan suatu algoritma yang baik karena kita mempunyai beberapa pilihan. Jika kita menghapus suatu blok yang akan diakses segera oleh CPU, maka terjadi miss penalty. Karena itu, diperlukan suatu keputusan cerdas dari suatu

pemilihan baris cache yang akan dikosongkan. Algoritma penggantian tertentu telah dicoba dan dipelajari efisiensinya. Beberapa di antaranya akan dibahas berikut ini:

Algoritma Random choice: Algoritma ini memilih baris cache secara acak tanpa suatu acuan pada frekuensi penggunaan sebelumnya. Algoritma ini diimplementasikan dengan hardware yang sederhana. Walaupun dapat muncul pada suatu algoritma biasa, dia mempunyai efisiensi praktis yang menarik.

Algoritma First-In First-Out (FIFO): Algoritma ini memilih set yang telah berada pada cache dalam waktu yang lama. Dengan kata lain, blok yang dimasukkan dalam cache pertama didorong keluar pertama. Asumsinya adalah item yang baru saja dimasukkan mungkin yang akan diperlukan.

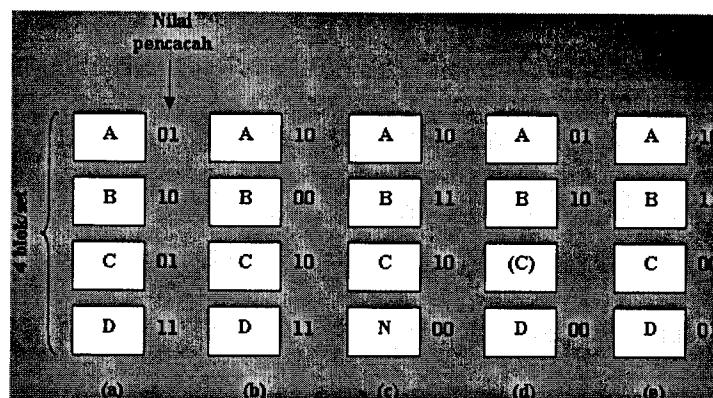
Algoritma Least Frequently Used (LFU): Algoritma ini memilih suatu blok yang telah digunakan oleh CPU yang paling sedikit. Asumsinya adalah cara serupa juga mungkin akan terjadi kemudian.

Algoritma Least Recently Used (LRU): Algoritma ini memilih item yang terlama dalam cache tidak (jarang) digunakan. Asumsinya adalah cara serupa akan berlanjut dan tidak mungkin dibutuhkan segera. Implementasi algoritma ini dapat dilakukan dengan menjalani sejarah penggunaan item oleh CPU. Penggunaan sebuah pencacah untuk setiap baris adalah merupakan suatu metode sederhana pemenuhan persyaratan.

Contoh 8.9

Desain suatu rangkaian untuk mengimplementasikan algoritma LRU penggantian untuk pemetaan asosiatif set 4-way.

- Pikirkan suatu cache asosiatif set 4-way. Gambar 8.19 menganggap sebuah pencacah biner (*binary counter*) dua bit untuk setiap blok dalam set. Kita namakan empat blok ini sebagai A, B, C dan D. Pencacah ('pencacah umur') untuk setiap blok memberikan informasi sejarahnya. Nilai awal ditunjukkan pada Gambar 8.19(a). Nilai pencacah mengindikasikan perluasan akses ke blok yang merepresentasikannya. Jika nilai pencacah = 00, artinya blok yang baru saja diakses. Jika nilai pencacah = 11, artinya blok yang belakangan diakses.



Gambar 8.19 Rangkaian algoritma LRU: (a) kondisi awal; cache penuh (b) 'HIT untuk blok B (c) 'MISS'; mengganti D (d) kondisi awal; cache tidak penuh (e) 'MISS' menduduki C

Kasus 1: Anggap ada sebuah cache hit untuk blok B. Pencacahnya mempunyai 10 sebagai nilai awal. Sekarang pencacahnya reset. Isi pencacah lain dibandingkan dengan nilai awal blok B. Pencacah yang mempunyai nilai lebih kecil (dari pencacah B) dinaikkan 1. Pencacah lain tidak diganggu. Jadi pencacah A dan C dinaikkan sedangkan pencacah D tak disentuh. Status yang diperbarui (update) ditunjukkan pada Gambar 8.19(b).

Kasus 2: Anggap ada sebuah cache miss bila status awal ditunjukkan pada Gambar 8.19(a). Sekarang ada dua kemungkinan: (a) cache penuh (b) cache tidak penuh.

Sekarang pertama kita menganggap keadaan cache penuh seperti pada Gambar 8.19(a). Pada pemeriksaan nilai semua pencacah, teramatinya bahwa D mempunyai nilai tertinggi 11. Sehingga D adalah blok yang belum terakses pada masa lalu. Jadi D harus diganti (disimpan ke memori utama) dan nilai item baru masuk ke sana. Pencacahnya dibuat 00. Pencacah lainnya dinaikkan 1. Nilai yang diperbarui diperlihatkan pada Gambar 8.19(c).

Selanjutnya mari kita menganggap kasus bila cache tidak penuh seperti ditunjukkan pada Gambar 8.19(d). Item baru dapat disimpan dalam suatu baris kosong dan pencacahnya dibuat 0. Pencacah lainnya dinaikkan 1. Gambar 8.19(e) menunjukkan status yang diperbarui.

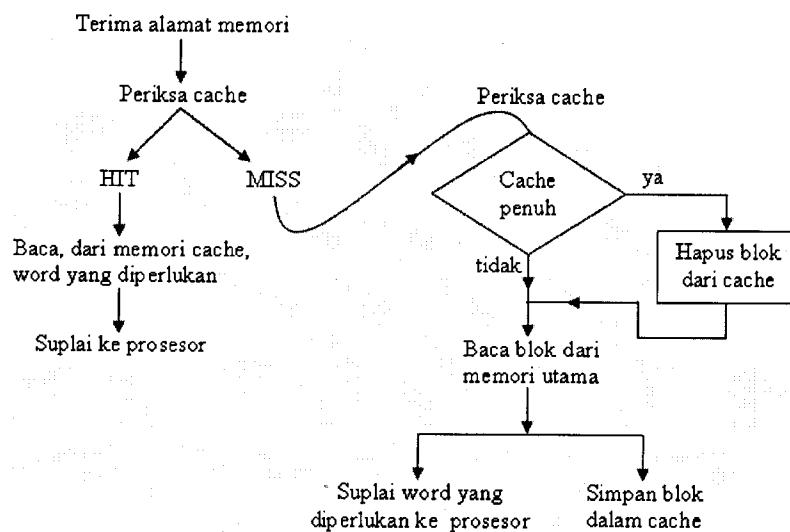
8.6.7 Kebijakan Penulisan Cache

Anggap CPU sedang mengeksekusi instruksi ADD yang hasilnya disimpan pada lokasi memori utama. Apakah hasilnya disimpan pada memori utama atau memori cache? Jika alamat hasil (operand) tidak sedang dipetakan dalam cache, maka jelas hasil akan disimpan di dalam memori utama. Di sisi lain, jika alamat pada saat sekarang dipetakan dalam cache, terdapat dua opsi:

1. Penulisan hasil dalam memori utama dan memori cache. Kebijakan ini dikenal sebagai *write-through policy*.
2. Penulisan hasil hanya dalam memori cache, tetapi ditandai sebuah flag dalam cache untuk mengingatkan isi memori utama yang bersesuaian adalah usang. Ke depan, kapan saja isi dalam memori cache akan diganti, cukup jika disimpan dalam memori utama pada saat itu. Kebijakan ini dikenal sebagai *write-back policy*.

Write-Through Policy

Write-through policy sederhana dan implementasinya tidak bertele-tele, tetapi memerlukan satu akses memori utama setiap saat ada keperluan penulisan. Karenanya, hal ini memperlambat eksekusi program. Juga kadang-kadang terjadi bahwa apa yang disimpan dalam memori utama hanyalah suatu hasil sementara (*intermediate result*) yang segera dibawa ke dalam cache atau diperbarui. Sehingga waktu yang tidak dibutuhkan digunakan dalam penyimpanan item tersebut dalam memori utama. Namun, frekuensi penulisan yang sia-sia tersebut bergantung pada tingkah laku program aplikasi.

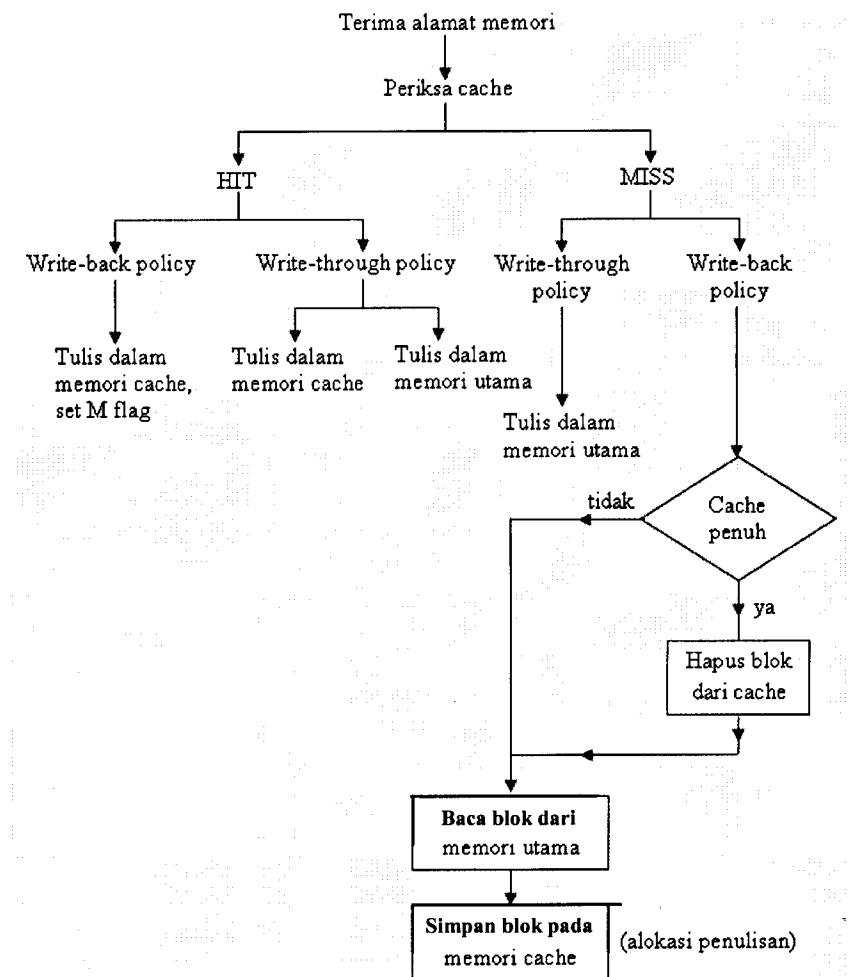


Gambar 8.20 (a) Urutan pembacaan memori

Write-Back Policy

Write-back policy menghemat waktu karena penulisan dalam memori utama tidak dilakukan pada setiap waktu operasi penulisan. Item tertentu mungkin tidak pernah ditulis dalam memori utama di mana bergantung pada instruksi selanjutnya. Hanya bila item harus dihapus dari cache, dia ditulis ke dalam memori utama. Apakah penghapusan akan terjadi atau tidak, tidak diketahui saat permulaan. Bergantung pada instruksi-instruksi selanjutnya.

Gambar 8.20(a) mengilustrasikan urutan operasi yang dilakukan oleh pengontrol cache untuk suatu operasi pembacaan memori oleh CPU. Gambar 8.20(b) mengilustrasikan urutan operasi penulisan memori oleh CPU.



Gambar 8.20(b) Urutan penulisan memori

8.6.8 Pengintaian dan Invalidasi

Ada kemungkinan lokasi memori utama diperbarui tanpa pengetahuan prosesor (dan memori cache). Contoh khas adalah operasi input dari hard disk ketika pengontrol DMA menyimpan data dalam memori utama. Apa yang terjadi bila pengontrol DMA menulis dalam suatu lokasi memori yang telah dipetakan dalam cache? Anggap bahwa alamat memori utama 1000

berisi karakter M. Karena alamat 1000 telah dipetakan dalam memori cache, maka data ini juga tersedia pada salah satu baris cache. Sekarang anggap pengontrol DMA menulis data hard disk karakter C pada alamat memori utama 1000. Jika diizinkan seperti itu, maka lokasi memori utama akan berisi C sedangkan lokasi cache yang sesuai akan berisi M. Karena ketidaksesuaian (*mismatch*) tidak dapat diizinkan antara memori utama dan memori cache, maka suatu logika pemantau bus memeriksa aktivitas memori (bus) oleh pengontrol DMA (atau prosesor lainnya jika ada). Hal ini dikenal sebagai *snooping* (pengintaian). Pada pendekripsi permulaan dari suatu urutan penulisan memori untuk suatu alamat yang telah dipetakan dalam memori cache, maka dengan segera pengontrol cache bersiaga menanyakan untuk membatalkan data dalam memori cache. Untuk maksud ini, alamat memori dikirim ke pengontrol cache. Pengontrol cache men-set INVALID flag untuk entry ini. Anggap kemudian CPU telah melakukan operasi baca memori untuk alamat 1000. Akan ada cache hit untuk alamat tersebut, tetapi pengontrol cache tidak menggunakan data cache yang terlihat pada *invalid flag status*. Malahan dia akan mengambil data baru dari memori utama dan me-load ke dalam cache selain mensuplainya ke dalam CPU.

Pembilasan Cache

Segara setelah power-on, tak ada satupun dalam memori cache yang valid karena tidak ada program yang telah dikerjakan CPU. Karena itu untuk semua baris cache, invalid flag bit harus di-set oleh pengontrol cache. Dengan cara yang sama perangkat eksternal dapat meminta CPU untuk melakukan invalidasi (*invalidation*) semua baris cache. Pembatalan/invalidasi massal yang demikian dikenal sebagai pembilasan cache (*cache flushing*). Beberapa CPU mempunyai instruksi FLUSH selama pengeksekusian operasi flush tersebut dilakukan.

8.6.9 Kinerja Cache dan Hit Rate

Desain memori cache dilakukan karena pertimbangan biaya dan kinerja. Parameter-parameter utama desain memori cache adalah:

1. Kapasitas memori cache
2. Ukuran baris cache (ukuran blok cache)
3. Ukuran set

Kapasitas Memori Cache: Peningkatan biaya sebanding dengan kapasitas. Dalam kenyataan, ada biaya tambahan yang tak terlihat karena pendekodean dan sirkuit kontrol lainnya. Karena itu, diperlukan untuk menjaga kapasitas memori cache sekecil mungkin.

Ukuran Baris Cache: Jika ukuran baris besar, membantu mengurangi jumlah pengaksesan memori utama. Tetapi ukuran baris yang besar mengurangi jumlah blok yang dapat dimuat dalam memori cache. Karena itu ukuran sebaiknya dipilih setelah studi yang hati-hati dari tingkah laku sifat untuk berbagai ukuran blok.

Ukuran Set: Jika terdapat baris yang lebih setiap set, hit rate menjadi lebih baik. Tetapi ukuran set yang terlalu besar meningkatkan biaya karena sirkuit bertambah untuk kesesuaian tag.

Hit Rate

Hit rate idealnya 1 agar setiap pengaksesan memori utama oleh prosesor dilayani oleh memori cache dan tidak ada akses memori utama yang diperlukan secara keseluruhan. Secara praktis, tidak mungkin hal ini tercapai dan karena itu dicoba hit rate mendekati 1. Desain yang sangat buruk menghasilkan hit rate mendekati 0. Ini artinya pengaksesan memori utama sangat banyak. Hit rate 0 artinya semua hasil pengaksesan memori utama oleh prosesor adalah MISS. Walaupun parameter-parameter desain tertentu membantu dalam pencapaian hit rate mendekati 1, tingkah laku program aplikasi dapat dengan mudah mengganggu hit rate.

Mari kita menganggap parameter-parameter berikut:

- h – hit rate
- $t_{(mm)}$ – waktu akses memori utama
- $t_{(cm)}$ – waktu akses memori cache
- $t_{(bm)}$ – waktu untuk mentransfer suatu blok dari memori utama ke memori cache

Kinerja sistem rata-rata diberikan oleh kebalikan dari waktu siklus instruksi rata-rata. Waktu siklus instruksi terdiri atas dua komponen:

1. Waktu aktivitas *cache independent* selama siklus instruksi
2. Waktu pengaksesan *cache dependent* (pengaksesan memori utama)

Komponen pertama bergantung pada clock internal prosesor dan organisasi hardware. Komponen kedua bergantung pada hit rate. Dengan mengabaikan komponen pertama untuk waktu yang ada, maka waktu rata-rata yang digunakan oleh prosesor selama pengaksesan memori diberikan oleh:

$$t(\text{cm}) + (1-h) \times t(\text{bm})$$

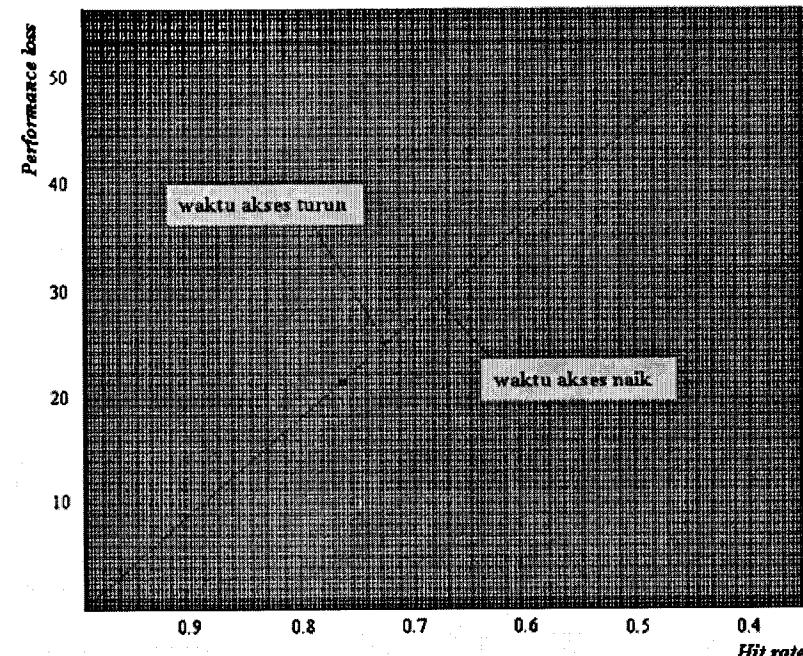
Jika ukuran blok kecil, $t(\text{bm})$ adalah kecil. Prosesor-prosesor saat ini, mentransfer semua blok dalam satu siklus bus. Misalnya, mikroprosesor 80486 melakukan *burst bus cycle* untuk tujuan ini. Mikroprosesor Pentium menggandakan lebar datapath eksternalnya untuk tujuan yang sama. Karena itu kita dapat mengaproksimasi bahwa $t(\text{bm})$ sama dengan $t(\text{mm})$ dan mengemukakan kembali waktu akses rata-rata $t(\text{ma})$ sebagai berikut:

$$t(\text{ma}) = t(\text{cm}) + (1-h) \times t(\text{mm})$$

Diasumsikan memori cache lima kali lebih cepat dari memori utama,

$$t(\text{ma}) = t(\text{cm}) + (1-h) \times 5 \times t(\text{cm})$$

Perhitungan untuk hit rate 0.98; 0.97; 0.96 kita peroleh nilai $t(\text{ma})$ berturut-turut adalah 1.1 $t(\text{cm})$, 1.15 $t(\text{cm})$ dan 1.2 $t(\text{cm})$. Hal ini menunjukkan bahwa pengurangan hit rate sedikit meningkatkan waktu eksekusi program yang besar. Gambar 8.21 menunjukkan hubungan antara hit rate dan kinerja. Proses desain komputer melibatkan simulasi tingkah laku sistem sebelum penyelesaian parameter-parameter cache.



Gambar 8.21 Hit rate vs kinerja

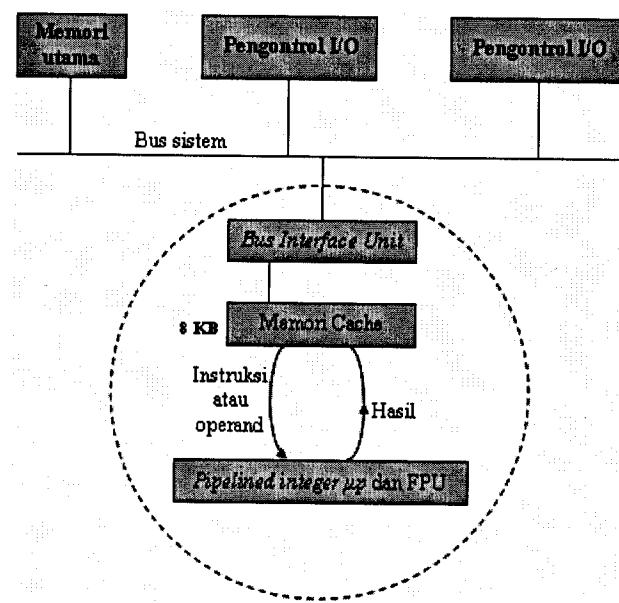
8.6.10 Cache Gabung dan Cache Pisah

Memori cache ada dua jenis:

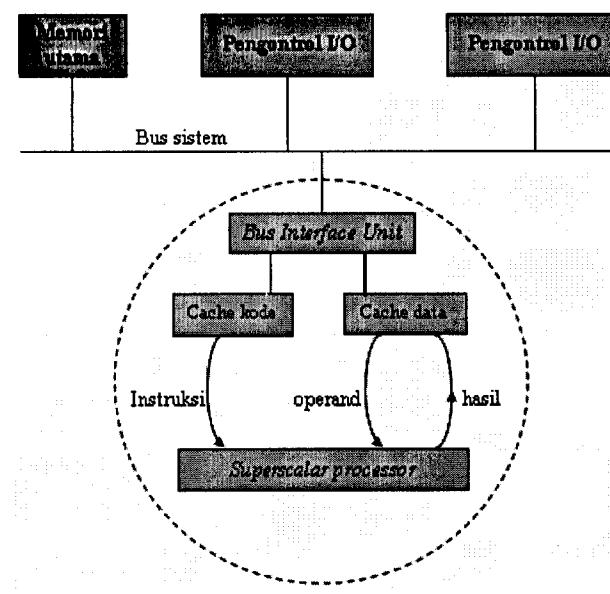
1. Cache-gabung (*unified cache*) yang menyimpan instruksi dan data. Mikroprosesor INTEL 80486 mempunyai cache-gabung seperti yang ditunjukkan pada Gambar 8.22.
2. Cache-pisah (*split cache*) yang mempunyai cache data dan instruksi yang terpisah. Mikroprosesor INTEL Pentium mempunyai cache-pisah yaitu 8 KB untuk instruksi dan 8 KB untuk data seperti yang ditunjukkan pada Gambar 8.23.

Cache-gabung mudah diimplementasikan tetapi menawarkan hit rate yang kecil secara khusus pada CPU pipeline di mana akses memori jamak mungkin diperlukan pada suatu waktu dengan tingkat berbeda seperti pengambilan instruksi, pengambilan operand, eksekusi, dan penyimpanan hasil. Bahkan pada CPU sederhana dengan prefetch instruksi, bila CPU ingin mengambil operand dari memori utama atau menyimpan hasil dalam

memori utama, anggap antrian instruksi kosong menyebabkan instruksi diambil dari memori. Memori cache-gabung mengawasi beberapa permintaan tunda, menyebabkan delay, hingga akses yang sekarang selesai. Cache-pisah menyediakan paralelisme. Keduanya dapat diakses secara paralel. Hal ini mengurangi waktu tunggu dan meningkatkan kinerja.



Gambar 8.22 Arsitektur cache-gabung pada mikroprosesor 80486

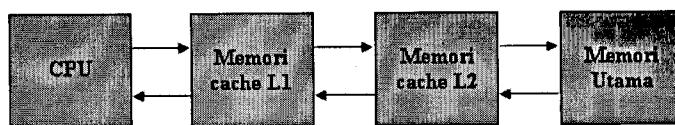


Gambar 8.23 Arsitektur cache-pisah pada Pentium

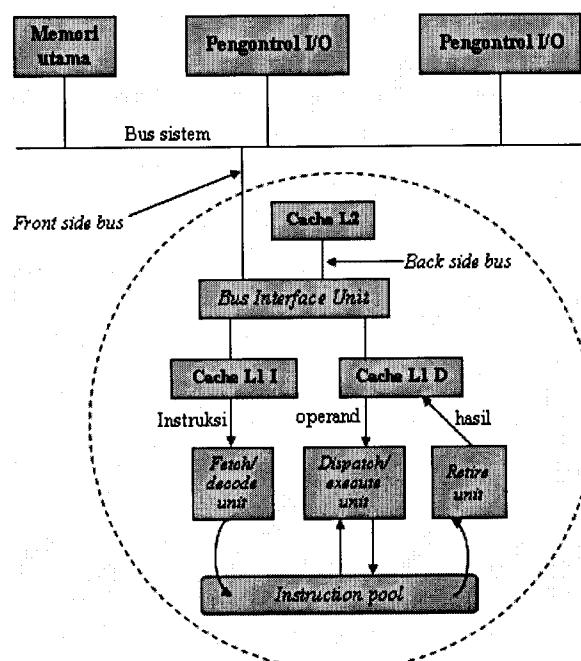
8.6.11 Cache Multi-Level

Bila ukuran memori utama besar, beberapa komputer menggunakan sistem memori cache dua level atau tiga level. Cache yang letaknya terdekat dengan prosesor disebut cache level 1 (L1) atau cache primer. Cache level berikutnya disebut cache level 2 (L2) atau cache sekunder. Pada umumnya mikroprosesor menggabungkan caches multi-level dalam satu chip. Gambar 8.24(a) menunjukkan konsep cache dua level dan Gambar 8.24(b) menunjukkan arsitektur cache dua level pada mikroprosesor Pentium Pro. Pentium Pro mengikuti arsitektur dual bus seperti yang ditunjukkan pada Gambar 8.25. Terdapat *dedicated bus* antara prosesor dan cache L2 yang dikenal dengan *backside bus* (bus belakang). *Front side bus* (bus depan) adalah bus sistem. Arsitektur *dual bus* memberikan dua keuntungan:

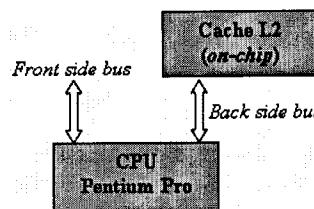
1. Paralelisme: transfer simultan pada kedua bus.
2. Komunikasi kecepatan tinggi antara CPU dan cache L2 karena dedicated bus. Hal ini tidak mungkin jika cache L2 juga terhubung ke bus sistem, karena keterbatasan bus sistem.



Gambar 8.24(a) Memori cache dua level



Gambar 8.24(b) Arsitektur cache dua level pada Pentium Pro

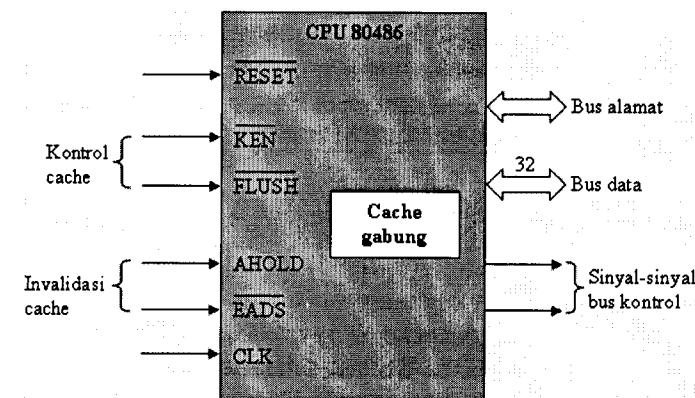


Gambar 8.25 Dedicated cache bus

Mikroprosesor INTEL 80286 dan 80386 mengusung memori cache eksternal tetapi tidak mempunyai memori cache internal. Mikroprosesor 80486 dan mikroprosesor INTEL berikutnya menggabungkan memori cache yang on-chip.

8.6.12 Cache Mikroprosesor 80486

Mikroprosesor 80486 mempunyai cache-gabung on-chip 8 KB untuk kode (instruksi) dan data secara bersama. Mikroprosesor 80486 mengikuti pemetaan asosiatif set 4-way. Memori cache disusun 128 set dan setiap set berisi empat baris. Setiap baris lebarnya 16 byte. Secara fisik, memori cache dibagi menjadi empat modul masing-masing 2 Kbyte. Setiap blok berisi 128 baris. Pada setiap blok terdapat 128 tag dari 21 bit. Cache mikroprosesor 80486 mengikuti strategi 'write-through'. Gambar 8.26 menunjukkan sebagian pin mikroprosesor 80486.



Gambar 8.26 Konfigurasi fungsional sebagian pin Mikroprosesor 80486

Kontrol Cache

Bit-bit CD dan NW pada register kontrol CRO menyediakan kontrol cache. Bit CD men-enable atau men-disable cache. Bit NW mengontrol 'write-through' memori dan melakukan invalidasi. Ada empat mode pengoperasian cache yang ditetapkan oleh kombinasi bit CD dan NM seperti yang ditunjukkan pada Tabel 8.1.

TABEL 8.1 Kombinasi bit-bit CD dan NM

CD	NM	Mode operasi
0	0	Isi Cache di-enable; <i>write-through</i> dan invalidasi di-enable; mode ini adalah mode operasi normal
0	1	Kombinasi ini tidak sah/invalid karena menyatakan secara tak langsung <i>non-transparent write-back cache</i> sedangkan cache 80486 adalah <i>write-through cache</i> ; pola ini jika di-load akan menghasilkan pembangkitan <i>general protection fault</i> dengan kode error 0.
1	0	Isi cache di-disable tetapi <i>write-through</i> dan invalidasi di-enable; mode ini digunakan oleh software untuk men-disable cache untuk interval pendek dan kemudian meng-enable-nya tanpa flushing isi semula (<i>original</i>).
1	1	Pada mode ini isi cache di-disable; <i>write-through</i> dan invalidasi juga di-disable, cache di-disable secara sempurna dengan men-set bit CD dan NM menjadi 1 dan kemudian pembilasan cache (program-program diagnostik menggunakan mode ini untuk memantau semua siklus memori secara eksternal). Jika cache tidak dibilas selama operasi pembacaan memori, cache hit dapat terjadi dan data (data salah) akan dibaca dari cache.

Kandungan Baris Cache

Area memori utama dapat di-cache dalam 80486. Namun, software dan hardware eksternal dapat menetapkan beberapa porsi memori sebagai memori yang tak dapat di-cache (*non-cacheable*). Software mencegah *chaching page* dengan men-set bit PCD dalam *page table entry*. Hardware eksternal membuat pin KEN (*cache enable*) tidak aktif, selama pengaksesan memori dinformasikan ke 80486 bahwa alamat memori tidak dapat di-cache.

Ketika mikroprosesor melakukan operasi pembacaan, jika alamat ada (dipetakan) dalam cache, maka terjadi cache hit dan data akan disuplai dari on-chip cache. Jika alamat tidak ada dalam cache, mikroprosesor harus melakukan operasi pembacaan memori eksternal. Jika alamat mempunyai porsi memori yang dapat di-cache/cacheable, 80486 melakukan pengisian baris cache. Selama pengisian baris, 16 byte dibaca kedalam 80486 dan terjadi pengisian baris.

Pengisian cache hanya terjadi untuk *cache-miss* selama operasi baca. Jika cache-miss terjadi selama operasi tulis, tidak terjadi perubahan selama cache diperhatikan. Jika terjadi cache hit selama operasi penulisan memori, maka baris diperbarui.

Invalidasi Baris Cache

Invalidasi baris cache diperlukan karena untuk mengindikasikan bahwa isi cache internal tertentu berbeda dari isi memori eksternal sehingga ini tidak digunakan oleh 80486. Hardware mempunyai suatu mekanisme untuk memeriksa operasi penulisan untuk memori eksternal oleh unit-unit lain. Bila prosesor lain atau pengontrol DMA menulis pada suatu bagian memori eksternal yang dipetakan dalam cache internal, maka 80486 melakukan invalidasi isi internal yang bersangkutan.

Ada dua langkah selama siklus invalidasi. Pertama, hardware eksternal mengaktifkan sinyal AHOLD (*address hold*). Untuk merespons hal ini, 80486 dengan segera melepaskan bus alamat. Selanjutnya, perangkat eksternal mengaktifkan sinyal EADS (*valid external address*) yang mengindikasikan bahwa suatu alamat valid berada pada bus alamat 80486. 80486 membaca alamat dari pin alamat, kemudian memeriksa apakah alamat ini ada di dalam cache internal. Jika alamat ada dalam cache internal, maka entry cache dibatalkan (di-invalidasi).

Umumnya hardware eksternal mengaktifkan sinyal EADS bila external master menyampaikan suatu alamat pada bus alamat. Jika EADS sendiri yang disampaikan tanpa ada sinyal AHOLD, maka 80486 sendiri yang mengeluarkan alamat invalidasi.

Bit valid untuk setiap baris mengindikasikan apakah suatu baris valid atau non-valid. Semua bit valid di-clear bila 80486 reset atau bila cache dibilas.

Penggantian Cache

Bila 80486 menambahkan baris dalam cache internal, pertama dia memeriksa apakah ada baris invalid dalam set. Jika terdapat baris invalid dalam set (keluar dari empat baris), maka baris invalid diganti dengan baris yang baru. Jika semua empat baris dalam set adalah valid, maka mekanisme pseudo terkecil yang terakhir yang digunakan untuk memperoleh baris yang akan diganti.

Pembilasan Cache

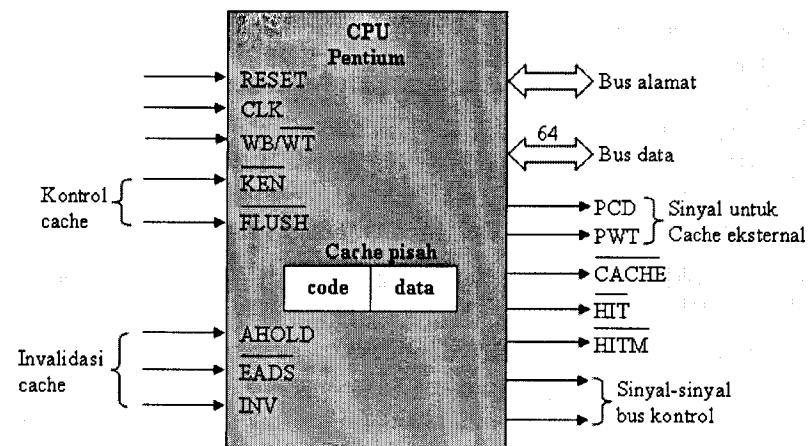
On-chip cache dapat dibilas dengan hardware atau software eksternal. Pembilasan cache menghasilkan pembersihan semua bit valid untuk semua baris dalam cache.

Hardware eksternal mengaktifkan pin FLUSH untuk pembilasan cache. Perangkat lunak menggunakan instruksi INVD atau WBINVD untuk membilas cache. Bila instruksi-instruksi ini dieksekusi, maka cache eksternal yang terhubung ke 80486 juga diisyaratkan untuk dibilas isinya. Untuk instruksi WBINVD, cache write-back eksternal akan menulis kembali baris 'dirty' dengan flag termodifikasi (dalam memori utama) sebelum membilas isinya. Cache eksternal diisyaratkan menggunakan pin-pin yang ditetapkan siklus bus dan byte enable. Selama cache internal diperhatikan, pengaruh instruksi INVD dan WBINVD adalah sama.

8.6.13 Invalidasi Cache pada Pentium

Gambar 8.27 menunjukkan sebagian pin dari prosesor Pentium. Urutan invalidasi cache Pentium lebih terlibat daripada 80486; Mikroprosesor Pentium mengikuti write-back policy. Invalidasi cache dilakukan dengan mengikuti dua tahap:

1. Hardware eksternal mengirim sinyal AHOLD ke Pentium dan dalam respons, Pentium melepaskan bus alamat dengan membuat pin alamat dalam keadaan tristate. Kemudian hardware eksternal meletakkan alamat memori pada pin-pin ini dan menginformasikannya ke Pentium dengan menggunakan sinyal EADS.
2. Setelah itu, Pentium melakukan siklus 'inquire' (bertanya) dan mengindikasikan hasilnya menggunakan sinyal HIT dan HITM. HIT mengindikasikan status 'hit' atau 'miss'. HITM melaporkan bahwa hit adalah untuk memodifikasi baris dalam cache data yang dinyatakan secara tak langsung bahwa isi suatu entri cache belum tersimpan pada lokasi memori utama yang sesuai). Sekarang hardware eksternal harus melakukan aksi yang tepat. Penyampaian sinyal INV meng-invalidasi baris cache. Pada kasus 80486, tak ada siklus bertanya dan sinyal EADS diterima, 80486 mulai melakukan invalidasi.



Gambar 8.27 Konfigurasi fungsional sebagian pin mikroprosesor Pentium

8.7 MEMORI VIRTUAL

Memori virtual diperlukan pada dua kasus dalam mengeksekusi program-program besar yang mempunyai ukuran melebihi ukuran fisik memori:

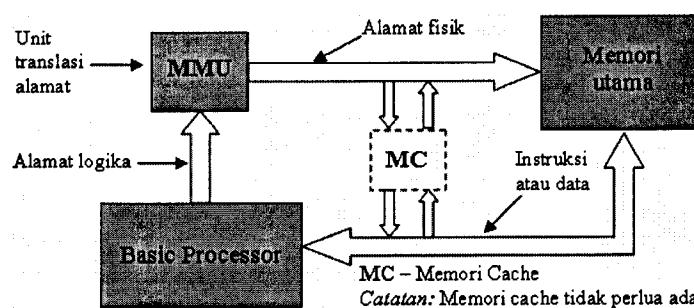
1. Ruang memori utama prosesor tidak cukup untuk menjalankan program besar.
2. Ukuran fisik memori utama dibiarkan kecil untuk mengurangi biaya walaupun prosesor mempunyai ruang memori logik yang besar.

Dahulu, kasus pertama bertanggung jawab untuk mempunyai memori virtual karena komputer pada waktu itu menawarkan ruang memori utama yang terbatas. Tetapi, sekarang ruang memori besar disediakan oleh komputer, karena itu kasus kedua yang merupakan alasan untuk mempunyai memori virtual. Karena penggunaan yang luas dalam beberapa bidang, program-program aplikasi sekarang lebih panjang dibandingkan sebelumnya.

Dalam sistem memori virtual, sistem operasi secara otomatis mengatur program-program yang panjang. Program pemakai dapat lebih besar daripada kapasitas fisik memori utama. Sistem operasi menyimpan semua

program pada harddisk yang mempunyai kapasitas yang lebih besar daripada ukuran memori fisik. Dalam satu waktu, hanya beberapa bagian program yang dibawa dari harddisk ke dalam memori utama. Pada saat diperlukan, bagian yang tidak berada dalam memori utama dikirim dari hard disk, dan pada saat yang sama, bagian dari suatu program yang berada dalam memori utama dikeluarkan dari memori utama dan disimpan pada hard disk. Proses ini dikenal sebagai “swapping” (pertukaran). Dalam sistem memori virtual, bila suatu program dieksekusi, pertukaran dilakukan setiap diperlukan pada suatu basis kontinu.

CPU mengambil instruksi dan data dari memori utama. Kapanpun diperlukan instruksi atau data yang sekarang tidak tersedia dalam memori utama, maka hardware membangkitkan interupsi khusus dikenal sebagai *virtual memory interrupt* atau *page fault*. Ketika direspon, sistem operasi me-load bagian program (berisi instruksi atau data yang diperlukan) dari hard disk ke memori utama. Page fault mirip dengan interupsi tetapi dia dapat terjadi dalam suatu siklus instruksi. Setelah interupsi page fault dilayani, CPU akan melanjutkan pemrosesan instruksi yang dieksekusi secara parsial. Untuk mendukung memori virtual, program tidak langsung mengalami memori fisik. Bila membaca operand (data) atau instruksi, dia menyediakan alamat logika dan hardware memori virtual menerjemahkannya menjadi alamat memori fisik yang ekivalen selama eksekusi. Gambar 8.28 mengilustrasikan konsep memori virtual. Translasi (penerjemahan) alamat dilakukan oleh *memory management unit* (MMU).



Gambar 8.28 Konsep memori virtual

Ada beberapa metode yang digunakan untuk translasi alamat atau pemetaan. Tiga metode yang populer adalah *Associative Mapping Scheme*,

Mapping by address Scheme dan *Segment Map Table Scheme*. Pada *Associative Mapping Scheme*, digunakan suatu memori khusus yang dikenal dengan ‘associative memory’ atau ‘content address memory’. Setiap entry pada memori ini berisi alamat logika dan alamat fisik ekivalen. Pengalaman memori ini dilakukan menggunakan alamat logika.

8.7.1 Keuntungan Memori Virtual

1. Ukuran program tidak dibatasi oleh ukuran memori fisik
2. Pemakaian tidak perlu mengestimasi alokasi memori. Alokasi memori dilakukan secara otomatis sesuai permintaan program.
3. *Manual folding (overlay)* dieliminasi untuk menjalankan program-program besar.
4. Program dapat di-load dalam suatu area memori fisik karena program tidak menggunakan alamat fisik.

8.7.2 Mekanisme Memori Virtual

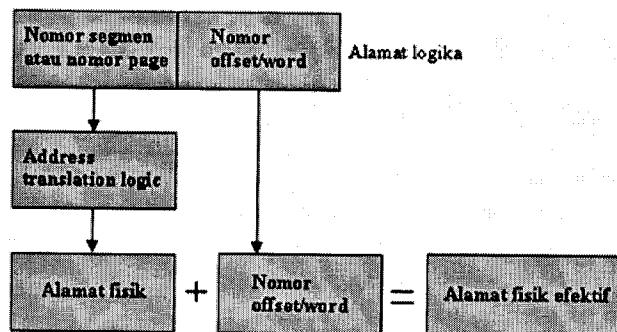
Ruang alamat CPU adalah rentang alamat valid yang dapat dibangkitkan oleh set instruksinya. Jumlah bit dalam register alamat memori menentukan ukuran maksimum ruang alamat. Alamat-alamat fisik merupakan lokasi-lokasi aktual dalam memori utama. Alamat logika adalah spesifikasi memori yang digunakan oleh CPU. Pada kebanyakan komputer, alamat logika dan alamat fisik adalah sama. Pada sistem memori virtual, alamat logika berbeda dengan alamat-alamat fisik aktual.

Alamat logika mempunyai dua bagian. Bagian pertama menentukan nomor modul program (nomor segmen atau nomor page). Bagian kedua menyediakan offset atau nomor word pada modul program tersebut yang menghubungkan ke awal modul tersebut. Sebuah tabel diberikan dalam hardware untuk menujukkan porsi (bagian atau modul) program yang tersedia sekarang dalam memori utama. Tabel ini berisi dua informasi dasar:

1. Porsi (bagian-bagian) program yang ada sekarang dalam memori utama.
2. Area memori utama di mana bagian program yang tersedia sekarang disimpan.

Kapanpun suatu instruksi atau operand harus diakses, CPU mencarinya di dalam tabel untuk menemukan apakah dia tersedia dalam memori utama atau tidak. Jika tersedia, maka MMU mengubah alamat logika menjadi alamat fisik ekivalen dan kemudian melakukan akses memori utama. Sebaliknya, jika instruksi atau operand tidak tersedia dalam memori utama maka interupsi dibangkitkan. Hal ini meminta sistem operasi melakukan swapping atau pertukaran. Bila terjadi swapping, sistem operasi meng-update tabel ini. Pada saat ini translasi alamat dilakukan dan memori alamat diakses.

Karena program mengalami operand atau instruksi menggunakan alamat logika, maka hardware memori utama mentraslasikan alamat logika menjadi alamat fisik sebelum pengeksekusian. Gambar 8.29 menunjukkan prinsip translasi alamat.



Gambar 8.29 Mekanisme memori virtual

Ada dua metode yang populer dalam implementasi memori virtual:

1. *Paging*
2. *Segmentation*

Pada metode paging, software sistem membagi program menjadi sejumlah page. Ini tidak dikenal pada sisi pemrogram. Semua page dalam suatu program berukuran sama. Dalam satu waktu, hanya beberapa page yang ada dalam memori utama. Pada sistem multiprogramming, dalam satu waktu, beberapa page dari program-program yang berbeda tersedia dalam memori utama. Sisa page dari program-program yang berbeda disimpan pada hard disk yang digunakan sebagai memori virtual.

Pada metode segmentasi, pemrogram (bahasa mesin) menyusun program ke dalam segmen-segmen berbeda. Ini adalah pembagian logika program menjadi modul-modul yang berguna. Segmen-segmen berbeda tidak mempunyai ukuran yang sama.

IBM 360/67 dan Control Data Cyber 180 adalah beberapa pendahulu dari sistem komputer yang menawarkan memori virtual. Mikroprosesor INTEL 80286 selanjutnya mengusung memori virtual. Mikroprosesor 80286 hanya mengikuti skema segmentasi sementara mikroprosesor 80386 mengadopsi skema paging dan skema segmentasi.

8.7.3 Paging

Alokasi memori fisik dibagi menjadi page frame yang ukurannya tetap, di mana sejumlah page proses disimpan dengan merekam informasi pada sebuah page table. Setiap proses mempunyai page table sendiri yang secara khas tersimpan pada memori utama, dan page table menyimpan lokasi fisik masing-masing virtual page proses. Page table mempunyai N baris, di mana N adalah jumlah virtual page dalam proses. Jika ada page yang prosesnya sekarang tidak berada dalam memori utama, maka page table mengindikasikannya dengan me-reset sebuah valid bit menjadi "0"; jika page ada di dalam memori utama, maka valid bit di-set menjadi "1". Karena itu setiap entry dari page table mempunyai dua field yaitu valid bit dan frame number.

Field-field tambahan sering ditambahkan untuk me-relay informasi lainnya. Misalnya, *dirty bit* (atau *modify bit*) dapat ditambahkan untuk mengindikasikan apakah page sudah diubah. Hal ini membuat pengembalian page ke disk lebih efisien, karena jika dia tidak diubah, maka tidak diperlukan penulisan kembali ke disk. Bit lainnya (*usage bit*) dapat ditambahkan untuk mengindikasikan penggunaan page. Bit ini di-set ke "1" jika page diakses. Setelah periode waktu tertentu, *usage bit* di-reset ke "0". Jika page diacu kembali, maka *usage bit* di-set ke "1". Namun, jika bit tetap "0", ini mengindikasikan bahwa page belum pernah digunakan dalam periode waktu tertentu, dan sistem bisa beruntung dengan mengirimkan page ini ke disk.

Page memori virtual ukurannya sama dengan page frame memori fisik. Memori proses yang dibagi menjadi page-page ukuran tetap seperti ini, dihasilkan dalam fragmentasi internal potensial ketika page terakhir disalin

ke memori. Proses sebenarnya tidak dapat menggunakan page frame keseluruhan, tetapi tidak ada proses lain yang bisa menggunakannya. Karena itu, memori yang tidak digunakan pada frame terakhir ini secara efektif terbuang. Ini bisa terjadi bahwa proses itu sendiri memerlukan sedikitnya satu page dalam keseluruhan, tetapi dia harus menempati sebuah page frame keseluruhan ketika disalin ke memori. Fragmentasi internal merupakan ruang yang tidak dapat digunakan dalam sebuah partisi memori yang diberikan (dalam hal ini, sebuah page).

Sekarang kita membicarakan bagaimana prinsip kerjanya. Bila sebuah proses membangkitkan sebuah alamat virtual, sistem operasi harus secara dinamis mentranslasikan alamat virtual ini menjadi alamat fisik dalam memori di mana data yang sebenarnya berada. (Untuk keperluan penyederhanaan, untuk sementara kita menganggap tidak mempunyai memori cache.) Misalnya, dari sebuah *program viewpoint*, kita melihat byte terakhir dari 10-byte program sebagai alamat 9, anggap instruksi-instruksi 1-byte dan alamat-alamat 1-byte, dan alamat awal 0. Namun, ketika yang sebenarnya di-load ke memori adalah alamat logika 9 (mungkin sebuah referensi ke label X dalam sebuah bahasa rakitan) sebenarnya dapat berada pada lokasi memori fisik 1239, yang disarankan program di-load mulai pada alamat fisik 1230. Harus ada cara mudah untuk dapat mengonversikan alamat logika 9 (atau alamat virtual 9) menjadi alamat fisik 1230.

Untuk menyelesaikan translasi alamat ini, sebuah alamat virtual dibagi menjadi dua field; page field dan offset field, untuk merepresentasikan offset dalam page di mana data yang diminta berada (disimpan). Proses translasi alamat ini mirip dengan proses yang digunakan ketika kita membagi alamat-alamat memori utama menjadi field-field untuk pemetaan memori cache. Dan mirip dengan blok-blok cache, ukuran page biasanya merupakan pangkat dari 2; ini merupakan ekstraksi sederhana dari nomor page dan offset dari alamat-alamat virtual.

Untuk mengakses data yang diberikan pada alamat virtual, sistem melakukan langkah-langkah berikut:

1. Ekstraksi nomor page dari alamat virtual
2. Ekstraksi offset dari alamat virtual
3. Translasikan nomor page menjadi nomor page frame fisik dengan mengakses page table.

- A. Cari (look up) nomor page di dalam page table (gunakan nomor virtual page sebagai indeks).
- B. Cek valid bit untuk page tersebut.
 1. Jika valid bit = 0, sistem membangkitkan page fault dan sistem operasi harus melakukan intervensi ke:
 - a. Lokasi page yang diinginkan pada disk
 - b. Cari sebuah page frame yang bebas (di sini bisa menghapus page "*victim*" dari memori dan menyalin kembali ke disk jika memori penuh)
 - c. Salin page yang diinginkan ke dalam page frame bebas dalam memori utama
 - d. Perbarui *page table*. (Virtual page hanya dibawa pada nomor frame yang harus dimilikinya dan valid bit pada page table yang dimodifikasi. Jika ada page "*victim*", valid bit-nya harus di reset ke nol.
 - e. Eksekusi resume proses menyebabkan page fault, dilanjutkan ke langkah B2
 2. Jika valid bit = 1, page ada dalam memori
 - a. Ganti nomor virtual page dengan nomor frame aktual
 - b. Akses data pada offset di dalam page frame fisik dengan menambahkan offset ke nomor frame untuk virtual page yang diberikan

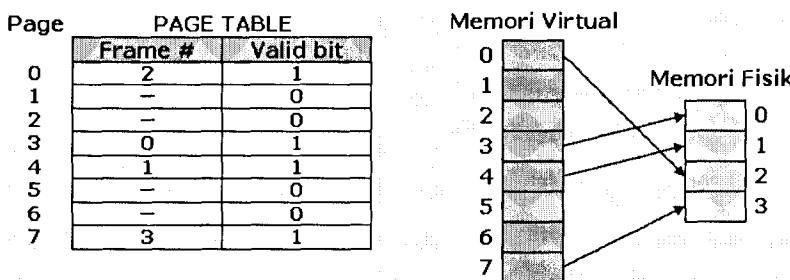
Perlu dicatat bahwa jika sebuah proses mempunyai frame-frame bebas di dalam memori utama ketika sebuah page fault terjadi, page yang diambil kembali yang terbaru ditempatkan di mana saja pada frame-frame bebas tersebut. Namun, jika memori dialokasikan untuk proses sedang penuh, maka sebuah victim page harus dipilih. Algoritma penggantian yang digunakan untuk memilih suatu victim sangat mirip dengan yang digunakan pada cache. FIFO, Random, LRU dan semua algoritma penggantian potensial lainnya untuk pemilihan sebuah victim page.

Mari kita meninjau sebuah *contoh*. Anggap kita mempunyai sebuah ruang alamat virtual 2^8 word untuk suatu proses yang diberikan (ini berarti program membangkitkan alamat-alamat dalam rentang 0 sampai 255 atau 00h sampai FFh), dan memori fisik 4 page frame (tanpa cache). Anggap juga bahwa page-page panjangnya 32 word. Alamat-alamat virtual terdiri atas 8 bit, dan alamat-alamat fisik terdiri atas 7 bit (4 frame dari 32 word masing-masing 128 word atau 2^7). Anggap juga beberapa page dari proses telah

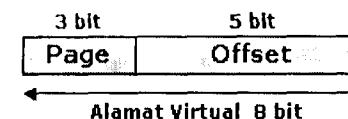
dibawa ke memori utama. Gambar 8.30 mengilustrasikan keadaan sekarang dari sistem.

Setiap alamat virtual mempunyai 8 bit dan dibagi menjadi dua field: page field mempunyai 3 bit yang mengindikasikan 2^3 page dari memori virtual ($2^8/2^5 = 2^3$). Setiap page lebarnya $2^5 = 32$ word, sehingga kita memerlukan 5 bit untuk page offset. Karena itu, sebuah alamat virtual 8 bit mempunyai format seperti yang diberikan pada Gambar 8.31.

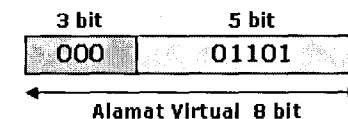
Anggap sistem sekarang membangkitkan alamat virtual $13 = 0Dh = 00001101b$. Pembagian alamat biner menjadi page field dan offset field. (Lihat Gambar 8.32), kita lihat page field $P = 000b$ dan offset field $01101b$. Untuk melanjutkan proses translasi, kita menggunakan nilai $000b$ dari page field sebagai indeks ke page table. Perhatikan entry yang ke-0 dalam page table, kita lihat bahwa virtual page 0 memetakan ke page frame fisik $2 = 10b$. Jadi alamat fisik yang ditranslasikan menjadi page frame 2, offset 13. Catatan bahwa sebuah alamat fisik hanya mempunyai 7 bit (2 bit untuk frame, karena ada 4 frame, dan 5 bit untuk offset). Penulisan dalam biner, menggunakan dua field menjadi $1001101b$ atau alamat $4Dh = 77d$ ditunjukkan pada Gambar 8.33. Kita juga dapat mencari alamat-alamat ini dengan cara lain. Setiap page mempunyai 32 word. Kita tahu alamat virtual diinginkan berada pada page 0, yang berarti dipetakan ke page frame fisik 2. Frame 2 mulai pada alamat 64. Sebuah offset dari 13 berada pada alamat 77.



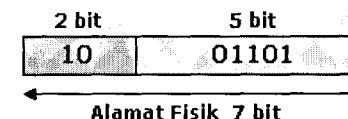
Gambar 8.30 Ilustrasi paging



Gambar 8.31 Format alamat virtual 8-bit dengan ukuran page $2^5 = 32$ word



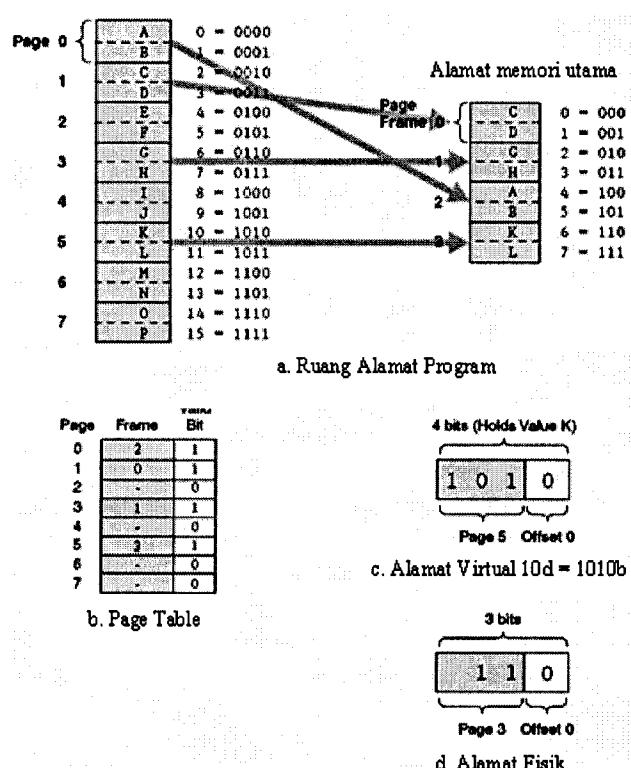
Gambar 8.32 Format alamat virtual $00001101b = 13d$



Gambar 8.33 Format alamat fisik $1001101b = 77d$

Mari kita melihat sebuah contoh yang lengkap dalam sistem nyata (tetapi kecil, dan sekali lagi tanpa cache). Anggap sebuah program panjangnya 16 byte, mempunyai akses ke sebuah memori 8 byte yang menggunakan pengalamanan byte (ini berarti setiap byte atau word mempunyai alamat sendiri), dan sebuah page panjangnya 2 word (byte). Ketika program mengeksekusi, dia membangkitkan string acuan alamat berikut (alamat diberikan dalam nilai desimal): 0, 1, 2, 3, 6, 7, 10, 11. (string acuan alamat ini mengindikasikan bahwa alamat 0 yang diacu pertamakali, kemudian alamat 1, kemudian alamat 2, dan seterusnya). Bila alamat 0 diperlukan, maka alamat 0 dan alamat 1 (dalam page 0) disalin ke page frame 2 dalam memori utama (dapat terjadi frame 0 dan 1 memori ditempati oleh proses lain sehingga tidak tersedia). Ini adalah contoh sebuah page fault, karena page dari program yang dikehendaki harus diambil/dibaca dari disk. Bila alamat 1 yang diacu, data sudah ada dalam memori (sehingga kita mempunyai sebuah page hit). Bila alamat 2 yang diacu, hal ini menyebabkan page fault lain, dan page 1 dari program disalin ke frame 0 dalam memori. Hal ini berlanjut, dan setelah alamat-alamat diacu dan page disalin dari disk ke memori utama, maka keadaan sistem ini

dapat dilihat pada Gambar 8.34a. Kita lihat bahwa alamat 0 dari program yang berisi nilai data "A", saat ini berada dalam lokasi memori $4 = 100b$. Karena itu, CPU harus ditranslasikan dari alamat virtual 0 ke alamat fisik 4, dan menggunakan skema translasi yang dibahas di atas untuk mengerjakan hal ini. Catatan bahwa alamat memori utama berisi 3 bit (ada 8 byte dalam memori), tetapi alamat-alamat virtual (dari program) harus mempunyai 4 bit (karena ada 16 byte dalam alamat virtual). Karena itu, pentranslasi juga harus mengubah alamat 4 bit menjadi alamat 3 bit.

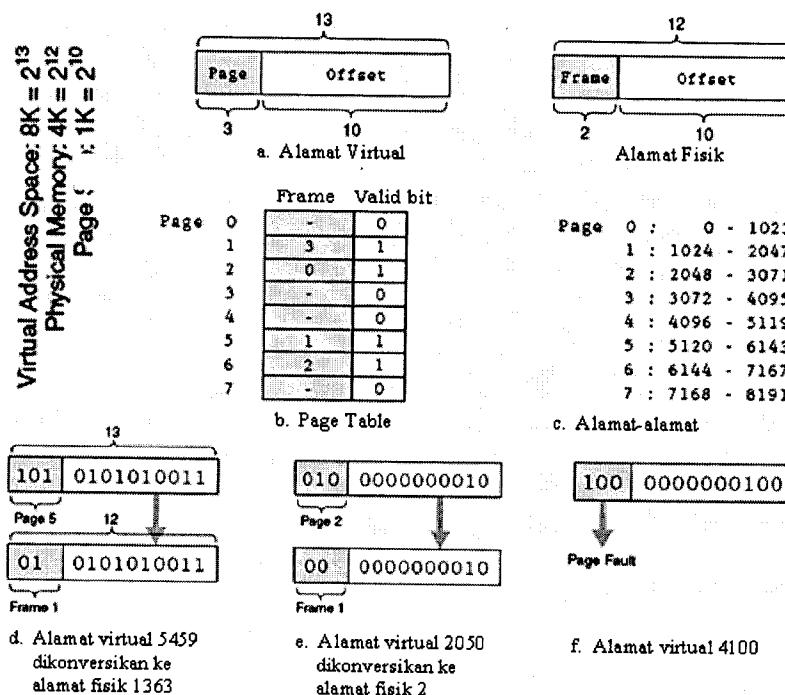


Gambar 8.34 Contoh paging dengan memori kecil

Gambar 8.34b menggambarkan page table untuk proses ini setelah page yang ada telah diakses. Kita dapat melihat bahwa page 0, 1, 3 dan 5 dari proses adalah valid, sehingga berada dalam memori. Page 2, 6, dan 7 tidak valid dan masing-masing akan menyebabkan page fault jika diajukan.

Mari kita melihat lebih dekat lagi pada proses translasi. Anggap sekarang CPU membangkitkan program, atau virtual, alamat $10 = 1010b$ untuk yang kedua. Kita lihat pada Gambar 8.34a bahwa data pada lokasi ini, "K", berada dalam alamat memori utama $6 = 0110b$. Namun demikian, komputer harus melakukan sebuah proses translasi khusus untuk menemukan data. Untuk menyelesaikan hal ini, alamat virtual, $1010b$ dibagi menjadi page field dan offset field. Page field panjangnya adalah 3 bit karena ada 8 page dalam program. Ini menyisakan 1 bit untuk offset, ini benar karena hanya ada 2 word pada setiap page. Pembagian field ini diilustrasikan Gambar 8.34c.

Ketika komputer melihat field-field ini, ada sebuah cara sederhana untuk mengubah ke alamat fisik. Nilai page field $101b$ digunakan sebagai indeks ke page table. Karena $101b = 5d$, kita gunakan 5 sebagai offset ke page table (Gambar 8.34b) dan melihat bahwa virtual page 5 memetakan ke physical frame 3. Kita sekarang mengganti $5d = 101b$ dengan $3d = 11b$, tetapi membiarkan offset-nya sama. Alamat fisik yang baru adalah $110b$, seperti yang ditunjukkan pada Gambar 8.34d. Proses ini sukses mentranslasikan dari alamat-alamat virtual ke alamat-alamat fisik dan mengurangi bit-bit dari empat ke tiga seperti yang diperlukan.



Gambar 8.35 Contoh paging dengan memori besar

Sampai saat ini kita telah bekerja dengan sebuah contoh sederhana, kita beranjak ke yang lebih besar lagi, contoh yang lebih realistik. Anggap kita mempunyai sebuah ruang alamat virtual 8K word, sebuah memori dengan ukuran 4K word yang menggunakan pengalaman byte, dan sebuah page dengan ukuran 1K word (tidak ada cache pada sistem ini, tetapi kita akan mendapatkan pengertian yang lebih dekat bagaimana memori bekerja dan pada akhirnya akan menggunakan paging dan cache dalam contoh-contoh kita), dan sebuah word dengan ukuran satu byte. Total alamat virtual 13 bit ($8K = 2^{13}$), di mana 3 bit digunakan untuk page field (ada $2^{13}/2^{10} = 2^3$ virtual page), dan 10 digunakan untuk offset (masing-masing page mempunyai 2^{10} byte). Alamat fisik hanya 12 bit ($4K = 2^{12}$), di mana 2 bit pertama sebagai page field (ada 2^2 page frame dalam memori utama) dan 10 bit sisanya sebagai offset dalam page. Format untuk alamat virtual dan alamat fisik ditunjukkan pada Gambar 8.35a.

Untuk contoh ini, mari menganggap kita mempunyai page table yang ditunjukkan pada Gambar 8.35b. Gambar 8.35c memperlihatkan sebuah tabel yang mengindikasikan berbagai alamat memori utama (dalam basis 10) yang berguna untuk pengilustrasian translasi.

Anggap CPU sekarang membangkitkan alamat virtual 5459d = 1010101010011b. Gambar 8.35d mengilustrasikan bagaimana alamat ini dibagi menjadi page field dan offset field dan bagaimana mengonversikan ke alamat fisik 1363d = 010101010011b. Pada dasarnya page field 101 dari alamat virtual digantikan oleh nomor frame 01, karena page 5 memetakan ke frame 1 (seperti yang ditunjukkan dalam page table). Gambar 8.35e mengilustrasikan bagaimana alamat 2050d ditranslasikan ke alamat fisik 2. Gambar 8.35f bagaimana alamat virtual 4100d membangkitkan sebuah page fault; page 4 = 100b tidak valid dalam page table.

Pemilihan ukuran sebuah page yang tepat merupakan hal yang sangat sulit. Ukuran page yang besar, page table kecil, agar menghemat ruang memori utama. Namun demikian, jika page terlalu besar, maka fragmentasi internal menjadi sangat buruk. Ukuran page yang lebih besar juga berarti transfer aktual sedikit dari disk ke memori utama sebagai akibat dari ukuran yang ditransfer terlalu besar. Namun, jika mereka terlalu besar, prinsip locality mulai dipecah dan kita memboroskan sumber daya dengan transfer data yang mungkin tidak diperlukan.

8.7.4 Waktu Akses Efektif Paging

Pada memori virtual dikenal juga waktu akses efektif (WAE) seperti halnya pada cache. Terdapat penalti waktu yang berkenaan dengan memori virtual. Untuk setiap akses memori yang dibangkitkan oleh prosesor, harus ada dua akses memori fisik—satu untuk mengacu ke page table dan satu untuk mengacu data aktual yang kita ingin akses. Mudah untuk melihat bagaimana pengaruh waktu akses efektif ini. Anggap sebuah akses memori utama memerlukan 200 ns dan page fault rate 1% (99% dari waktu kita mencari page yang kita butuhkan dalam memori). Anggap diperlukan 10 ms untuk mengakses sebuah page yang tidak terdapat dalam memori (waktu 10 ms ini termasuk waktu yang diperlukan untuk mentransfer page ke memori, memperbarui page table dan akses data). Waktu akses efektif untuk suatu akses memori sekarang adalah:

$$WAE = 0.99(200 \text{ ns} + 200 \text{ ns}) + 0.01(10 \text{ ms}) = 100396 \text{ ns}$$

page 100% berada di dalam memori, WAE akan menjadi:

$$WAE = 1.00(200 \text{ ns} + 200 \text{ ns}) = 400 \text{ ns}$$

yang melipatgandakan waktu akses memori. Pengaksesan page table memerlukan sebuah akses memori tambahan karena page table sendiri disimpan dalam memori utama.

Kita dapat mempercepat pencarian page table dengan menyimpan nilai pencarian page yang terakhir ke dalam sebuah cache page table yang disebut dengan translation look-aside buffer (TLB). Setiap entry TLB terdiri atas sebuah virtual page number dan frame number-nya yang sesuai. Suatu keadaan yang mungkin dari TLB untuk contoh page table sebelumnya diberikan pada Tabel 8.2.

TABEL 8.2 Keadaan TLB untuk Gambar 8.35

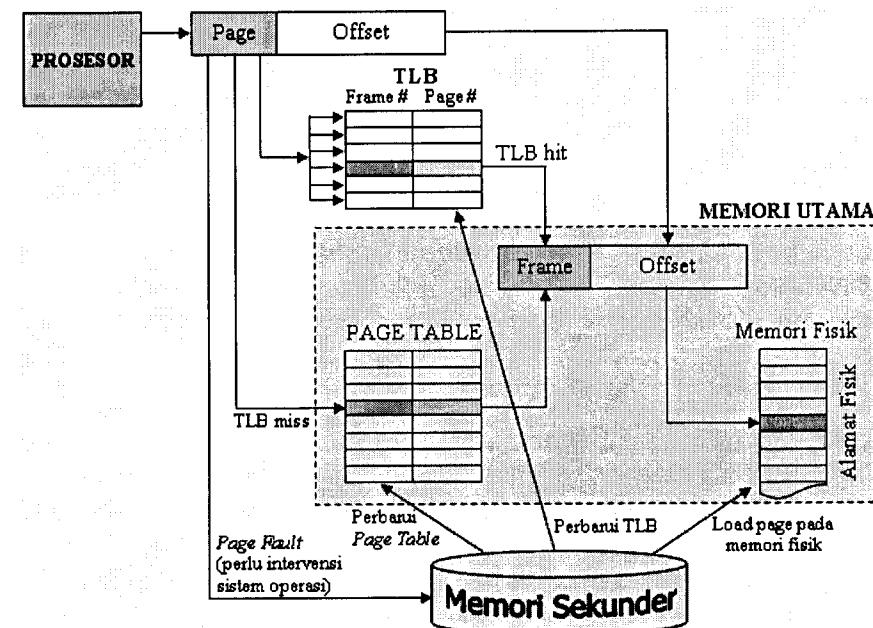
Virtual Page Number	Physical Page Number
—	—
5	1
2	0
—	—
—	—
1	3
6	2

Secara khas TLB diimplementasikan seperti cache asosiatif, dan pasangan virtual page/frame dapat dipetakan di mana saja. Ini merupakan langkah-langkah yang diperlukan untuk pencarian alamat bila menggunakan TLB (lihat Gambar 8.36):

1. Ekstraksi page number dari alamat virtual
2. Ekstraksi offset dari alamat virtual
3. Cari virtual page number dalam TLB
4. Jika pasangan (virtual page #, page frame #) ditemukan dalam TLB, maka tambahkan offset ke physical frame number dan akses lokasi memori.
5. Jika terdapat TLB miss, pergi ke page table untuk mengambil frame number yang diperlukan. Jika page ada dalam memori,

gunakan frame number yang sesuai dan tambahkan offset untuk menghasilkan alamat fisik.

6. Jika page tidak ada dalam memori utama, bangkitkan sebuah page fault dan mulai kembali (restart) akses bila page fault selesai.



Gambar 8.36 Penggunaan TLB

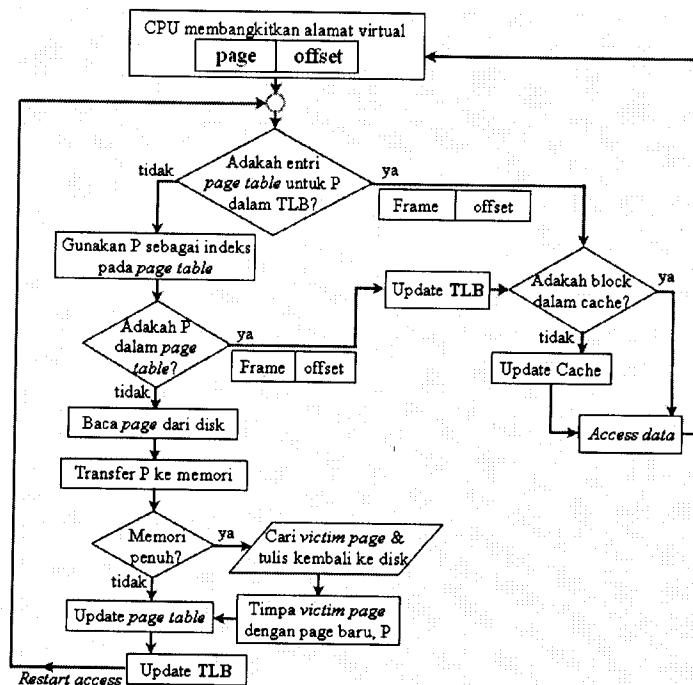
8.7.5 Penggunaan Cache, TLB, dan Paging Bersama Sekaligus

Karena TLB pada dasarnya adalah sebuah cache, penggabungan konsep ini bersama sekaligus dapat membingungkan. Menelusuri proses keseluruhan akan membantu Anda untuk memahami gambaran keseluruhan. Ketika CPU membangkitkan sebuah alamat, alamat tersebut relatif terhadap program yang dimilikinya, atau sebuah alamat virtual. Alamat virtual ini harus dikonversikan menjadi alamat fisik sebelum data retrieval dapat berlangsung. Ada dua cara menyelesaikan hal ini: (1) menggunakan

TLB untuk mencari frame dengan penempatan sebuah pasangan yang di-cache terakhir (page, frame); atau (2) pada kasus sebuah TLB miss, gunakan page table untuk mencari frame yang sesuai di dalam memori utama (secara khas tentunya TLB diperbarui pada titik). Frame number ini kemudian digabungkan dengan offset yang diberikan dalam alamat virtual untuk membuat alamat fisik.

Pada titik ini, alamat virtual telah dikonversikan menjadi alamat fisik tetapi data pada alamat belum didapat. Ada dua kemungkinan untuk menemukan data: (1) cari cache untuk melihat apakah data yang tersimpan ada; atau (2) pada suatu cache miss, pergi ke lokasi memori utama aktual untuk mendapatkan data (secara khas tentunya cache diperbarui pada titik ini).

Gambar 8.37 mengilustrasikan proses penggunaan TLB, paging, dan memori cache.



Gambar 8.37 Penggabungan bersama: TLB, page table, cache dan memori utama

8.7.6 Segmentasi

Walau merupakan metode yang paling umum dikenal, paging bukan satu-satunya cara untuk mengimplementasikan memori virtual. Metode yang kedua yang digunakan oleh beberapa sistem adalah **segmentasi**. Segmentasi merupakan pembagian ruang alamat virtual menjadi sama, page ukurannya tetap, dan ruang alamat virtual dibagi menjadi alamat logika, unit-unit variable-length, atau segmen-segmen. Memori fisik sebenarnya tidak dibagi atau dipartisi menjadi apapun. Bila sebuah segmen perlu disalin ke memori fisik, sistem operasi mencari suatu bagian memori yang bebas yang cukup besar untuk menyimpan semua segmen. Setiap segmen mempunyai sebuah alamat *base* yang mengindikasikan di mana dia ditempatkan di dalam memori, dan sebuah *bound limit* yang mengindikasikan ukurannya. Masing-masing program terdiri atas multisegmen, yang sekarang mempunyai sebuah *segment table* bersama sebagai pengganti sebuah page table. Segment table ini merupakan kumpulan sederhana dari pasangan base/bound untuk setiap segmen.

Akses memori diterjemahkan oleh penyediaan sebuah nomor segmen dan sebuah offset dalam segmen. Pengecekan error dilakukan untuk menjamin offset berada dalam bound yang diinginkan. Jika dia ada, maka nilai base untuk segmen tersebut (diperoleh dalam segment table) ditambahkan ke offset yang menghasilkan alamat fisik aktual. Karena paging berdasarkan pada sebuah blok ukuran-tetap dan segmentasi berdasarkan pada logical block, maka lebih mudah menggunakan segmentasi untuk keperluan proteksi dan sharing. Misalnya, ruang alamat virtual dapat dibagi menjadi sebuah code segment, sebuah data segment, sebuah stack segment dan sebuah symbol table segment, masing-masing dengan ukuran yang berbeda. Lebih mudah mengatakan "Saya ingin melakukan share semua data saya, sehingga membuat data segment saya dapat diakses oleh siapa saja" kemudian dia mengatakan "OK", di mana page-page data saya berada, dan sekarang bahwa saya telah menemukan empat page tersebut, mari membuat tiga page dapat diakses, tetapi hanya setengah dari page yang keempat yang dapat diakses.

Seperti halnya dengan paging, segmentasi menderita karena fragmentasi. Paging membuat fragmentasi internal karena sebuah frame dapat ditempatkan pada sebuah proses yang tidak diperlukan frame keseluruhan.

Pada sisi lain, segmentasi menderita karena fragmentasi eksternal. Bongkahan-bongkahan bebas yang berada dalam memori menjadi terpecah belah. Pada akhirnya, terdapat banyak bongkahan-bongkahan kecil, tetapi tidak cukup besar untuk menyimpan segmen keseluruhan. Perbedaan antara fragmentasi eksternal dan internal adalah bahwa dengan fragmentasi eksternal, ruang memori total yang cukup dapat tetap berada untuk alokasi sebuah proses, tetapi ruang ini tidak berurutan. Pada fragmentasi internal, memori sederhana tidak tersedia karena sistem mempunyai memori yang dialokasikan banyak untuk suatu proses yang tidak diperlukan. Untuk mengatasi fragmentasi eksternal, sistem menggunakan beberapa sort dari *garbage collection*.

8.7.7 Skema Segmentasi pada Intel 80286

Mikroprosesor 80286 mengusung implementasi memori virtual yang mudah dengan menyediakan dua fitur hardware:

1. Segmen tanpa ada pengecualian (interupsi memori virtual).
2. Instruksi-instruksi *restartable*.

Mikroprosesor 80286 mempunyai dua mode pengoperasian:

1. Mode pengalamatan riil 8086 (*real mode*)
2. Mode pengalamatan virtual terproteksi (*protected mode*)

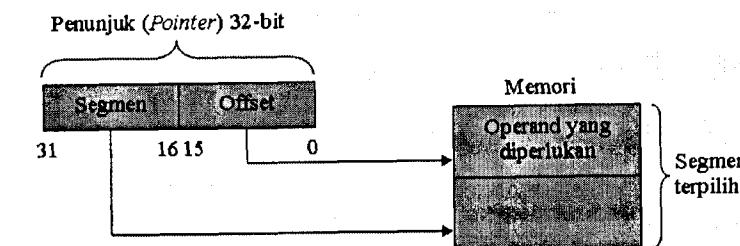
Multiprogramming, proteksi memori dan penggunaan memori virtual tidak mungkin ada pada mode riil. Pada mode pengalamatan terproteksi, mikroprosesor 80286 menawarkan multiprogramming, memori virtual dan proteksi memori. Bila 80286 reset, dia masuk ke dalam mode riil. Dia memulai eksekusi instruksi dari alamat FFFF0h. Untuk mempertahankan pada mode proteksi, sistem operasi menggunakan suatu instruksi khusus (*load machine status word*, LMSW) dengan bit *protection enable* (PE) dalam machine status word adalah '1'. Mikroprosesor 80286 menyediakan mode riil untuk memberikan kompatibilitas ke bawah dengan 8086/8088. Bila program-program yang dikembangkan untuk 8086/8088 dieksekusi pada sistem berbasis 80286, maka 80286 beroperasi dalam mode riil.

Pengalamatan Memori Mode Riil

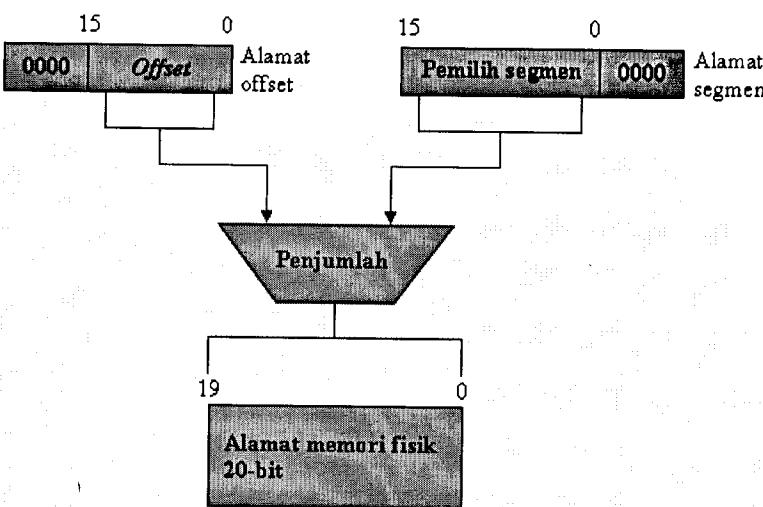
Pada mode riil, 80286 mempunyai sifat yang mirip dengan 8088 (atau 8086). Program menggunakan alamat-alamat riil. Bila program (atau instruksi) mengalami memori, terdapat dua komponen:

1. *Segment selector*
2. *Offset*

Bagian segment selector (pemilih segmen) menunjukkan segmen yang diinginkan dalam memori. Dia menyediakan alamat start segmen. Nilai offset menunjukkan alamat byte yang diinginkan dalam segmen. Gambar 8.38 menunjukkan prinsip pengalamatan memori untuk pembacaan atau penulisan memori, dan Gambar 8.39 menunjukkan mekanisme pengalamatan mode riil. Selama pengaksesan memori, 80286 mengubah segment selector 16 bit (yaitu isi dari register segmen) menjadi alamat 20 bit dengan melakukan penggeseran ke kiri sebanyak empat bit. Kemudian menambahkan offset ke dalamnya. Jadi 80286 membangkitkan alamat fisik 20-bit dan rentang memori maksimum 1 MB.



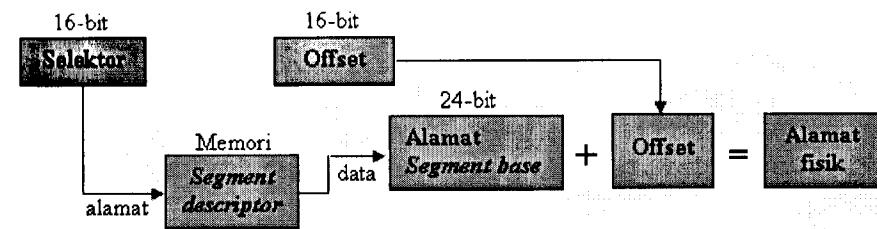
Gambar 8.38 Prinsip pengalamatan dengan 80286



Gambar 8.39 Pengalamatan memori dalam mode riil

Pengalamatan Memori Mode Proteksi (Memori Virtual)

Pada mode proteksi, pengalamatan memori sampai 16 MB karena 80286 membangkitkan alamat 24-bit. Lagi pula, dia menawarkan ruang alamat virtual 1 GB untuk setiap tugas yang dipetakan oleh 80286 menjadi ruang alamat riil 16 MB. Program yang ditulis dalam mode proteksi menggunakan alamat-alamat virtual. Mode proteksi mengisolasi program pemakai (*user program*) yang berbeda dengan menyediakan proteksi memori yang juga menjamin privasi setiap program dan data. Gambar 8.40 menunjukkan mekanisme pengalamatan memori dalam mode proteksi. Pada mode proteksi, selector tidak langsung menyediakan alamat start segmen. Bahkan dia menunjuk pada suatu lokasi memori di mana alamat start segmen disimpan. Lokasi ini adalah bagian dari *segment descriptor* bagi segmen.

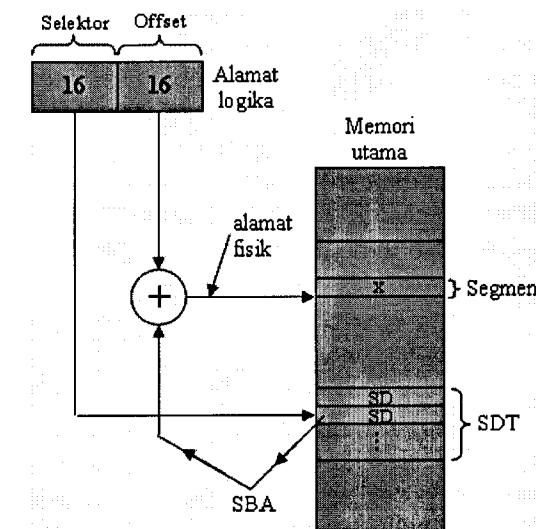


Gambar 8.40 Pengalamatan memori dalam mode proteksi pada 80286

Sekilas Mekanisme Segmentasi dan Memori Virtual 80286

Gambar 8.41 mengilustrasikan mekanisme translasi alamat 80286. Program untuk 80286 memberikan alamat logika sebagai pointer 32-bit. Ada dua bagian pada pointer:

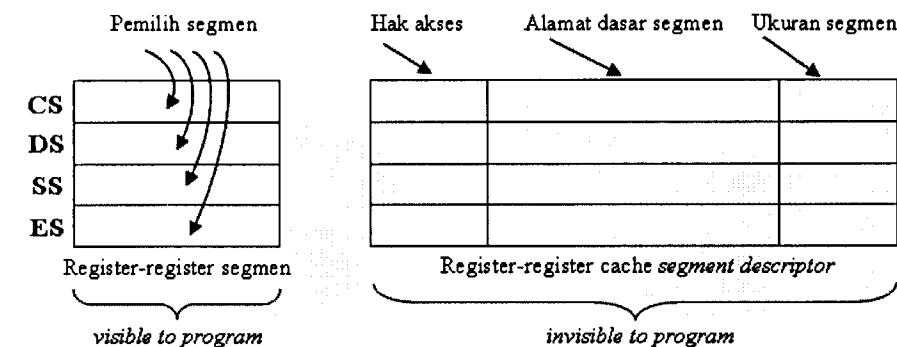
1. Selector 16-bit
2. Offset 16-bit



Gambar 8.41 Metode segmentasi translasi alamat

Selektor (pada register segmen) menyediakan indeks untuk *memory resident table* yang dikenal sebagai *Segment Descriptor Table* (SDT) yang berisi *segment descriptor* untuk semua segmen. Segment descriptor menjaga alamat dasar (start) segmen untuk semua segmen yang sedang ada dalam memori utama. Dia juga mempunyai atribut lain untuk segmen seperti ukuran segmen, hak akses (tipe proteksi) dan status keberadaan atau ketidak-beradaan dalam memori utama. Selector digunakan untuk mengakses segment descriptor (pada SDT) dari segmen yang sekarang diperlukan dan memperoleh alamat dasar segmen jika segmen yang diperlukan sedang berada dalam memori utama. Offset 16-bit pada alamat logika ditambahkan ke dalam alamat dasar segmen untuk memperoleh alamat fisik. Jika segmen yang diperlukan tidak tersedia dalam memori utama, hal ini diketahui dari segment descriptor. Karena itu bila SDT diakses, maka suatu interupsi yang dikenal dengan pengecualian 'segment not present' dibangkitkan oleh MMU. Prosesor men-switch ke sistem operasi yang me-load segmen yang diperlukan dari memori sekunder ke memori utama dan juga memperbarui segment descriptor pada SDT.

Pembaca akan kaget memperhatikan bahwa setiap pengaksesan memori utama (baca/tulis) dalam siklus instruksi melibatkan segment descriptor di mana dia sendiri ada dalam memori utama. Akses memori utama (baca) tambahan ini jelas adalah sebagai *advising* alamat dasar segmen. Hal ini menyebabkan penalti berat terhadap waktu eksekusi instruksi. Namun, suatu teknik yang digunakan untuk mengeliminasi penalti tersebut pada hampir semua pengaksesan adalah dengan menggabungkan 'cache' dalam prosesor. Mikroprosesor 80286 mempunyai suatu register '*segment descriptor cache*' untuk masing-masing dari empat jenis segmen (*code, data stack, dan extra*). Segera setelah nilai selector di-load ke dalam register segmen, 80286 secara otomatis membaca segment descriptor dari SDT dalam memori utama dan me-load ke dalam register '*segment descriptor cache*' yang sesuai sebagai suatu persiapan sebelumnya. Untuk semua pengaksesan memori berikutnya dalam segmen, prosesor membaca dari register '*segment descriptor cache*' internal dan bukan dari SDT dalam memori utama. Jadi, cache kecil dalam MMU mempercepat proses translasi alamat. Gambar 8.42. menunjukkan empat register '*segment descriptor cache*' dihubungkan ke empat segmen. Ini jelas bahwa hal ini merupakan register internal murni yang tak dapat diakses oleh program.



Gambar 8.42 Penggunaan 'descriptor cache'

8.7.8 Mekanisme Memori Virtual Intel 80386

Mikroprosesor 80386 mempunyai tiga mode operasi:

1. Mode alamat riil (mode riil)
2. Mode proteksi
3. Mode virtual 8086 (V86 mode)

Mode alamat riil: Mode ini mengemulasi prosesor 8086, dengan beberapa fitur tambahan (seperti kemampuan untuk keluar dari mode ini). Hal ini mirip dengan mode riil 80286.

Mode proteksi: Pada mode ini, seluruh instruction set dan fitur 80386 tersedia. Fitur-fitur utama adalah multitasking, proteksi memori, paging dan operasi-operasi istimewa.

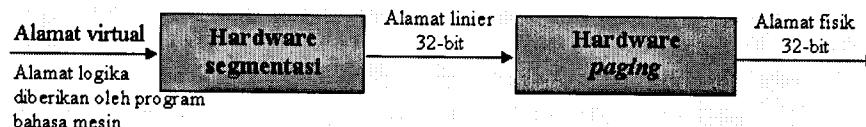
Mode virtual 8086: Ini merupakan mode khusus dalam mode proteksi. Mode ini menawarkan proteksi dan manajemen memori (tidak seperti mode alamat riil) untuk pengembangan program pada 8088. Prosesor dapat memasuki mode virtual 80886 dari mode proteksi untuk menjalankan suatu program yang ditulis untuk prosesor 8086, kemudian meninggalkan mode virtual 8086 dan masuk kembali ke mode proteksi untuk melanjutkan program yang menggunakan set instruksi 32-bit.

Ruang Alamat Mikroprosesor 80386

Ada tiga ruang alamat untuk 80386:

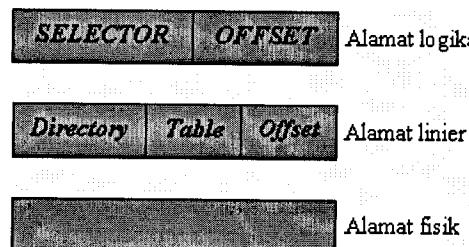
1. Ruang alamat logika (*logical address space*)
2. Ruang alamat linier (*linear address space*)
3. Ruang alamat fisik (*physical address space*)

Program bahasa mesin (kode objek) menggunakan alamat logika untuk alamat instruksi dan operand. Unit segmentasi mentranslasikan ruang alamat logika menjadi ruang alamat linier 32-bit. Bila paging unit di-enable, paging unit mentranslasikan ruang alamat linier menjadi ruang alamat fisik. Bila paging unit tidak di-enable, alamat linier dihubungkan ke alamat fisik. Gambar 8.43 mengilustrasikan translasi alamat yang digunakan oleh 80386.

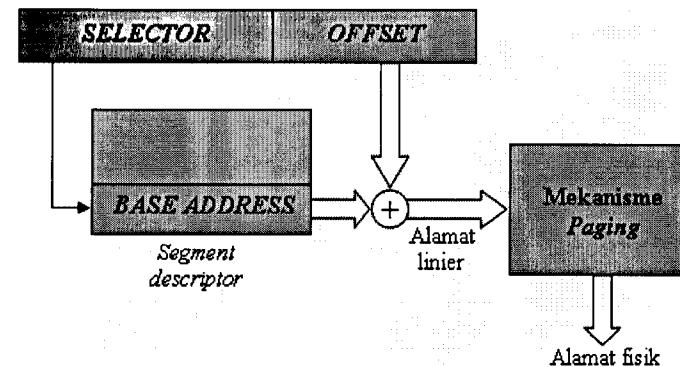


Gambar 8.43 Segmentasi dan paging-80386

Gambar 8.44 menunjukkan format alamat. Alamat logika mempunyai dua bagian: (a) selector (b) offset. Selector berada dalam register segmen. Offset dikalkulasi oleh prosesor dengan penjumlahan field -field *base*, *index*, dan *displacement* per mode pengalamatan. Unit segmentasi mentraslasikan alamat logika menjadi alamat linier. Jika paging di-enable, maka paging unit mentranslasikan alamat linier menjadi alamat fisik. Jika paging di-disable, maka alamat logika sama seperti alamat fisik. Gambar 8.45 menunjukkan translasi alamat dua level. Translasi level pertama serupa dengan mekanisme segmentasi dalam 80386.



Gambar 8.44 Format alamat untuk mikroprosesor 80386



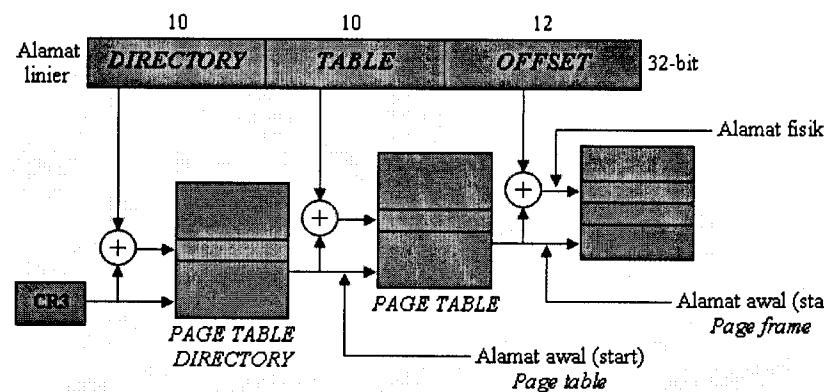
Gambar 8.45 Translasi alamat dua level pada mikroprosesor 80386/80486

Mekanisme Paging

Masukan untuk mekanisme *paging* adalah alamat linier. Semua tiga field dalam alamat linier digunakan sebagai offset oleh mekanisme *paging*. Mekanisme *paging* menggunakan dua level dari tabel untuk mentraslasikan alamat linier menjadi alamat fisik. Tabel tersebut adalah:

1. Page directory yang berisi alamat start dari page table dan informasi kontrol untuknya.
2. Page table yang berisi alamat start dari page frame dan informasi kontrol untuknya.

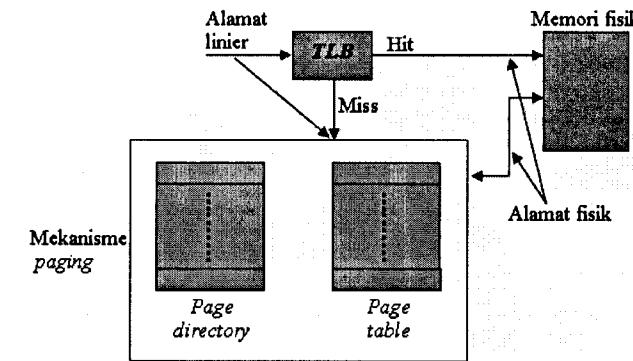
Ukuran page adalah 4KB dan semua tabel juga 4 KB. Gambar 8.46 menunjukkan mekanisme *paging*. Register CR3 berisi alamat fisik start dari page directory. Pada alamat ini, field 'directory' dalam alamat linier ditambahkan. Hasilnya menunjuk ke suatu entry dalam page directory. Karena itu dengan pengaksesan page directory, alamat awal (start) page table yang diperlukan sekarang didapatkan. Pada alamat ini, field 'table' dalam alamat linier ditambahkan. Hasilnya menunjuk ke suatu entry dalam page table. Entry ini menyediakan alamat fisik start dari page frame. Pada alamat ini, 'offset' dalam alamat linier ditambahkan. Hasilnya adalah alamat fisik 32-bit.



Gambar 8.46 Mekanisme paging pada mikroprosesor 80386/80486

Translation Lookaside Buffer

Dengan adanya dua level acuan tabel yang diperlukan untuk menentukan alamat fisik, maka diperlukan dua pengaksesan memori karena semua tabel berada dalam memori utama. Hal tersebut akan menurunkan kinerja yang besar jika diimplementasikan seperti itu. Untuk memecahkan masalah ini, mikroporsesor 80386 mempunyai sebuah dedicated cache yang dikenal dengan *translation lookaside buffer* (TLB). TLB ini adalah *entry cache memory* 32-bit. Setiap entry menyimpan sebuah page table entry. Mekanisme TLB secara otomatis menjaga 32 page table entry yang paling sering digunakan. Karena setiap page berukuran 4 KB, alamat fisik 128 KB dicakup oleh 32 entry TLB. TLB pada prosesor 80386 memberikan hit rate 95%, sedangkan TLB pada prosesor 80486 memberikan hit rate 98%. TLB adalah merupakan memori cache asosiatif set dengan 4-way. Gambar 8.47 menunjukkan penggunaan TLB. Dari alamat linier, TLB memberikan hit atau miss. Pada kasus hit, TLB mensuplai alamat page frame. Pada kasus miss, mekanisme paging diaktifkan. Jika page yang diperlukan tidak ada dalam memori utama, ‘page fault’ terjadi sebagai sebuah interupsi. Page fault dapat juga terjadi pada page directory atau page table. Sistem operasi menangani page fault dengan membawa entry yang diperlukan dari memori sekunder ke memori utama dan dia memperbarui tabel.



Gambar 8.47 Penggunaan TLB

Peranan Sistem Operasi

Sistem paging pada mikroporsesor 80386 dikenal dengan demand paging karena suatu page dikeluarkan ke memori utama bila diperlukan. Sistem operasi melakukan tugas-tugas berikut:

1. Men-set page table
2. Men-set page directory
3. Me-load register CR3
4. Membilas TLB ketika suatu perubahan dibuat untuk suatu page table entry
5. Mengimplementasikan kebijakan swapping (algoritma penggantian)
6. Menangani page fault

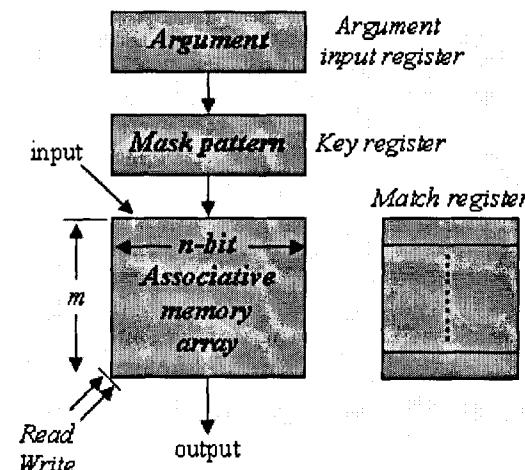
8.8 MEMORI ASOSIASIF

Memori asosiatif dikenal juga dengan *Content Addressable Memory* (CAM). Dia berbeda dengan memori lainnya seperti RAM, ROM, disk dan yang lainnya yang alamiah dalam pengaksesan dan pembacaan isi memori. Pada memori lain, alamat diberikan sebagai input untuk pembacaan suatu lokasi. Pada kasus memori asosiatif, biasanya suatu bagian dari isi atau semua isi digunakan untuk pengaksesan isi. Diberikan sebuah *argument* sebagai input, memori cache dapat mencari dan melaporkan apakah dia mempunyai *argument* ini dalam suatu lokasi. Memori asosiatif mencari semua lokasi secara simultan dalam paralel. Dia berguna sebagai memori

cache di mana dalam suatu pencarian lokasi diperlukan untuk menyesuaikannya dengan TAG yang diberikan.

Gambar 8.48 mengilustrasikan diagram blok memori asosiatif dengan m lokasi dari setiap n bit. Bila dilakukan penulisan ke dalam memori asosiatif, tidak ada alamat yang digunakan. Memori asosiatif mendapatkan lokasi kosong untuk penyimpanan data input. Untuk melakukan operasi pencarian, argument (yaitu TAG) diberikan pada register argument yang mempunyai n bit. Dua jenis pencarian yang mungkin adalah:

1. Pencarian pada semua argument
2. Pencarian pada suatu bagian (field) dalam argument



Gambar 8.48 Diagram blok associative memory

Key register mensuplai *mask pattern* yang mengindikasikan bit argument untuk dimasukkan sebagai pola pencarian. Jika ada bit 0 dalam *mask pattern*, maka bit-bit yang sesuai dari register argument tidak merespons. Pada pencarian semua argument, *key register* mensuplai 1 semua. Pada saat pembacaan (pencarian), memori asosiatif menempatkan semua word yang cocok dengan argument yang diberikan dan menandainya dengan men-set bit-bit yang sesuai dalam *match register*. Selanjutnya, pembacaan dari semua lokasi ini dilakukan dengan akses sekuensial ke word yang sesuai (match).

Memori asosiatif mahal karena memerlukan matching logic pada setiap sel. Dari sudut pandang kemampuan pencarian kecepatan tinggi yang dimilikinya, dia merupakan pilihan terbaik untuk memori cache dan TLB dalam unit translasi alamat.

RINGKASAN

Dua parameter memori utama yang sangat penting pada komputer: kecepatan dan kapasitas. Kecepatan memori utama lebih lambat daripada prosesor. Kapasitas fisik memori utama terbatas karena faktor harga. Teknik memori virtual membantu menjalankan program yang panjang dengan ruang memori utama yang terbatas sedangkan fluktuasi ketidaksesuaian kecepatan diatasi dengan mengikuti teknik-teknik seperti prefetch instruksi, interleave memori, buffer tulis dan memori cache.

Tujuan prefetch instruksi adalah untuk menerima instruksi berikutnya dari memori utama, lebih awal, ketika instruksi yang ada sekarang sedang dieksekusi. Hal ini dicapai dengan suatu set dari 'prefetch buffer' di mana instruksi yang diambil sebelumnya (lebih awal) disimpan. Prefetch instruksi adalah sebuah fitur hardware dan program tidak mengetahuinya.

Memory interleaving adalah suatu teknik reorganisasi memori utama menjadi modul-modul independen agar semua bandwidth ditingkatkan berlipat ganda. Interleave memori n-way menghadirkan bandwidth n-kali bandwidth yang non-interleaving.

Secara fungsional, buffer tulis adalah kebalikan dari prefetch instruksi. Buffer tulis menyebabkan CPU menjadi terbebas dalam melakukan penulisan memori. CPU mengeluarkan informasi mengenai siklus tulis memori ke buffer tulis dan mendahului siklus instruksi berikutnya. Buffer tulis menunggu hingga memori bebas dan melakukan operasi penulisan memori dengan menerima informasi dari buffer tulis.

Memori cache adalah suatu buffer tengah antara CPU dan memori utama. Tujuannya adalah untuk mengurangi waktu tunggu CPU selama pengaksesan memori utama. Sebagian program dan data sebelumnya dibawa ke dalam memori cache. Pengontrol cache menjaga lokasi memori utama yang akan disalin pada memori cache pada waktu yang diberikan. Bila CPU memerlukan suatu instruksi atau operand, dia menerimanya dari memori cache (jika tersedia) sebagai pengganti dari pengaksesan memori utama. Jadi, akses memori selesai dalam waktu yang singkat sehingga memori utama tampak seperti memori yang cepat. Namun, jika item yang diperlukan tidak ada di dalam memori cache, maka dilakukan pengaksesan ke memori utama. Kapasitas memori cache sangat kecil karena mahal (dibandingkan dengan memori utama).

Item yang ditemukan pada memori cache disebut cache hit. Jika terjadi sebaliknya disebut cache miss di mana jika terjadi cache miss maka prosesor harus mengaksesnya ke memori utama. Cache hit yang lebih sering terjadi, lebih baik karena setiap terjadi 'miss' maka harus dilakukan pengaksesan memori utama. Waktu yang digunakan untuk membawa item yang diperlukan dari memori utama dan menyulainya ke prosesor disebut sebagai 'miss penalty'. Jika memori cache penuh, maka beberapa item dihapus untuk pembuatan ruang bagi entry baru. Pengeluaran suatu bagian dari memori cache adalah berdasarkan pada suatu algoritma khusus dan dapat memperbesar hit rate. Hit rate (disebut juga hit ratio) menyediakan bilangan jumlah akses yang dihadapi 'cache hit' terhadap total jumlah akses. Memori cache ada dua jenis: cache-gabung (unified cache) dan cache-pisah (split cache). Beberapa komputer menggunakan dua level atau tiga level sistem memori cache.

Ada tiga metode pemetaan memori cache yang populer. Direct Mapping, Fully Associative Mapping dan Set Associative Mapping. Jika sebuah blok memori utama akan disimpan, baris cache di mana blok tersebut harus ditulisi ditentukan oleh fungsi pemetaan. Dengan cara yang sama, jika CPU membaca dari lokasi memori utama, pengontrol cache menggunakan fungsi pemetaan untuk mengidentifikasi baris cache di mana blok memori utama disimpan. Selama operasi tulis, ada dua strategi yang diperbolehkan yaitu write-through policy dan write-back policy.

Jika memori cache sudah penuh, maka beberapa isi memori cache harus dihapus untuk membuat ruang bagi entry baru. Masalah penggantian adalah suatu hal yang penting. Hal ini dilakukan bila memori cache penuh dan suatu blok baru dari memori utama harus disimpan dalam memori cache. Algoritma penggantian tertentu telah dicoba dan dipelajari efisiensinya. Beberapa di antaranya adalah *algoritma random choice*: Algoritma ini memilih baris cache secara acak tanpa suatu acuan pada frekuensi penggunaan sebelumnya. Algoritma *First-In First-Out (FIFO)*: Algoritma ini memilih set yang telah berada pada cache dalam waktu yang lama. Dengan kata lain, blok yang dimasukkan dalam cache pertama didorong keluar pertama. Asumsinya adalah bahwa item yang baru saja dimasukkan mungkin yang akan diperlukan. *Algoritma Least Frequently Used (LFU)*: Algoritma ini memilih suatu blok yang telah digunakan oleh CPU yang paling sedikit. Asumsinya adalah bahwa cara serupa juga mungkin akan terjadi kemudian. *Algoritma Least Recently Used (LRU)*: Algoritma ini

memilih item yang terlama dalam cache tidak (jarang) digunakan. Asumsinya adalah bahwa cara serupa akan berlanjut dan tidak mungkin dibutuhkan segera.

Ada kemungkinan lokasi memori utama diperbarui tanpa pengetahuan prosesor (dan memori cache). Contoh khas adalah operasi input dari hard disk ketika pengontrol DMA menyimpan data dalam memori utama. Hal ini disebabkan karena mismatch antara memori utama dan memori cache. Untuk mencegah hal ini maka suatu logika pemantau bus ditugaskan untuk memantau aktivitas memori (bus) oleh pengontrol DMA (atau prosesor lainnya jika ada). Hal ini dikenal sebagai *snooping* (pengintaian). Pada pendekslan permulaan dari suatu urutan penulisan memori untuk suatu alamat yang telah dipetakan dalam memori cache, maka dengan segera pengontrol cache bersiaga menanyakan untuk membatalkan data dalam memori cache. Untuk maksud ini, alamat memori dikirim ke pengontrol cache. Pengontrol cache men-set *INVALID flag* untuk entry ini.

Pada sistem memori virtual, sistem operasi secara otomatis mengatur program-program yang panjang. Sistem operasi menyimpan semua program pada hard disk yang mempunyai kapasitas yang lebih besar daripada ukuran memori fisik. Pada satu waktu, hanya beberapa bagian program yang dibawa ke dalam memori utama dari hard disk. Pada saat diperlukan, bagian yang tidak berada dalam memori utama dikirim—dari hard disk, dan pada saat yang sama, bagian dari suatu program yang berada dalam memori utama dikeluarkan dari memori utama dan disimpan pada hard disk. Proses ini dikenal sebagai “*swapping*” (pertukaran).

Ada dua metode yang populer dalam implementasi memori virtual, yaitu paging dan segmentation. Pada metode paging, software sistem membagi program menjadi halaman-halaman. Ini tidak dikenal pada programmer. Pada metode segmentasi, pemrogram (bahasa mesin) menyusun program ke dalam segmen-segmen berbeda yang tidak mempunyai ukuran yang sama.

Memori asosiatif (*associative memory*) dikenal juga dengan *Content Addressable Memory* (CAM). Pada kasus memori asosiatif ini, umumnya suatu bagian dari isi atau semua isi digunakan untuk pengaksesan isi. Memori asosiatif mencari semua lokasi secara simultan dalam paralel. Memori asosiatif mahal karena memerlukan matching logic pada setiap sel, karena itu kecepatannya tinggi.

SOAL-SOAL ULANGAN

1. Ada beberapa teknologi memori yang berbeda yaitu (1) pita magnetik, (2) disk optik, (3) hard disk magnetik, (4) memori _____.
2. Hierarki memori berdasarkan waktu akses dari yang lambat ke yang cepat yaitu: (1) memori sekunder, (2) memori utama/primer, (3) memori cache, dan (4) _____.
3. Teknologi disk optik menawarkan _____ yang lebih baik dibandingkan dengan hard disk magnetik tetapi lebih lambat.
4. Memori cache dapat ditempatkan lebih dari satu level. Memori cache yang terdekat dengan CPU disebut cache level _____.
5. Cache _____ lebih cepat daripada cache *off-chip* dengan teknologi yang sama.
6. Ada dua isu yang berhubungan dengan fungsi memori utama dalam komputer yaitu: (1) kecepatan dan (2) _____.
7. Dua faktor penting yang berkontribusi terhadap waktu siklus instruksi yaitu: (1) kecepatan clock prosesor, dan (2) _____.
8. Empat teknik dalam mengatasi masalah kelambatan memori utama yaitu: (1) *prefetch* instruksi, (2) *interleave* memori, (3) *buffer tulis*, dan (4) _____.
9. _____ merupakan solusi yang tepat untuk mengeksekusi program yang panjang dengan memori utama yang terbatas.
10. _____ adalah teknik untuk mengakses instruksi berikutnya dari memori utama, ketika *current instruction* dieksekusi oleh prosesor.
11. _____ adalah teknik reorganisasi (pembagian) memori utama menjadi modul-modul independen yang jamak untuk peningkatan bandwidth.
12. Dalam *interleaving* 2-way, dua pengaksesan selesai dalam 50 ns. Karena itu bandwidth memori _____ MB/sec.
13. Secara umum, *interleave* memori *n*-way memberikan bandwidth sebesar _____ bandwidth kasus *non-interleaving*.
14. _____ adalah teknik yang digunakan untuk membebaskan CPU dari tugas operasi penulisan memori.
15. Penyangga (buffer) yang digunakan antara CPU dan memori utama untuk mengatasi kelambatan memori utama adalah _____.
16. Operasi cache berdasarkan pada “*locality of reference*” yang merupakan sifat yang melekat dalam program. Hal ini terbagi dua yaitu (1) *temporal locality*, dan (2) _____.

17. Instruksi sekarang yang diambil dari cache dapat diperlukan kembali dengan segera, disebut _____.
18. Instruksi-instruksi dalam cache yang berdekatan dengan *current instruction* dapat diperlukan segera, disebut _____.
19. Ada tiga metode popular dalam teknik pemetaan memori cache yaitu: (1) pemetaan langsung (*direct mapping*), (2) pemetaan asosiatif penuh (*fully associative mapping*), dan (3) _____.
20. Jika alamat memori yang dicari CPU ditemukan (terpetakan) dalam cache, disebut _____.
21. Jika alamat memori yang dicari CPU tidak ditemukan (terpetakan) dalam cache, disebut _____.
22. Bilangan yang menunjukkan jumlah akses yang ditemukan di dalam memori cache terhadap jumlah akses keseluruhan, disebut _____.
23. Blok memori utama yang hanya dapat dipetakan pada satu baris khusus (spesifik) di dalam cache, disebut pemetaan _____.
24. Blok memori utama dapat dipetakan pada sembarang baris di dalam cache, disebut pemetaan _____.
25. Jika memori cache sudah penuh, maka beberapa isi memori cache diganti/dihapus untuk membuat ruang bagi entry baru, disebut _____.
26. Waktu yang digunakan untuk membawa item yang diperlukan dari memori utama dan menyuplainya ke CPU karena tidak tersedia dalam memori cache, disebut _____.
27. Algoritma penggantian cache yang memilih suatu blok yang paling jarang digunakan oleh CPU, disebut _____.
28. Algoritma penggantian cache yang memilih suatu blok yang paling lama berada di dalam cache, disebut _____.
29. Kebijakan penulisan cache ada dua cara yaitu: (1) *write-through policy*, dan (2) _____.
30. Kebijakan penulisan cache di mana penulisan hasil dilakukan dalam memori utama dan memori cache, disebut _____.
31. Memori cache ada dua jenis, yaitu: (1) *unified cache*, dan (2) _____.
32. *Unified cache* adalah memori yang menggabungkan antara (1) _____, dan (2) _____.
33. Pada memori virtual dikenal (1) alamat logika, dan (2) alamat _____.
34. Alamat logika mempunyai dua bagian. Bagian pertama menentukan nomor modul program (nomor segmen atau nomor page). Bagian kedua menyediakan _____ atau nomor word pada modul program tersebut yang menghubungkan ke awal modul tersebut.

35. Ada dua metode yang populer dalam implementasi memori virtual, yaitu: (1) paging, dan (2) _____.

SOAL-SOAL LATIHAN

1. Sebuah memori mempunyai waktu akses 20 ns. Hitung perolehan kinerja karena prefetch instruksi (instruction prefetch), anggap bahwa 10% dari instruksi adalah instruksi branch.
2. Sebuah CPU mempunyai ruang memori 1 MB. Rancang memori interleaving 4-way dengan waktu siklus IC memori 20 ns. Hitung bandwidth efektif.
3. Sebuah memori mempunyai waktu akses 50 ns. Hitung perolehan kinerja karena buffer tulis (write buffer) jika siklus memori untuk baca dan tulis berada dalam ratio 3:1
4. Sebuah memori cache kecepatannya 10 kali lebih cepat dari memori utama. Hit ratio adalah 0.9.
 - a) hitung peningkatan kinerja pada fetch instruksi karena memori cache. Anggap MISS terjadi pada perpindahan informasi pertama dari memori utama ke cache dan kemudian dari cache ke CPU.
 - b) Telah diamati bahwa dengan dua kali ukuran memori cache kejadian MISS setengah. Berapa peningkatan kinerjanya?
5. Sebuah memori utama komputer terdiri atas 1024 blok yang masing-masing 256 word. Memori cache mempunyai delapan set masing-masing empat blok. Tentukan field TAG, SET dan WORD pada alamat memori.
6. Sebuah komputer mempunyai memori utama 64 MB dan memori cache 32 KB. Setiap baris dalam memori cache menyimpan 8 byte.
 - a) Tunjukkan format alamat memori untuk pemetaan langsung, pemetaan asosiatif penuh, pemetaan asosiatif set 8-way.
 - b) Berapa jumlah baris yang terdapat dalam cache.
7. Sebuah memori cache asosiatif set 2-way berisi empat set. Memori utama mempunyai 2K blok yang masing-masing terdiri atas delapan word.
 - a) Tunjukkan format alamat memori utama yang mengizinkan kita untuk memetakan alamat dari memori utama ke memori cache. Berikan field dan ukurannya masing-masing.

- b) Hitung hit rate (hit ratio) untuk suatu program yang melakukan loop 3 kali dari lokasi 8 sampai 51 di dalam memori utama.
8. Anggap bahwa Anda mempunyai mikroprosesor dengan rentang pengalamanan 1 megabyte dan memori cache dengan saluran alamat 12 bit. Kapasitas setiap blok adalah 64 byte. Prosesor menganut pemetaan langsung.
- Berapa jumlah baris dalam memori cache?
 - Berapa bit yang dibutuhkan untuk tag?
 - Anggap prosesor melakukan operasi fetch instruksi dari alamat memori utama 3FB0Ah. Pada baris berapakah item tersebut dipetakan di dalam memori cache?
9. Ulangi pertanyaan no.1 di atas untuk pemetaan asosiatif penuh (*fully associative mapping*).
10. Anggap bahwa anda mempunyai mikroprosesor dengan rentang pengalamanan 1 megabyte dan 4 kilobyte memori cache pemetaan asosiatif set. Kapasitas setiap blok adalah 64 byte. Kapasitas set adalah 16 blok.
- Berapa jumlah baris dalam memori cache?
 - Berapa jumlah set dalam memori cache?
 - Berapa bit yang dibutuhkan untuk tag?
 - Anggap prosesor melakukan operasi fetch instruksi dari alamat memori utama 3FB0Ah. Pada set berapakah item tersebut dipetakan di dalam memori cache?
11. Sebuah komputer mempunyai memori utama 16MB. Memori cache yang terpasang mempunyai saluran alamat 13 bit. Terdapat 1048576_{10} jumlah blok dalam memori utama dan teknik yang digunakan adalah set associative mapping 4-way.
- Tentukan panjang medan (field) word, medan set dan medan tag
 - Berapa jumlah baris di dalam cache?
 - Berapa jumlah set di dalam cache?
 - Berapa byte isi setiap baris cache?
 - Anggap prosesor melakukan fetch instruksi pada memori utama dengan nomor blok 125_{10} . Pada set berapakah word yang dibaca tersebut harus dipetakan di dalam memori cache?
12. Sebuah komputer mempunyai alamat logika 24 bit dan alamat memori fisik 12 bit. Jika ukuran page 2K, hitung jumlah page dan jumlah blok memori utama.

13. Sebuah ruang alamat logika komputer mempunyai 64 segmen. Ukuran segmen 64K word. Memori fisik mempunyai page 1K dengan masing-masing 4K word. Tentukan format alamat logika dan format alamat fisik.
14. Anggap sebuah page table proses berisi seperti pada tabel berikut. Gunakan format yang ditunjukkan pada Gambar 6.15a, tunjukkan di mana page proses ditempatkan di dalam memori.

Frame	Valid Bit
1	1
-	0
0	1
3	1
-	0
-	0
2	1
-	0

15. Anggap sebuah page table proses berisi seperti pada tabel berikut. Gunakan format yang ditunjukkan pada Gambar 6.15a, tunjukkan di mana page proses ditempatkan di dalam memori.

Frame	Valid Bit
-	0
3	1
-	0
-	0
2	1
0	1
-	0
1	1

16. Sebuah sistem mengimplementasikan sebuah ruang memori virtual yang di-page untuk setiap proses menggunakan one-level page table. Ukuran maksimum ruang alamat virtual adalah 16 MB. Page table untuk proses running melibatkan valid entry berikut (tanda → menunjukkan bahwa sebuah virtual page dipetakan pada page frame yang diberikan, yaitu ditempatkan pada frame tersebut):

Virtual page 2 → Page frame 4

Virtual page 4 → Page frame 9

Virtual page 1 → Page frame 2

Virtual page 3 → Page frame 16

Virtual page 0 → Page frame 1

Ukuran page 1024 byte dan ukuran memori fisik maksimum mesin adalah 2 MB.

- a) Berapa bit yang diperlukan untuk setiap alamat virtual?
- b) Berapa bit yang diperlukan untuk setiap alamat fisik?
- c) Berapa jumlah maksimum entry dalam sebuah page table?
- d) Pada alamat fisik manakah akan ditranslasikan alamat virtual 1524_{10} ?
- e) Alamat virtual manakah yang akan ditranslasikan ke alamat fisik 1024_{10} ?

BAB 9

ORGANISASI INPUT/OUTPUT

Sasaran bab ini:

1. Pengontrol I/O

2. Jenis-Jenis Bus

3. Antarmuka Sistem

4. Antarmuka Perangkat

5. Antarmuka Paralel

6. Antarmuka Serial

7. Teknik-Teknik I/O

8. Port I/O

9. Programmed I/O

10. Interupsi

11. DMA

12. Prosesor I/O

13. Transfer Sinkron dan Asinkron

14. Arbitrasi Bus

Prosesor dan memori bekerja sama dalam mengeksekusi program dalam komputer. Mereka berkomunikasi dengan unit input dan output untuk keperluan pemakai dan sistem. Secara terpisah, kinerja sistem komputer bergantung pada metode-metode yang digunakan untuk komunikasi antara unit-unit fungsional. Mode transfer data yang digunakan pada berbagai perangkat peripheral dalam sistem komputer merupakan pertimbangan pemilihan kinerja dan biaya sistem, di samping perangkat.

Ada beberapa pertimbangan sehingga peripheral tidak dihubungkan secara langsung dengan sistem bus antara lain:

- Karena beberapa peripheral menggunakan metode operasi yang beragam, maka tidak akan cocok menggabungkan logika prosesor untuk mengontrol berbagai perangkat peripheral.
- Transfer data peripheral lebih lambat dibandingkan prosesor atau memori, tidak cocok jika menggunakan bus sistem yang berkecepatan tinggi untuk berkomunikasi langsung dengan peripheral.
- Pada sisi lain pemindahan data oleh peripheral lebih cepat dibandingkan dengan memori atau prosesor. Ketidakcocokan ini membuat kinerja yang kurang efisien.
- Peripheral biasanya menggunakan format data yang berbeda dibanding komputer tempat peripheral terpasang.
- Pengontrol I/O mempunyai dua fungsi utama:
 - Penghubung antara prosesor dan memori melalui sistem bus atau sentral switch.
 - Penghubung antara satu atau lebih peripheral melalui data link.

Pada bagian ini difokuskan pada komunikasi antar sistem komputer dan teknik-teknik I/O.

9.1 PENGONTROL I/O dan I/O DRIVER

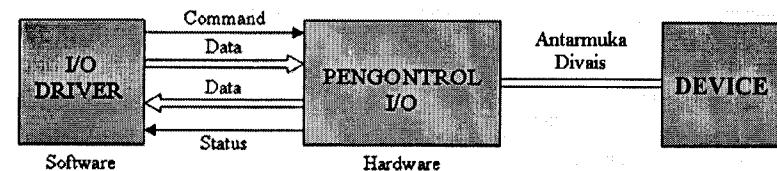
Setiap komputer didukung oleh sejumlah perangkat peripheral. Untuk menggunakan perangkat peripheral, dibutuhkan dua modul yaitu:

1. Modul hardware yang disebut "Pengontrol I/O" yang melakukan antarmuka perangkat peripheral ke inti sistem (CPU/memori, lihat Gambar 9.1).



Gambar 9.1 Hubungan pengontrol I/O dan perangkat

2. Modul software disebut "I/O driver" yang menyampaikan berbagai perintah ke pengontrol I/O untuk melakukan sejumlah operasi I/O (Gambar 9.2).



Gambar 9.2 Hubungan I/O driver dan perangkat

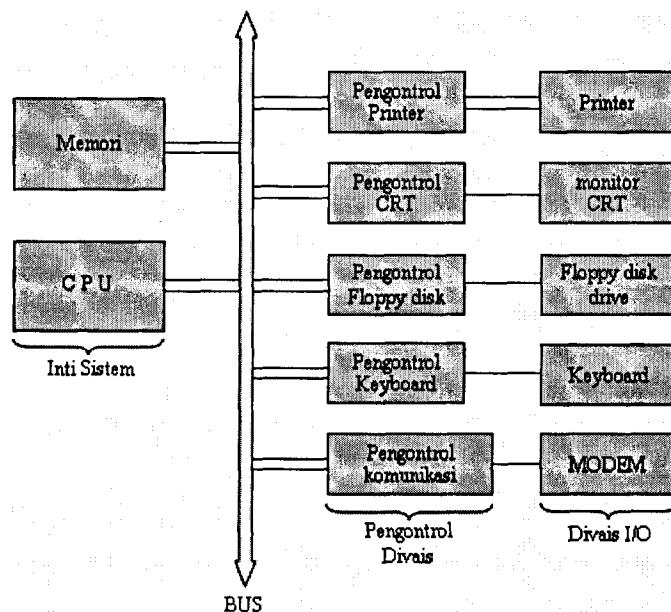
9.1.1 Pengontrol Perangkat

Antarmuka antara perangkat peripheral ke inti sistem dapat dilakukan dengan menggunakan pengontrol perangkat. Hubungan pengontrol perangkat ke CPU atau memori adalah melalui bus. Pada Gambar 9.3 diberikan diagram blok yang menunjukkan hubungan berbagai unit-unit fungsional hardware ke bus. Pengontrol perangkat diantarangkanakan ke bus sistem pada satu sisi dan perangkat peripheral di sisi lain. Kedua sisi ini berturut-turut disebut antarmuka sistem dan antarmuka perangkat seperti yang ditunjukkan pada Gambar 9.4.

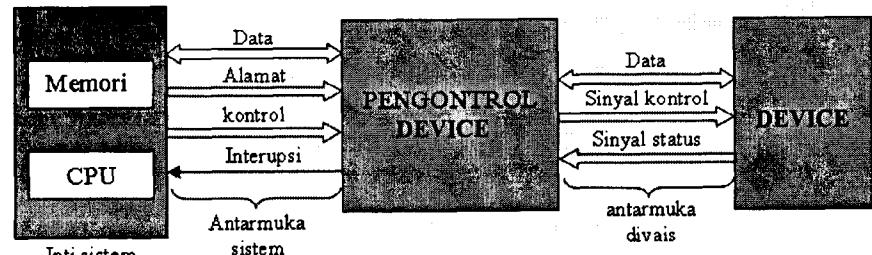
Istilah "antarmuka" diartikan sebagai sinyal-sinyal yang terdapat antara dua subsistem dan protokol komunikasi antara subsistem. Antarmuka sistem mengandung sinyal-sinyal antara inti sistem dan pengontrol I/O. Sinyal-sinyal ini ditentukan oleh CPU. Antarmuka perangkat mengandung sinyal-sinyal antara pengontrol I/O dan perangkat I/O. Antarmuka perangkat tergantung jenis perangkat. Contoh antarmuka perangkat antara lain: RS-232C, antarmuka centronics, SA450, ST-506, SCSI, IDE, USB, Firewire.

Ada dua jenis antarmuka perangkat: antarmuka serial dan antarmuka paralel. Pada antarmuka serial (Gambar 9.5), hanya ada satu saluran data dan bit-bit data dalam suatu byte ditransmisikan bit demi bit (satu per satu secara serial). Contoh RS-232C. Antarmuka paralel (Gambar 9.6) menyediakan delapan saluran data secara paralel supaya delapan bit data (byte) ditransmisikan dari sistem secara serempak/simultan, contoh antarmuka ini adalah centronics.

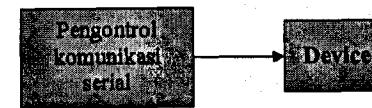
Perancang komputer menggunakan berbagai macam teknik dalam merancang pengontrol I/O. Ada pengontrol yang bekerja lebih seperti seorang pembawa pesan antara software dan perangkat peripheral. Istilah yang tepat untuk pengontrol yang demikian adalah adaptor antarmuka. Pengontrol printer biasanya termasuk dalam jenis ini. Pengontrol lainnya adalah pengontrol pintar (*intelligent*) yang merupakan sebuah pengontrol yang canggih dengan keistimewaan (fitur) lebih, seperti *self-diagnostic*, *error retry*, *error correction*, dan sebagainya. Pengontrol yang demikian dapat mengerjakan perintah yang kompleks. Contoh pengontrol pintar adalah pengontrol hard disk. Gambar 9.7 menampilkan klasifikasi pengontrol perangkat.



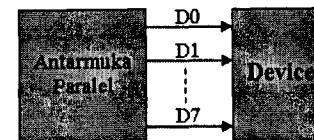
Gambar 9.3 Unit-unit fungsional dan bus



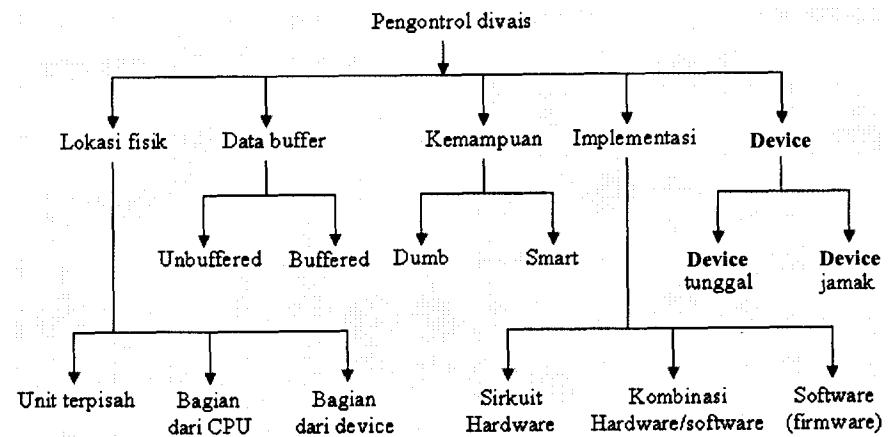
Gambar 9.4 Hubungan komunikasi pengontrol



Gambar 9.5 Antarmuka serial



Gambar 9.6 Antarmuka parallel



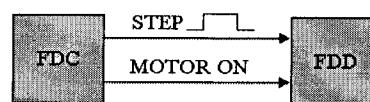
Gambar 9.7 Jenis-jenis pengontrol perangkat

Sinyal-Sinyal Kontrol

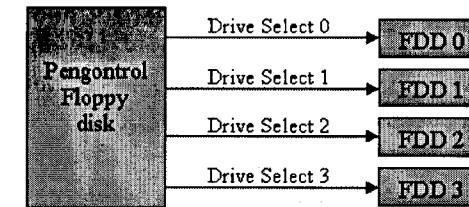
Setiap sinyal kontrol menentukan informasi/tindakan kontrol yang khusus pada perangkat. Pada dasarnya ada tiga jenis sinyal kontrol:

1. Aksi perangkat internal
2. Transfer data
3. Pemilihan antarmuka

Pengontrol membangkitkan sinyal kontrol yang tepat bergantung pada perintah yang diterima dari CPU/device driver. Misalnya, untuk mengeksekusi perintah pencarian, pengontrol floppy disk memberikan sinyal STEP ke floppy disk drive (Gambar 9.8). Saat menerima pulsa STEP, disk drive menggerakkan head R/W pada satu track. Bila pengontrol membangkitkan sinyal MOTOR-ON, disk drive memutar spindle motor. Untuk memilih disk drive, pengontrol mengirimkan sinyal "drive select" sebagai hasil hubungan perangkat secara logika ke pengontrol. Perangkat hanya mengenali sinyal yang lain jika telah terhubung secara logika ke pengontrol. Beberapa perangkat secara fisik dapat diantarmukakan ke pengontrol, tetapi hanya ada satu yang terhubung secara logika pada satu waktu seperti yang ditunjukkan pada Gambar 9.9. Sinyal kontrol untuk transfer data menunjukkan pada perangkat bahwa sekarang pengontrol sedang mensuplai data dalam kondisi operasi output atau mengharapkan/menunggu data dalam kondisi operasi input. Misalnya, sinyal "strobe" pada pengontrol printer melaporkan pada printer bahwa pengontrol telah menempatkan byte data pada saluran data. Printer segera menerima data. Tabel 9.1 menunjukkan aksi yang dilakukan oleh floppy disk drive pada sejumlah sinyal kontrol pada antarmuka floppy.



Gambar 9.8 Sinyal kontrol



Gambar 9.9 Koneksi logika pengontrol floppy disk dengan floppy disk driver

TABEL 9.1 Sinyal-sinyal kontrol pada antarmuka floppy

No.	Sinyal kontrol	Aksi perangkat
1	<i>Head select</i>	FDD memilih head atas (0) atau head bawah (1) sesuai keadaan (0 atau 1) pada sinyal ini.
2	<i>Step</i>	FDD menggerakkan head ke track yang berdekatan dalam arah yang ditunjukkan oleh sinyal kontrol arah
3	<i>Direction</i>	Sinyal ini memutuskan apakah FDD harus menggerakkan head ke dalam (menuju pusat) atau ke luar (menjauhi pusat) untuk setiap pulsa step
4	<i>Write enable</i>	FDD mensuplai arus penulisan ke head baca/tulis
5	<i>Motor enable (motor on)</i>	FDD memulai memutar spindle motor
6	<i>Drive select</i>	FDD mendapat hubungan secara logik

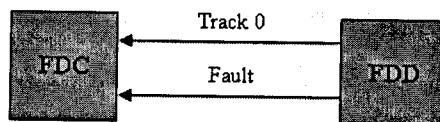
Sinyal-Sinyal Status

Setiap sinyal status mengindikasikan suatu kondisi spesifik perangkat. Ada tiga jenis sinyal status:

1. Pemilihan antarmuka
2. Transfer data
3. Status/kondisi internal

Misalnya, sinyal "Track 0" diberikan oleh disk drive (Gambar 9.10) bila head baca/tulis berada pada track 0 (track terluar). Sinyal FAULT dibangkitkan oleh floppy disk drive bila ada kondisi error pada drive. "SLDC"

dari printer mengindikasikan bahwa telah terjadi hubungan secara logic pada pengontrol. Sinyal ACK memberitahukan pengontrol bahwa printer telah menerima byte data yang ditransmisikan oleh pengontrol. Tabel 9.2 memberikan arti sinyal status pada antarmuka floppy.



Gambar 9.10 Contoh sinyal status

TABEL 9.2 Sinyal status pada antarmuka floppy

No.	Sinyal status	Arti
1	Track 0	Head baca/tulis ditempatkan pada track 0
2	Write protect	Diskette pada FDD dalam keadaan terproteksi
3	Ready	FDD siap melakukan operasi
4	Fault	Terdapat kondisi tak normal pada FDD; misalnya kedua head terpilih secara bersamaan
5	Index	Terdeteksi adanya index hole; pada setiap rotasi, index hole pertama yang terdeteksi

Fungsi Pengontrol Perangkat

Fungsi pengontrol perangkat secara keseluruhan adalah sebagai berikut:

- Menerima perintah (*command*) dari CPU.
- Menganalisis perintah dan mengeksekusinya.
- Menerima sinyal status dari perangkat dan melakukan tindakan yang tepat/sesuai.
- Mentransfer data dari CPU/memori ke perangkat.
- Mentransfer data dari perangkat ke CPU/memori.
- Mengubah format data yang diterima dari perangkat misalnya serial ke paralel.
- Mengubah format data yang diterima dari CPU/memori misalnya paralel ke serial.

- Membangkitkan error checking code (parity bit, CRCC atau ECC) selama operasi penulisan.
- Memeriksa error pada data yang diterima dari perangkat.
- Melakukan pembatalan eksekusi command pada semua error.
- Mencoba kembali perintah pada semua error.
- Melapor pada CPU pada akhir eksekusi perintah.

Pengontrol perangkat dirancang untuk hanya melakukan fungsi-fungsi tertentu bergantung pada perangkat dan sistem.

Jenis-Jenis Perintah

Fungsi utama pengontrol perangkat adalah melakukan eksekusi perintah yang ditransmisikan oleh CPU/software. Jenis-jenis perintah tersebut adalah:

- Perintah transfer data
- Perintah transfer status
- Perintah kontrol posisi
- Perintah diagnostik
- Perintah pemilihan mode

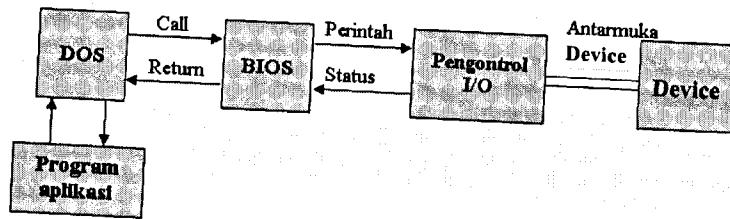
9.1.2 I/O driver

I/O driver merupakan program yang melakukan berbagai operasi I/O dengan memberikan serangkaian perintah yang sesuai ke perangkat I/O pengontrol peripheral. Berikut adalah operasi-operasi tertentu yang dilakukan oleh beberapa I/O driver:

- menampilkan pesan pada CRT
- mencetak sejumlah baris oleh printer
- membaca file dari floppy diskette
- menampilkan isi dari suatu lokasi memori
- menyimpan isi memori ke hard disk

Operasi I/O dapat dilakukan dengan memanggil I/O driver yang relevan dan melewatkannya parameter-parameter relevan untuk operasi. Setelah menyelesaikan operasi I/O, I/O driver kembali mengontrol program terpanggil dan melewatkannya kembali parameter-parameter mengenai penyelesaian operasi. Gambar 9.11 mengilustrasikan tiga deretan antar-

maka antara pengontrol perangkat dan program aplikasi. Istilah "BIOS" (*Basic Input Output control System*) menunjukkan kumpulan dari I/O driver.



Gambar 9.11 Tiga deretan antarmuka

Hardware versus Software

Hardware pengontrol I/O dan software I/O driver sama-sama berfungsi sebagai antarmuka I/O. Pada kenyataannya, fungsi-fungsi tersebut di-share antara hardware (pengontrol I/O) dengan software (I/O driver). Proporsi sharing ditetapkan oleh arsitek dengan mempertimbangkan beberapa aspek seperti biaya dan kinerja. Bila fungsi-fungsi lain didelegasikan pada program dalam I/O driver, kecepatan operasi menurun tetapi biaya hardware juga menurun sebab sirkuit di dalam pengontrol I/O sedikit.

I/O driver bagi perangkat peripheral dasar disediakan oleh komputer personal yang merupakan bagian dari BIOS yang secara fisik disimpan di dalam chip ROM. I/O driver untuk peripheral lainnya diberikan pada floppy diskette atau CD (*compact disc*). Hal ini perlu di-instal ke dalam hard disk untuk dibawa ke dalam RAM setelah booting.

9.1.3 Perangkat Keras Pengontrol

Ada beberapa metode yang digunakan dalam perancangan hardware pengontrol I/O:

- Desain hardwired (*random logic*)
- Desain berbasis LSI fungsi khusus
- Desain berbasis mikroprosesor
- Custom built IC* atau *gate array*.

Desain Hardwired

Pada metode ini, desain perangkat keras pengontrol mempunyai sirkuit yang menggunakan komponen-komponen hardware yang standar. Pada umumnya pengontrol perangkat yang lama termasuk dalam jenis ini.

Pengontrol Berbasis LSI Fungsi Khusus

Pada metode ini, IC LSI yang khusus dapat diprogram, dirancang secara eksklusif untuk penggunaan pengontrol I/O khusus. Keuntungan metode ini adalah mengurangi ukuran hardware pengontrol, waktu desain dan biaya. Pada umumnya mikrokomputer menggunakan pengontrol floppy disk berbasis NEC 765 (atau Intel 8272) yang merupakan pengontrol floppy disk yang bersifat dapat diprogram. Pengontrol CRT pada komputer sebelumnya menggunakan IC 6845 yang merupakan IC pengontrol CRT. Pengontrol komunikasi serial (port serial asinkron) menggunakan IC UART 8250 (*Universal Asynchronous Receiver Transmitter*). Desain pengontrol hard disk menggunakan WD1010A yang merupakan IC yang didedikasikan untuk pengontrol hard disk.

Pengontrol Berbasis Mikroprosesor

Pada metode ini, semua fungsi-fungsi pengontrol diperoleh dari pemrograman mikroprosesor yang didedikasikan. Hal ini memudahkan desain dan mengurangi biaya hardware. Metode ini biasanya disertai dengan fungsi-fungsi kompleks yang banyak yang dieksekusi oleh pengontrol I/O. Namun, kecepatan operasi dari pengontrol yang demikian lebih rendah daripada pengontrol metode lain. Antarmuka keyboard pada IBM PC/AT berdasarkan pada prinsip ini. Mereka menggunakan 8042 yang merupakan IC mikrokontroler, sebuah mikroprosesor sederhana yang di dalamnya dilengkapi dengan memori dan I/O port.

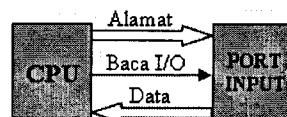
IC Pengontrol Custom Built

Saat ini, sejumlah pengontrol telah dikembangkan seperti *special custom built IC* yang menggabungkan semua sirkuit yang diperlukan bagi pengontrol. Hal ini dikenal sebagai IC ASIC (*Application Specific Integrated Circuit*). Keuntungan metode ini adalah mengurangi waktu pengembangan pengontrol dan ukuran pengontrol.

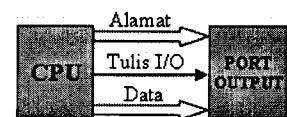
9.2 PORT I/O

Sebuah port I/O adalah suatu unit hardware yang dapat dialami program di mana CPU dapat mentransfer informasi. Setiap port mempunyai alamat sendiri-sendiri yang digunakan ketika berkomunikasi satu sama lain. Sebuah port bisa berupa unit hardware yang berdiri sendiri (*independen*) atau berupa bagian dari suatu unit hardware.

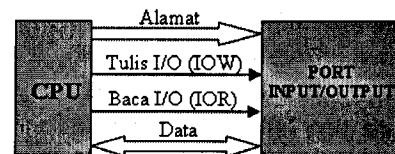
Port input menyuplai data ke bus sedangkan port output menerima data dari bus. Sebuah port input memberikan data ke bus bila dia menerima alamatnya dan sinyal baca I/O (IOR) seperti yang ditunjukkan pada Gambar 9.12. Sebuah port output menerima data dari bus bila dia menerima alamatnya dan sinyal tulis I/O (IOW) seperti yang ditunjukkan pada Gambar 9.13. Sebuah port yang mengirim dan menerima data disebut port input/output (port I/O) atau port dua-arah (*bidirectional*) seperti yang ditunjukkan pada Gambar 9.14. Dalam hal ini ada dua port (port input dan output) dengan alamat yang sama.



Gambar 9.12 Pembacaan dari port input



Gambar 9.13 Penulisan ke port output



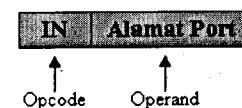
Gambar 9.14 Komunikasi dengan port dua-arah

9.2.1 Instruksi IN

Program membaca data dari suatu port dengan menggunakan instruksi IN yang mempunyai format seperti pada Gambar 9.15. Fungsi instruksi IN adalah mentransfer data dari port input ke register CPU. Operasi-operasi yang dilakukan oleh CPU ketika mengeksekusi instruksi IN adalah sebagai berikut:

1. CPU mengirimkan alamat port. Hanya port yang mempunyai alamat yang sesuai yang terpilih.
2. CPU membangkitkan sinyal IOR. Port yang terpilih pada langkah 1 yang merespons sinyal IOR dan memberikan datanya pada bus.
3. CPU membaca data dari bus dan me-load data pada register prosesor.

Data pada register dapat disalin ke memori dengan instruksi lain seperti STORE atau MOVE.

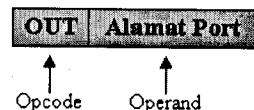


Gambar 9.15 Format instruksi IN

9.2.2 Instruksi OUT

Anggap data yang akan dikirim ke port output adalah di dalam lokasi memori X. Langkah pertama adalah data ini ditransmisikan dari lokasi memori X ke register di dalam CPU dengan instruksi MOVE atau LOAD. Berikutnya instruksi OUT diberikan dengan format yang diberikan pada Gambar 9.16. Fungsi instruksi OUT adalah mentrasfer data dari register ke port output. Operasi-operasi yang dilakukan oleh CPU untuk instruksi OUT adalah sebagai berikut:

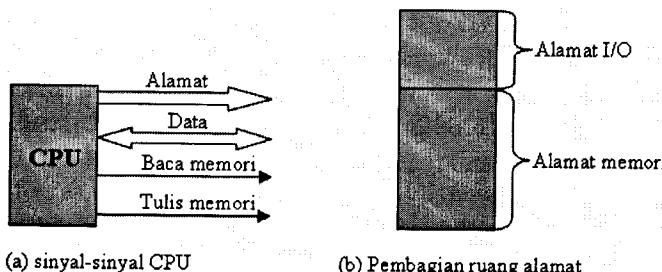
1. CPU mengirimkan alamat port. Hanya port yang mempunyai alamat yang sesuai yang terpilih.
2. CPU mengirimkan data (dari akumulator) pada bus.
3. CPU mengirimkan sinyal IOW. Port yang terpilih pada langkah 1 merespons sinyal IOW dan menghasilkan data dari bus.



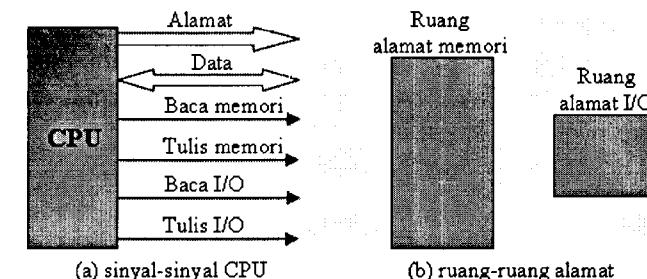
Gambar 9.16 Format instruksi OUT

9.2.3 Memory Mapped I/O dan Direct I/O

Jenis-jenis pengalaman port I/O yang telah dibahas di atas dikenal dengan *Direct I/O* atau *I/O mapped I/O*. Pada skema tersebut port I/O diperlakukan berbeda dengan lokasi memori dan program menggunakan instruksi IN dan OUT untuk berkomunikasi dengan port-port. Terdapat skema lain yang dikenal dengan *memory mapped I/O*. Pada skema ini, port-port I/O diperlakukan seperti layaknya lokasi-lokasi memori. Karena itu instruksi IN dan OUT tidak digunakan. Mikroprosesor tidak harus mengetahui apakah yang diakses tersebut adalah lokasi memori atau port I/O. Hanya program saja yang sadar akan hal ini. Total ruang alamat mikroprosesor didistribusikan ke dalam ruang alamat I/O dan ruang alamat memori. Karena itu, kapasitas memori efektif berkurang sebanyak ruang alamat yang dialokasikan untuk port-port I/O. Gambar 9.17 dan Gambar 9.18 mengilustrasikan konsep *memory mapped I/O* dan *Direct I/O*. Pada suatu komputer, skema-skema yang digunakan dapat beragam. Misalnya pada IBM PC, menggunakan skema direct I/O, walaupun Intel 8088 mampu mendukung kedua skema tersebut.



Gambar 9.17 Memory mapped I/O



Gambar 9.18 Direct I/O

9.3 TRANSFER SINKRON DAN ASINKRON

Ketika dua unit harus berkomunikasi satu dengan lainnya untuk transfer data, biasanya salah satunya menjadi *master* dan satunya lagi menjadi *slave*. Transfer data satu word dilakukan dalam satu periode clock atau kadang-kadang lebih dari satu periode clock. Ada dua jenis transfer data bergantung pada mekanisme dari pewaktuan data: sinkron dan asinkron. Transfer sinkron memungkinkan di antara dua unit masing-masing mengetahui tingkah laku satu sama lain. Transfer asinkron mengizinkan komunikasi antara dua jenis unit yang tidak dikenal.

9.3.1 Transfer Sinkron

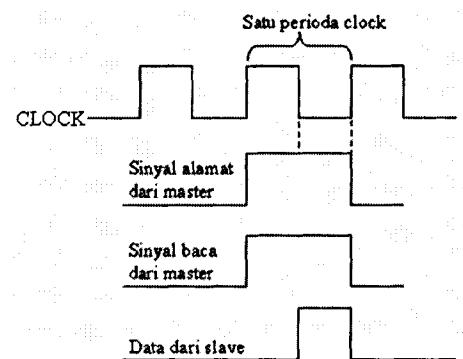
Pada transfer metode sinkron, pengiriman dan penerimaan unit adalah disuplai dengan sinyal clock yang sama. Master melakukan suatu urutan aksi untuk transfer data dalam urutan yang telah ditetapkan sebelumnya; setiap aksi adalah sinkron terhadap tebing naik atau tebing turun dari clock. Master didesain untuk mensuplai data pada suatu waktu ketika slave dinyatakan siap untuk itu. Jika diperlukan, master akan mengawali dengan delay yang cukup untuk menjaga respons yang lambat dari slave, tanpa permintaan slave. Bila master mengirim data ke slave, data ditransmisikan oleh master tanpa mengharapkan pengakuan (*acknowledgement*) dari slave sebagai tanda terima/respons. Dengan cara yang sama, bila master membaca data dari slave, slave tidak menginformasikan bahwa data telah ditempatkan atau master tidak memberikan pengakuan bahwa dia telah membaca data. Baik master maupun slave melakukan ‘penempatan’ dan ‘penerimaan’ data pada tebing sinyal clock yang telah ditetapkan. Karena

master dan slave mengetahui waktu respons masing-masing, maka tidak mungkin terjadi kekeliruan. Sebelum memulai transfer data, master secara logika akan memilih slave dengan mengirimkan alamat slave atau mengirimkan sinyal pemilih perangkat ("device select") ke slave. Perangkat dipilih tetapi tidak ada pengakuan dari slave ke master.

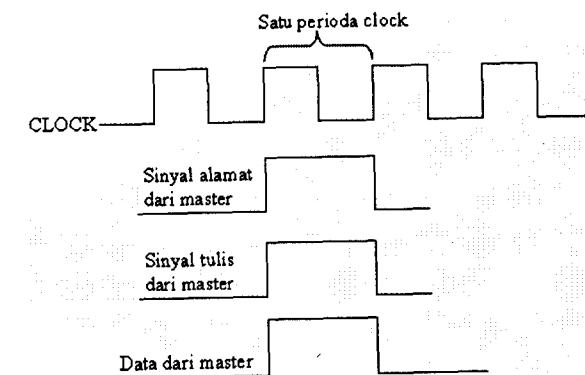
Gambar 9.19 menunjukkan diagram pewaktuan untuk operasi baca transfer sinkron. Master mentransmisikan alamat slave dan sinyal baca pada tebing naik clock. Slave menempatkan data pada tebing turun clock. Master menghapus alamat dan sinyal baca pada tebing naik clock berikutnya dan slave juga menghapus datanya pada tebing itu. Operasi pembacaan semuanya dilakukan dalam satu periode clock. Gambar 9.20 menunjukkan diagram pewaktuan untuk operasi penulisan sinkron. Master menempatkan tiga macam item berikut pada tebing naik clock:

1. Alamat slave
2. Data
3. Sinyal tulis

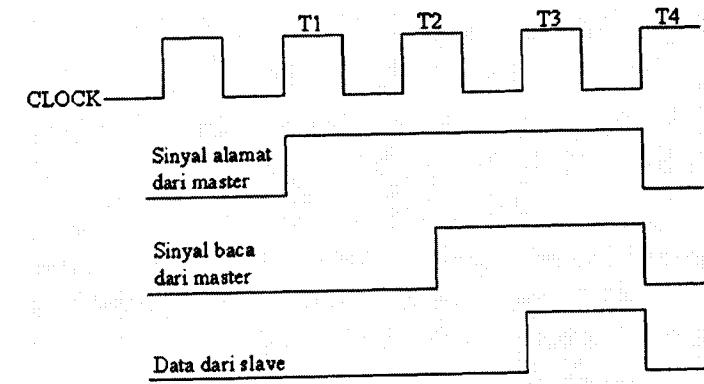
Slave memperoleh data sebelum akhir dari siklus clock karena master menghapus semua sinyal-sinyalnya pada clock berikutnya. Jika slave tidak dapat merespons dalam suatu clock karena responsnya tinggi, maka master mengizinkan siklus transfer data untuk menyebarkan pada periode clock jamak. Gambar 9.21 menunjukkan suatu operasi baca sinkron siklus jamak.



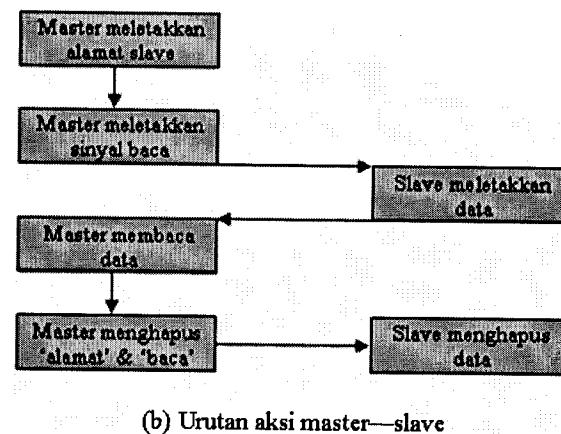
Gambar 9.19 Diagram pewaktuan operasi baca transfer sinkron



Gambar 9.20 Diagram pewaktuan operasi tulis transfer sinkron



(a) Diagram pewaktuan



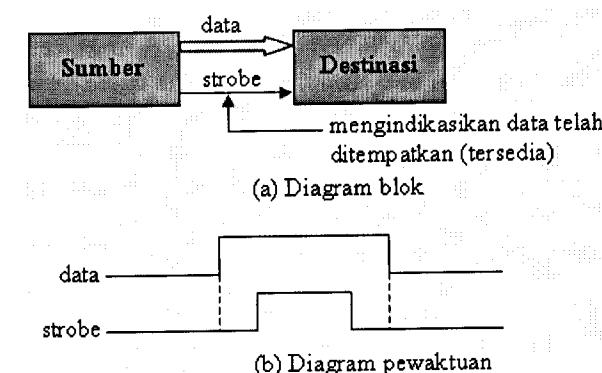
Gambar 9.21 Pembacaan sinkron dalam multiclock

9.3.2 Transfer Asinkron dan Handshaking

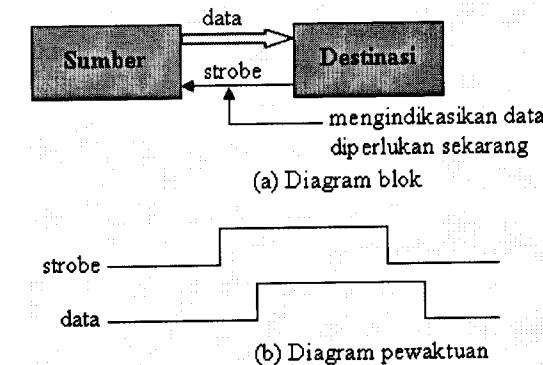
Pada metode transfer asinkron, tidak ada clock bersama antara master dan slave. Mereka mengikuti beberapa urutan protokol pengakuan selama rangkaian transfer data. Bila master mengirim data, dia menginformasikan slave menggunakan sinyal ACCEPT STROBE yang mengindikasikan bahwa dia telah menempatkan data pada saluran data. Saat terjadi respons, slave menerima data dari saluran data. Gambar 9.22 menunjukkan suatu unit yang diinisiasi transfer asinkron. Sumber menjaga delay pewaktuan yang tepat antara sinyal-sinyal data aktual dan ACCEPT STROBE. Dia menempatkan data pertama, dan setelah beberapa delay, kemudian membangkitkan sinyal STROBE. Dengan cara yang sama, sebelum penghapusan data dia menghapus strobe dan setelah beberapa delay dia menghapus data. Dua leading dan trailing akhir delay ini, menjamin transfer data antara pembangkitan dan penerimaan akhir. Unit destinasi (tujuan) juga dapat memulai transfer data dengan mengirimkan sebuah REQUEST STROBE ke unit pengiriman seperti yang ditunjukkan pada Gambar 9.23. Ketika respons, unit unit sumber menempatkan data pada saluran data. Setelah menerima data, destinasi akhir menghapus REQUEST STROBE. Hanya setelah REQUEST STROBE dirasakan telah terhapus, maka sumber menghapus data. Protokol ini berulang untuk setiap data (byte atau word). Ada risiko dalam metode ini. Anggaplah sumber memulai transfer asinkron.

Jika pada sumber/destinasi terjadi kesalahan (fault), maka salah satu masalah berikut ini akan terjadi:

1. Destinasi tidak merasakan sinyal STROBE dari sumber. Karena itu dia tidak memperoleh data yang ditransmisikan oleh sumber. Hal ini tidak diketahui oleh sumber.
2. Sumber secara tetap membangkitkan sinyal STROBE bila transfer tidak diperlukan. Sekarang destinasi secara sederhana menerima isi saluran data seperti data sesungguhnya tanpa mengetahui masalah yang terjadi.

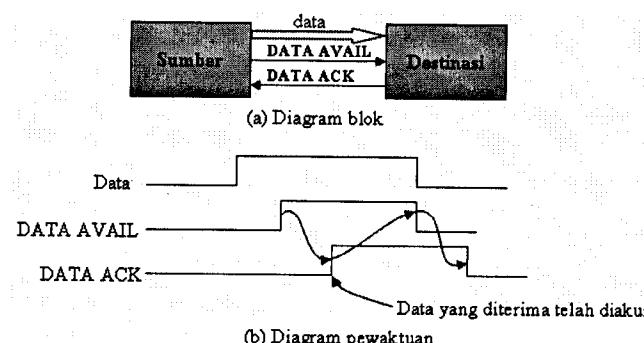


Gambar 9.22 Transfer data asinkron inisiasi sumber

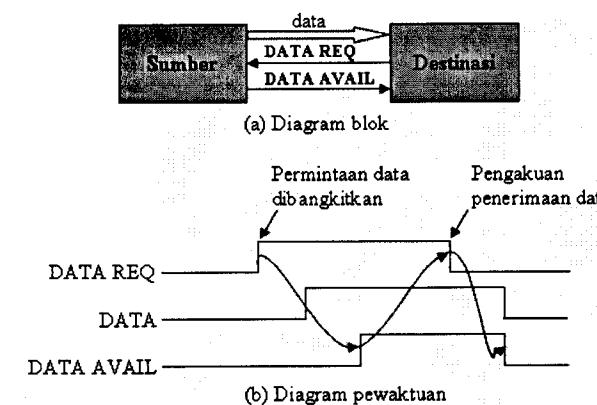


Gambar 9.23 Transfer data asinkron inisiasi destinasi

Masalah yang sama juga dapat terjadi pada transfer yang dilakukan destinasi. Untuk mengatasi hal ini, suatu protokol jabat tangan (*handshaking*) biasa dilakukan antara sumber dan destinasi. Pada protokol handshaking, baik sumber dan destinasi membangkitkan sinyal strobe dan sinyal pengakuan (*acknowledgement*). Gambar 9.24 menunjukkan sumber yang melakukan inisiasi protokol handshaking. Sumber menempatkan data pertama dan setelah beberapa delay kemudian, sinyal DATA AVAIL diberikan. Pada saat merasakan (*sensing*) sinyal DATA AVAIL, maka destinasi menerima data dan kemudian memberikan sinyal DATA ACK yang mengindikasikan bahwa data telah diterima. Pada saat merasakan sinyal DATA ACK, maka sumber menghapus data dan sinyal DATA AVAIL. Pada saat merasakan sinyal DATA AVAIL sudah hilang, maka destinasi menghapus sinyal DATA ACK. Karena itu baik sumber dan destinasi merespons naik atau turunnya akhir sinyal, suatu aksi abnormal di satu ujung adalah bukan merupakan urutan aslinya. Gambar 9.25 menunjukkan destinasi yang melakukan inisiasi protokol handshaking. Destinasi pertama mengirimkan sinyal DATA REQ. Pada saat merasakan sinyal ini, sumber menempatkan data dan juga memberikan sinyal DATA AVAIL. Pada saat merasakan sinyal DATA AVAIL, maka destinasi memperoleh data dan kemudian menghapus sinyal DATA REQ. Pada saat merasakan hal ini, maka sumber menghapus data dan sinyal DATA AVAIL.



Gambar 9.24 Protokol handshaking inisiasi sumber



Gambar 9.25 Protokol handshaking inisiasi destinasi

Keuntungan transfer asinkron adalah:

1. Karena sumber dan destinasi mengikuti handshaking, maka keandalan transfer terjamin
2. Suatu unit destinasi yang lambat pada bus tidak memperlambat transfer data ke unit destinasi cepat.

Kekurangan transfer asinkron adalah:

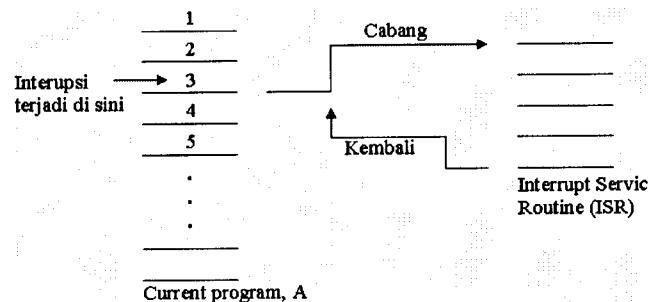
1. Desain lebih rumit dan mahal karena adanya sinyal-sinyal tambahan dan sequence.
2. Unit destinasi yang lambat pada bus dapat menahan bus kapan pun dia mendapat kesempatan untuk berkomunikasi.

9.4 PENANGANAN INTERUPSI PADA PC

Interupsi adalah sebuah sinyal atau kejadian (*event*) yang meminta perhatian CPU, di mana CPU akan menunda sementara eksekusi program yang sedang berjalan (*current program*) dan mengeksekusi program lain yang berkaitan dengan interupsi. Penanganan interupsi adalah suatu fungsi yang dilakukan dengan melibatkan gabungan hardware dan software. Pada bagian ini kita akan membahas aspek-aspek penanganan interupsi.

9.4.1 Konsep Interupsi

Aktivitas CPU pada satu waktu adalah pemrosesan instruksi kecuali bila CPU dalam keadaan HALT, maka CPU menjalankan beberapa atau program lain. Interupsi adalah suatu sinyal atau suatu event yang digunakan untuk meminta CPU supaya menunda program yang sedang berjalan dan mengambil program lain yang berhubungan dengan interupsi. Program ini dikenal dengan Rutin Layanan Interupsi (*interrupt service routine*, ISR). Gambar 9.26 menunjukkan ilustrasi konsep interupsi. Ketika proses interupsi telah selesai, maka CPU umumnya akan melanjutkan program sebelumnya yaitu tempat terjadinya interupsi ketika program utama ditinggalkan. Karena itu sebelum pengambilan ISR, CPU menyimpan alamat instruksi berikutnya (isi program counter) pada saat bercabang ke ISR. Penyimpanan dapat dilakukan di dalam CPU atau dalam memori.



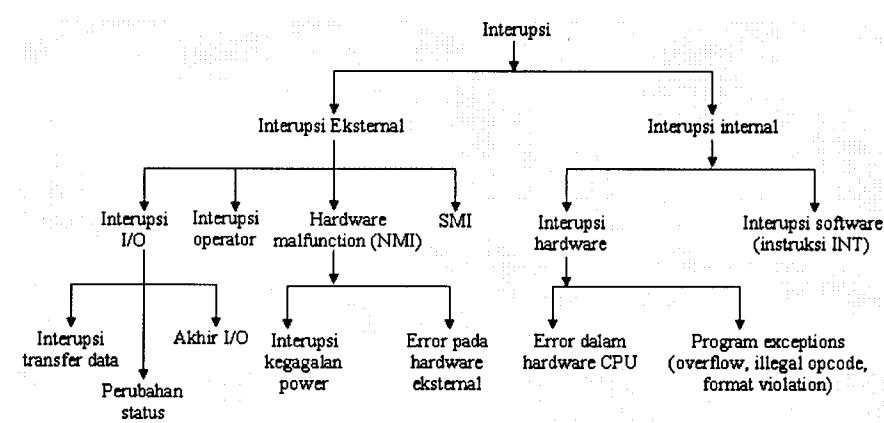
Gambar 9.26 Konsep interups

9.4.2 Jenis-Jenis Interupsi

Secara umum ada dua jenis interupsi yaitu interupsi internal dan eksternal. Gambar 9.27 menunjukkan jenis-jenis interupsi dan pada Tabel 9.3 dijelaskan perbedaannya.

TABEL 9.3 *Interupsi internal dan eksternal*

Interupsi Eksternal	Interupsi Internal
1. Interupsi eksternal dibangkitkan oleh sirkuit hardware di luar CPU.	1. Interupsi internal dibangkitkan dalam CPU.
2. Interupsi ini pada dasarnya adalah interupsi hardware.	2. Interupsi ini dapat berupa hardware atau software.
3. Interupsi hardware dibangkitkan oleh sirkuit.	3. Interupsi software dibangkitkan oleh sebuah instruksi di dalam program.



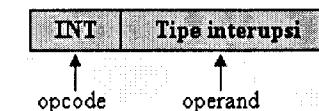
Gambar 9.27 Jenis-jenis interups

Definisi jenis-jenis interupsi diberikan sebagai berikut:

- **Interupsi Transfer Data.** Interupsi transfer data dibangkitkan oleh pengontrol I/O untuk mengindikasikan ke CPU bahwa dia siap untuk mentransfer data dengan CPU. Selama eksekusi perintah READ, interupsi merupakan suatu indikasi pembacaan data dengan pengontrol perangkat. Pada sisi lain, selama perintah WRITE, interupsi merupakan permintaan untuk satu byte data oleh pengontrol perangkat. Tepatnya, program CPU (ISR) akan membaca satu byte data dari pengontrol atau memberikan satu byte data ke pengontrol .

- **Interupsi Akhir I/O.** Interupsi akhir I/O (end of I/O interrupt) dibangkitkan oleh suatu pengontrol I/O pada penyelesaikan eksekusi dari suatu perintah untuk mengindikasikan ke CPU (*software*) bahwa operasi I/O telah selesai. Hal ini dapat dinyatakan dalam dua kondisi berikut:
 1. Operasi I/O berhasil
 2. Operasi I/O dibatalkan karena beberapa kondisi abnormal.
 ISR (I/O driver *software*) membaca status yang tepat dari pengontrol I/O dan berdasarkan pada status, penyelesaian alamiah operasi I/O dikonfirmasikan.
- **Interupsi Perubahan Status.** Interupsi ini digunakan untuk menginformasikan software mengenai perubahan status dari suatu perangkat I/O. Misalnya suatu perangkat I/O dapat berubah dari kondisi READY ke NON-READY. Saat respons, pengontrol perangkat dapat membangkitkan interupsi.
- **Interupsi Kegagalan Power.** Interupsi ini dibangkitkan oleh suatu hardware khusus yang memantau fluktuasi suplai catu daya. Interupsi ini adalah informasi awal bahwa power akan gagal (*padam*). Selanjutnya ISR dapat menyimpan status CPU dalam memori cadangan baterai. Atau dia dapat memulai operasi suplai power cadangan atau generator, dan sebagainya.
- **Interupsi Operator.** Interupsi ini adalah untuk menginterupsi sistem operasi untuk melakukan aksi khusus. Umumnya digunakan hanya pada sistem multiprogramming.
- **Interupsi Malfungsi Hardware Eksternal.** Jika ada suatu malfungsi seperti kegagalan memori atau error transfer data, maka interupsi ini dibangkitkan.
- **Interupsi Error Hardware Internal CPU.** Interupsi ini merupakan hasil dari suatu deteksi beberapa hardware yang mengalami malfungsi seperti kegagalan data dalam CPU.
- **Program Exception Interrupt.** Interupsi ini disebabkan oleh kondisi abnormal yang dialami oleh CPU selama eksekusi program. Hal ini tidak disebabkan oleh error tetapi karena situasi khusus yang disebabkan oleh program. Beberapa program dengan kasus luar biasa seperti illegal opcode, pelanggaran format instruksi, pelanggaran format operand dan overflow.
- **Interupsi Software.** Interupsi software dibuat oleh program agar CPU bercabang secara sementara dari current program ke program

lain. Instruksi INT (interupsi) digunakan untuk pembangkitan interupsi. Gambar 9.28 menunjukkan format instruksi INT. Nomor tipe interupsi dalam instruksi INT adalah ekivalen dengan kode vektor untuk interupsi hardware. Penggunaan nomor ini, CPU mengakses lokasi yang sesuai dengan tabel vektor dalam RAM yang menyimpan alamat awal ISR (subprogram).

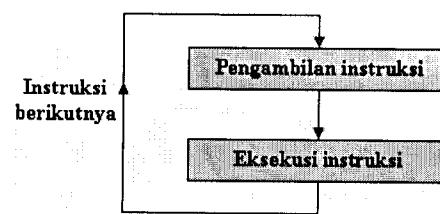


Gambar 9.28 Format instruksi INT

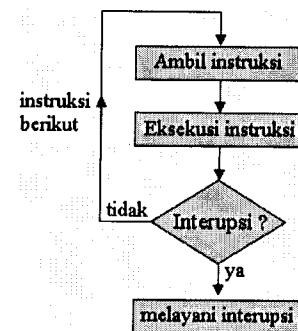
- **System Management Interrupt (SMI).** SMI merupakan interupsi untuk keperluan khusus yang digunakan untuk mengurangi konsumsi power dengan mematikan susbsistem yang tidak digunakan dan memberikan power pada subsistem yang hanya digunakan. Proses ini dilakukan secara kontinu.

9.4.3 Penginderaan Interupsi

Pemrosesan instruksi melibatkan pengambilan instruksi dan eksekusi instruksi dalam suatu program. (Gambar 9.29). CPU mengindra (*sense*) kehadiran sebuah interupsi setelah phase eksekusi suatu instruksi (*current instruction*) selesai (Gambar 9.30). Jika ada suatu interupsi, CPU tidak memulai phase pengambilan pada instruksi berikutnya (*next instruction*), tetapi CPU mulai melayani interupsi. Umumnya CPU mulai melayani interupsi setelah selesai melakukan phase eksekusi tanpa melihat kapan suatu interupsi terjadi. Namun, ada pengecualian dalam hal ini, yaitu jika pemrosesan yang sedang dilakukan adalah sebuah instruksi kompleks, di mana eksekusi instruksi tersebut memerlukan waktu yang panjang, maka CPU menghentikan sementara program yang sedang diproses dan kemudian segera melayani interupsi.



Gambar 9.29 Siklus instruksi



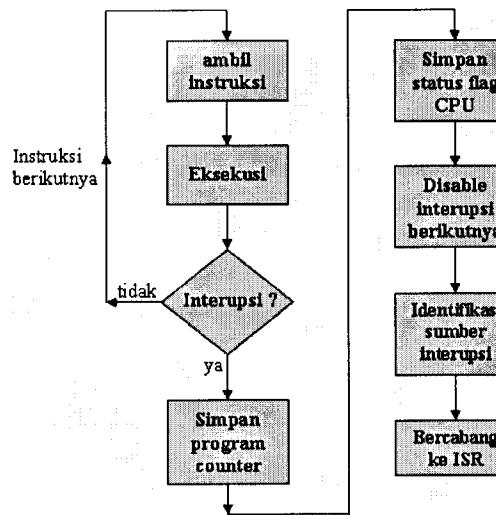
Gambar 9.30 Pengindraan interupsi oleh CPU

9.4.4 Pelayanan Interupsi

Ada tiga aksi berbeda yang dilakukan secara otomatis oleh hardware CPU pada pengindraan sebuah interupsi (Gambar 9.31), yaitu:

1. Penyimpanan status CPU
2. Menemukan penyebab interupsi
3. Bercabang ke ISR

Ketiga aksi ini dilakukan oleh hardware CPU dan tidak ada software yang terlibat dalam aksi ini.



Gambar 9.31 Urutan pelayanan interupsi

9.4.5 Kembali dari ISR

ISR bertanggung jawab mengembalikan kontrol ke lokasi program yang diinterupsi bila ISR telah selesai. Teknik sederhana yaitu menggunakan instruksi IRET (*interrupt return*—kembali dari interupsi) dalam ISR. Dengan menggunakan instruksi IRET, ISR mengarahkan CPU kembali ke program yang diinterupsi. Status CPU yang tersimpan dalam stack dipulihkan kembali seperti sebelum ada interupsi. Sebagai hasilnya, flag pada saat sebelum terjadinya interupsi disimpan kembali (*restore*) dan CPU melanjutkan eksekusi program yang mengalami penundaan ketika interupsi dilayani.

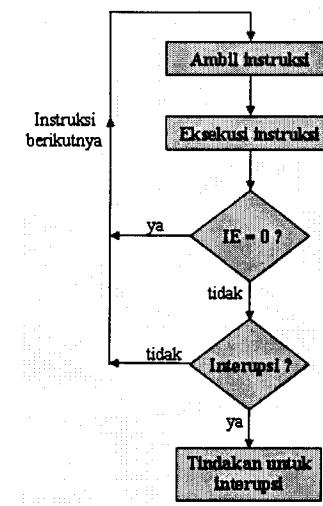
9.4.6 Enable dan Disable Interupsi

Jika suatu program tidak ingin melakukan ada interupsi, dia menginformasikan CPU tidak melayani interupsi. Hingga program memberikan perintah lain ke CPU membolehkan interupsi, CPU mengabaikan interupsi dan karena itu interupsi yang datang pada saat itu ditunda (menunggu).

Sebuah flag yang dikenal dengan “*Interrupt Enable*” (IE) digunakan untuk tujuan ini. Bila flag ini “1”, maka CPU merespons adanya interupsi. Bila flag ini “0”, maka CPU mengabaikan adanya interupsi. Gambar 9.32

mengilustrasikan siklus instruksi yang dimodifikasi memperhitungkan *interrupt enable flag*. Untuk men-set flag IE, program harus memberikan instruksi “*Enable interrupt*” (EI). Ketika instruksi ini dieksekusi, CPU men-set flag IE. Untuk me-reset flag IE, program harus memberikan instruksi “*Disable Interrupt*” (DI). Ketika instruksi ini dieksekusi, CPU me-reset flag IE. Jadi tingkah laku CPU yang berkenaan dengan pelayanan interupsi adalah dikontrol oleh program yang sedang dieksekusi. Ada dua situasi khusus bila CPU me-reset flag IE:

1. Selama urutan reset (*power on* atau *manual reset*), CPU membersihkan flag IE.
2. Selama pelayanan interupsi, CPU me-reset flag IE dengan segera setelah penyimpanan status CPU dan sebelum bercabang ke ISR.



Gambar 9.32 Pengindraan interupsi dan interrupt enable flag

9.4.7 Interupsi Tersarang

Ketika CPU sedang mengesekusi sebuah rutin layanan interupsi, jika dia mengikuti interupsi lain, dia dikenal dengan interupsi tersarang (*interrupt nesting*). Anggaplah awalnya, CPU sedang mengeksekusi program A ketika interupsi 1 terjadi. Setelah CPU melakukan aksi pada layanan interupsi (menyimpan status CPU, mengidentifikasi sumber interupsi dan bercabang

ke ISR), dia memulai eksekusi ISR1. Sekarang mari kita katakan interupsi 2 terjadi. CPU melakukan lagi aksi layanan interupsi dan memulai eksekusi ISR2. Ketika dia mengeksekusi ISR2, interupsi 3 terjadi. CPU melakukan lagi layanan interupsi dan memulai ISR3. Setelah ISR3 selesai, CPU melanjutkan bagian ISR2 yang tersisa. Setelah ISR2 selesai, CPU mengeksekusi bagian ISR1 yang tersisa. Setelah ISR1 selesai, CPU kembali ke program A dan melanjutkan pada tempat terjadinya pencabangan sebelumnya.

9.4.8 Prioritas Interupsi

Permintaan interupsi berasal dari berbagai sumber yang dikoneksikan sebagai input ke pengontrol interupsi. Segera setelah pengontrol interupsi mengindra keberadaan interupsi dari satu atau lebih pemohon interupsi, dengan segera dia memberikan sinyal interupsi ke CPU. Pengontrol interupsi menetapkan prioritas tetap untuk berbagai pemohon interupsi. Misalnya, IRQ0 ditetapkan sebagai prioritas tertinggi di antara delapan pemohon interupsi. Penetapan prioritas menurun dari IRQ0 ke IRQ7. IRQ7 mempunyai prioritas terendah sehingga dia hanya dilayani ketika tidak ada pemohon interupsi lagi yang lain.

9.4.9 Penghalangan Interupsi yang Dapat Dipilih

Anggap program hanya membolehkan interupsi tertentu tetapi menghalangi interupsi lainnya. Hal ini dapat dicapai dengan langkah-langkah, pertama program memberikan instruksi EI sehingga flag IE di-set dan CPU tidak mengabaikan interupsi. Kedua, program mengarahkan pengontrol interupsi untuk mengabaikan masukan pemohon-pemohon interupsi tertentu. Jadi, program telah secara selektif menghalangi interupsi tertentu. Pola penghalang diberikan oleh program, langsung ke pengontrol interupsi, memutuskan interupsi mana yang dihalangi pada suatu saat. Interupsi tertentu dapat mempunyai urgensi yang lebih tinggi secara alami dari program sekarang dan karenanya tidak dapat dihalangi. Interupsi lain mempunyai urgensi yang lebih rendah dari program sekarang yang dihalangi agar program sekarang dieksekusi tanpa terlalu banyak interupsi. Anggap dua interupsi hadir dalam waktu yang sama. Dengan penghalangan interupsi prioritas yang lebih tinggi pada level pengontrol interupsi, interupsi prioritas lebih rendah dilayani terlebih dahulu, sedangkan interupsi prioritas yang lebih tinggi ditunda.

9.4.10 Interupsi yang Tak Dapat Dihalangi

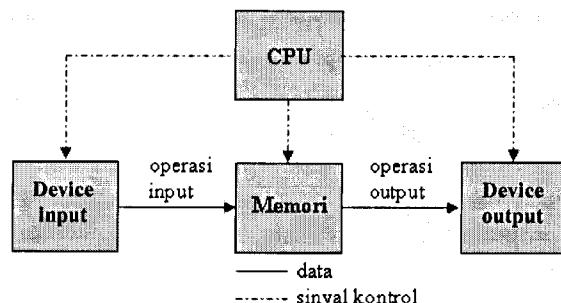
Interupsi yang tak dapat dihalangi (*non-maskable interrupt*, NMI) dibangkitkan untuk keperluan perhatian yang sangat penting dari CPU (ISR). Pengindraan NMI tidak bergantung pada flag IE. Prosesor didesain dengan suatu kode vektor tetap untuk NMI.

9.5. TEKNIK-TEKNIK I/O PADA PC

Komunikasi antara pengguna dan komputer dapat berupa program atau sejumlah tipe data untuk suatu program. Istilah ‘Trasnfer data’ adalah perpindahan informasi antara CPU/memori dan perangkat/peripheral I/O. Transfer data umumnya dilakukan dengan satu word setiap waktu dalam setiap langkah.

Ada dua jenis transfer data seperti yang ditunjukkan pada ilustrasi Gambar 9.33:

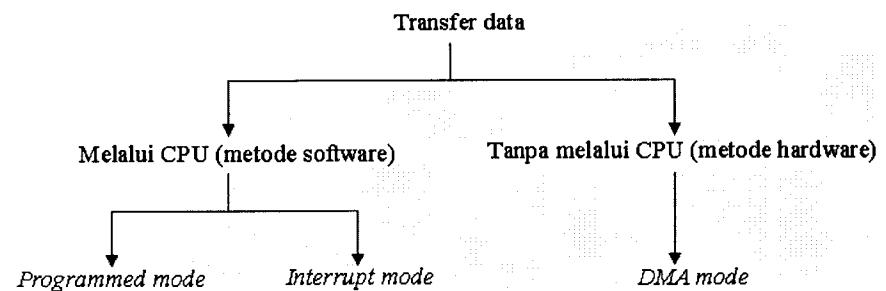
1. Transfer dari perangkat input ke memori
2. Transfer dari memori ke perangkat output



Gambar 9.33 Dua jenis transfer data

Ada dua teknik pelaksanaan transfer data:

1. Transfer data melalui CPU (metode software)
2. Transfer data tanpa melalui CPU (metode hardware)



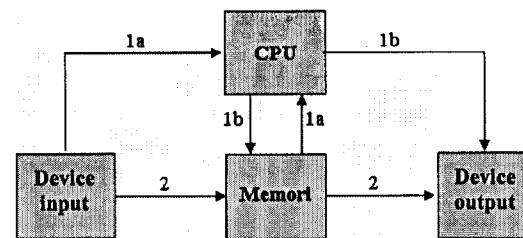
Gambar 9.34 Teknik-teknik transfer data

Gambar 9.34 mengilustrasikan teknik-teknik transfer data, di mana perbedaan utama antara metode software dan metode hardware adalah perluasan pelibatan CPU dalam operasi aktual transfer data. Pada metode software, tugas-tugas mengenai operasi input/output diimplementasikan dalam suatu program/rutin yang dieksekusi oleh CPU. Karena itu, CPU tidak terbebani secara total dalam operasi I/O. Pada metode hardware, program mendelegasikan tanggung jawab pelaksanaan operasi I/O ke unit hardware lain yang disebut pengontrol DMA (*DMA controller*).

Pada ilustrasi Gambar 9.35, ditunjukkan ada dua langkah dalam metode software, misalnya untuk memindahkan sebuah byte data dari input perangkat maka akan mengikuti dua langkah berikut:

1. Baca byte data dari perangkat input ke CPU (langkah 1a)
2. Pindahkan byte data dari CPU ke lokasi memori (langkah 1b)

Dua langkah ini diulangi untuk mentransfer setiap byte data. Semua langkah-langkah ini dicapai dengan pemrograman CPU, yaitu dengan menggunakan instruksi-instruksi yang tepat. Karena itu metode ini dinamakan juga metode software. Pada metode hardware, software CPU tidak terlibat secara aktual dalam transfer byte data. Software hanya menyediakan parameter-parameter awal tertentu ke hardware dan hardware yang melakukan transfer aktual byte data tanpa keterlibatan CPU. Metode ini populer disebut teknik DMA (*Direct Memory Access*).



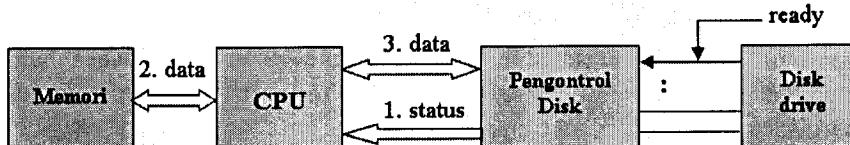
Gambar 9.35 Prinsip-prinsip transfer data

9.5.1 Programmed I/O Mode

Pada *programmed I/O mode*, program atau rutin I/O masing-masing melakukan empat aktivitas untuk setiap byte data yang ditransfer:

1. Pembacaan status perangkat peripheral
 2. Menganalisa apakah perangkat siap untuk mentransfer data atau tidak
 3. Jika perangkat siap, ke langkah 4 untuk mentransfer data, jika perangkat tidak siap maka ke langkah 1 untuk looping sampai perangkat siap untuk mentransfer data
 4. Melakukan transfer data dalam dua langkah. Untuk operasi input, dua langkah berikut dilakukan:
 - a. Pembacaan data dari perangkat input ke CPU
 - b. Penyimpanan data dalam suatu lokasi memori
- Untuk operasi output, dua langkah berikut yang akan dilakukan:
- a. Pembacaan data dari memori ke CPU
 - b. Menyampaikan data ke perangkat output

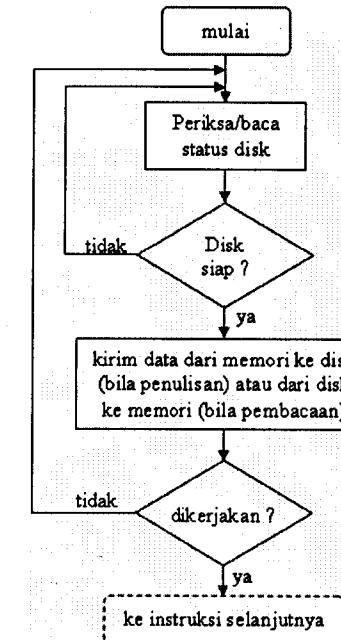
Gambar 9.36 mengilustrasikan diagram blok dan Gambar 9.37 diagram alir transfer data untuk programmed I/O mode.



Langkah 1. Pembacaan status device
 Langkah 2. Transfer data dari/ke memori
 Langkah 3. Pengiriman data ke/dari device

Ketiga langkah tersebut diulangi untuk setiap byte.
 Setelah langkah 1, langkah 2 dikerjakan jika device siap, jika tidak maka langkah 1 diulangi.

Gambar 9.36 Diagram blok programmed I/O mode



Gambar 9.37 Diagram alir transfer data (disk) pada programmed I/O mode

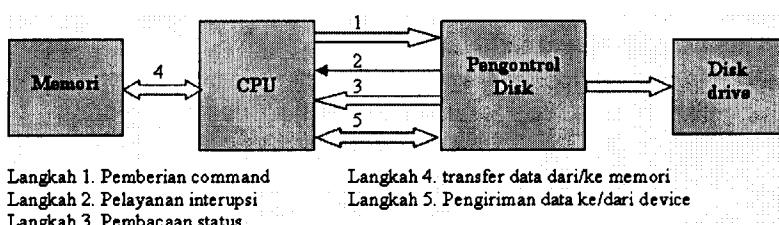
Diagram alir pada Gambar 9.37 ditunjukkan pertama CPU memeriksa status disk dengan membaca suatu register khusus yang dapat diakses dalam memori, atau dengan menyampaikan instruksi I/O khusus bagaimana arsitektur ini mengimplementasikan I/O. Jika disk tidak siap dibaca atau ditulisi, maka proses loop akan kembali dan memeriksa status secara kontinu hingga disk siap. Keadaan ini disebut sebagai "busy-wait". Bila disk akhirnya sudah siap, maka transfer dapat dilakukan antara disk dan CPU. Setelah transfer selesai, CPU memeriksa untuk mengetahui apakah terdapat permitaan komunikasi lain untuk disk. Jika ada, maka proses diulang kembali dan sebaliknya CPU melanjutkan pekerjaan lain.

Kekurangan programmed I/O mode yaitu pada kecepatan transfer data yang bergantung pada jumlah atau berapa kali langkah dari langkah 1 ke 3 diulangi, yang pada akhirnya bergantung pada kecepatan perangkat peripheral tersebut. Jika perangkat lambat, maka I/O routine akan menggunakan waktu yang lama untuk melakukan looping pada tiga langkah tersebut. Dengan kata lain waktu CPU terbuang.

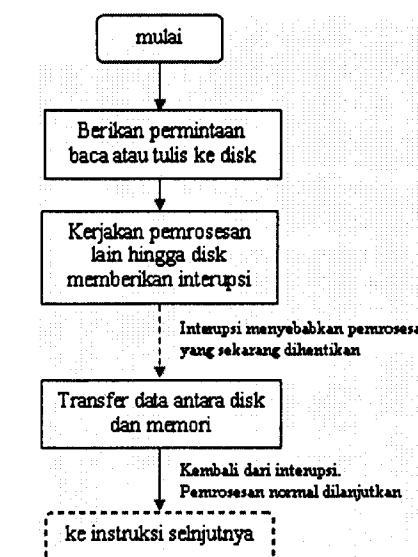
9.5.2 Interrupt Mode

Pada programmed I/O mode, status perangkat dipantau oleh software (routines I/O), sedangkan pada *interrupt mode*, software (routines I/O) tidak menunggu sampai perangkat siap. Bahkan hardware pengontrol perangkat secara kontinu memantau status perangkat dan membangkitkan interupsi ke CPU segera ketika perangkat sudah siap untuk transfer data. Software rutin I/O segera menghentikan CPU. Setelah transfer byte data selesai, rutin I/O melepaskan CPU untuk bebas melakukan program atau rutin lainnya. Bila interupsi berikutnya dibangkitkan lagi, rutin I/O mengambil alih kembali kontrol. Jadi, pada interrupt mode, software (CPU) melakukan transfer data tetapi tidak terlibat dalam pengecekan apakah perangkat siap untuk transfer data atau tidak. Dengan kata lain, langkah 1 sampai langkah 3 pada programmed I/O mode didelegasikan ke perangkat keras pengontrol perangkat. Jadi jelas ini pemanfaatan CPU yang lebih baik. CPU dapat mengeksekusi program lain hingga interupsi diterima dari perangkat.

Untuk operasi transfer data pada interrupt mode, pengontrol perangkat harus mempunyai sejumlah inteligensi tambahan untuk pengecekan status perangkat dan pembangkitan sebuah interupsi kapanpun transfer data diperlukan. Hal ini menghasilkan sirkuit perangkat keras ekstra di dalam pengontrol. Gambar 9.38 dan 9.39 mengilustrasikan transfer data dalam interrupt mode.



Gambar 9.38 Diagram blok interrupt mode



Gambar 9.39 Diagram alir transfer data (disk) pada interrupt mode

9.5.3 DMA Mode

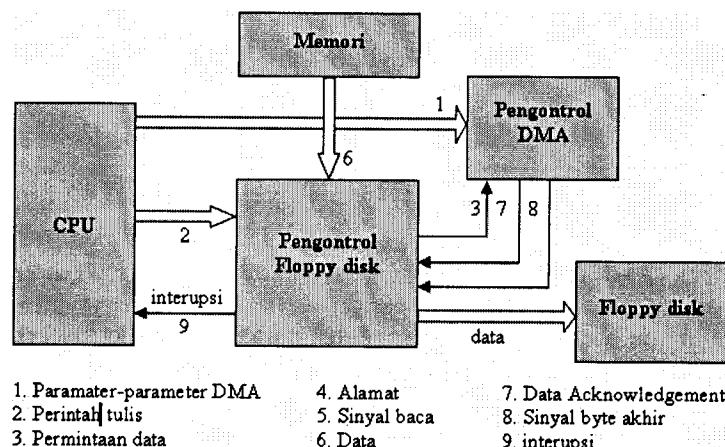
DMA mode dapat melakukan transfer data langsung ke/dari memori. Pada DMA mode, software hanya melakukan ‘inisiasi’ yang melibatkan pengiriman command (perintah) ke pengontrol DMA dan pengontrol perangkat. Operasi aktual yang berhubungan dengan transfer byte data dilakukan oleh pengontrol DMA yang merupakan unit hardware tersendiri (independen). Pengontrol DMA dapat mengakses memori untuk operasi pembacaan atau penulisan tanpa bantuan dari CPU. Pengontrol perangkat meminta pengontrol DMA bahwa satu byte akan ditransfer (antara memori dan pengontrol perangkat) pengganti interupsi CPU. Parameter-parameter DMA berikut disediakan oleh software ke pengontrol DMA:

1. Alamat awal memori
2. Jumlah Byte
3. Arah: input atau output

“Alamat awal memori” menetapkan lokasi memori dari mana byte data disimpan atau dibaca. “Jumlah byte” menetapkan jumlah byte yang akan ditransfer. “Arah” menetapkan apakah transfer data adalah input atau

output. Operasi input melibatkan penerimaan data dari pengontrol perangkat dan penulisannya pada memori. Operasi output melibatkan pembacaan data dari memori dan pensuplaiannya ke pengontrol perangkat.

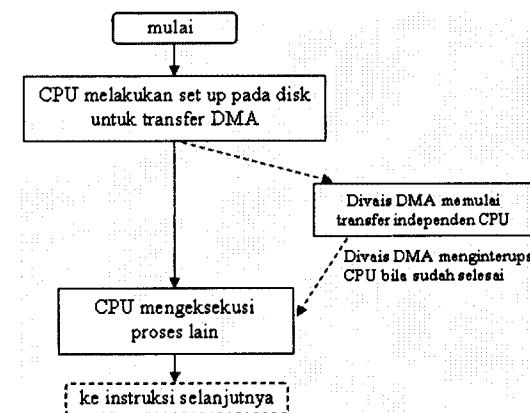
Terlepas dari penyampaian parameter-parameter DMA ke pengontrol DMA, software juga memberikan suatu command dan parameter-parameter command yang relevan ke pengontrol perangkat. Misalnya untuk membaca data dari floppy diskette, command "Read" ditransmisikan ke floppy disk controller beserta parameter-parameter command yang relevan seperti nomor track, nomor Side (head), nomor awal sektor, nomor akhir sektor. Gambar 9.40 dan 9.41 mengilustrasikan transfer data dengan DMA mode.



Gambar 9.40 Diagram blok DMA mode

Pengontrol DMA mempunyai register yang menyimpan parameter-parameter DMA yang ditawarkan oleh software. Ketika parameter DMA diterima, pengontrol DMA siap untuk transfer data tetapi inisiatif berasal dari pengontrol perangkat.

Pada Gambar 9.41 ditunjukkan diagram alir untuk transfer data (disk) DMA mode. CPU melakukan set-up pada perangkat DMA dan memberikan sinyal perangkat untuk mulai transfer. Ketika transfer terjadi, CPU melanjutkan eksekusi proses lain. Bila transfer DMA selesai, perangkat memberitahukan ke CPU melalui interupsi. Sistem yang melakukan DMA juga melakukan interupsi.



Gambar 9.41 Diagram alir transfer data pada DMA mode

Urutan DMA

Pada pembahasan ini kita mengambil sebuah contoh pembacaan satu sektor data dari disket. Anggaplah kita hanya ingin membaca awalnya 400 byte data dari sektor nomor 2 pada track nomor 5 sisi 1. Rutin I/O memberikan parameter-parameter berikut ke pengontrol DMA:

- Alamat awal: 60000
- Jumlah byte: 400
- Arah: input

Rutin I/O juga memberikan perintah berikut dan parameter-parameter perintah ke pengontrol disket:

Perintah READ DATA

- Track nomor 5
- Sisi nomor 1
- Sektor awal nomor 2
- Sektor akhir nomor 2

Pada saat perintah READ DATA diterima, pengontrol disket (FDC) memilih head 1 dan dia mulai membaca aliran bit data dari disket. Pengontrol disket mulai merakit bit-bit data menjadi byte (yaitu konversi serial ke paralel). Data pada sektor 2 harus ditransfer ke memori. FDC mengidentifikasi sektor 2 dari field ID sektor 2. Field ID tidak ditransfer ke memori. Segera setelah satu byte di baca, pengontrol membangkitkan

sinyal DMAREQ (permintaan DMA) yang ditransmisikan ke pengontrol DMA. Hal ini mengindikasikan satu byte harus ditransfer. Pengontrol DMA dengan segera membangkitkan sinyal HOLDREQ (permintaan HOLD) yang mencapai CPU. Hal ini mengindikasikan bahwa pengontrol DMA memerlukan kontrol bus untuk mentransfer byte data. Jika CPU tidak sedang menggunakan bus, maka dia memberikan persetujuan ke pengontrol DMA dengan membangkitkan sinyal HOLDACK (*hold acknowledgement*). Sekarang pengontrol DMA mengambil alih bus. Dengan segera pengontrol DMA memberikan DMAACK (*DMA acknowledgement*) yang menyetujui DMAREQ dari pengontrol disket. Pada saat respon, FDC menempatkan byte data pada bus. Pada saat yang sama, pengontrol DMA memberikan alamat awal memori dan sinyal MEMW (tulis memori) ke memori. Sekarang memori menyimpan data yang ada pada bus data ke lokasi yang ditetapkan oleh bus alamat. Jadi satu byte telah ditransfer dari pengontrol disket ke memori. Pada saat yang sama, pengontrol DMA melakukan dua operasi penjagaan (*house-keeping*) seperti yang diberikan berikut ini:

- Menaikkan alamat memori
- Menurunkan hitungan byte (byte count)

FDC membuang/menghilangkan sinyal DMAREQ. Pada saat respons, pengontrol DMA menghilangkan sinyal DMAACK serta sinyal HOLDREQ. Pada saat respons, CPU menarik persetujuan bus ke pengontrol DMA dengan menghilangkan sinyal HOLDACK.

Urutan DMA diulangi untuk setiap byte. Namun, pengontrol DMA tidak akan mentransfer parameter '*byte count*' lagi. Karena itu bila hitungan byte (*byte count*) mencapai nol, pengontrol DMA menginformasikan FDC tidak berusaha melakukan transfer data lagi. Untuk tujuan ini, pengontrol DMA memberikan sinyal LAST BYTE (TERMINAL COUNT) bersama sinyal DMAACK. Sekarang FDC mengetahui bahwa ini adalah byte terakhir yang ditransfer dan tidak ada lagi permintaan transfer data yang akan dibangkitkan oleh FDC.

Selama semua urutan DMA, software (dan CPU) tidak menyadari/mengetahui kemajuan yang dicapai oleh operasi I/O khusus ini. Ketika menerima sinyal LAST BYTE, FDC membangkitkan interupsi ke CPU. Sekarang CPU beralih ke software yang mulai operasi I/O. Jadi interupsi dari pengontrol perangkat adalah suatu informasi ke software bahwa operasi I/O telah selesai.

9.6 PORT PARALEL

Berbagai peripheral baru seperti pita magnetik, portable disk drive, adaptor LAN, CD ROM drive, printer sharer, dan sebagainya dapat dengan mudah diantarmuka ke PC via port paralel. Karena peripheral baru tersebut menawarkan kecepatan transfer tinggi dan transfer dua-arah, maka diperkenalkanlah standar IEEE 1284.

Standar IEEE 1284 menetapkan suatu antarmuka peripheral paralel dua-arah ke PC. Standar ini memungkinkan komunikasi dua-arah kecepatan tinggi antara PC dan peripheral eksternal. Dia menetapkan lima mode untuk transfer data keluar di mana di antaranya dua mode adalah komunikasi dua-arah, dan yang lainnya satu-arah (*uni-directional*). Pada salah satu mode, dia mendukung port paralel original tanpa ada perubahan. Pada Tabel 9.4 diberikan mode-mode tersebut.

TABEL 9.4 Mode-mode port paralel IEEE 1284

Mode	Datapath	Tipe Arah	Tanggung Jawab protokol	Keterangan
Mode Kompatibel (centronics/mode standar)	8-bit	Forward (PC ke peripheral)	CPU/BIOS (sama seperti SPP)	Mode ini utamanya untuk kompatibel ke bawah ke peripheral lama; bersama dengan mode nibble atau byte memungkinkan transfer dua arah.
Mode Nibble	4-bit	Reverse (peripheral ke PC)	CPU/BIOS	Empat saluran status dari peripheral digunakan mentransfer data dari peripheral; dua siklus

Mode	Datapath	Tipe Arah	Tanggung Jawab	Keterangan protokol
Mode Byte	8-bit	Reverse	CPU/BIOS	dibutuhkan untuk pembacaan satu byte data.
Mode Enhanced Parallel Port (EPP)	8-bit	Dua-arah	Hardware pengontrol	Mengubah datapath dalam SPP menjadi path dua-arah dengan penambahan sebuah ketentuan untuk men-disable interface driver bagi saluran data.
Mode Extended Capability Port (ECP)	8-bit	Dua-arah	Hardware pengontrol	Utamanya untuk peripheral non-printer seperti CD-ROM, tape drive, hard disk drive, jaringan, adapter, dll.

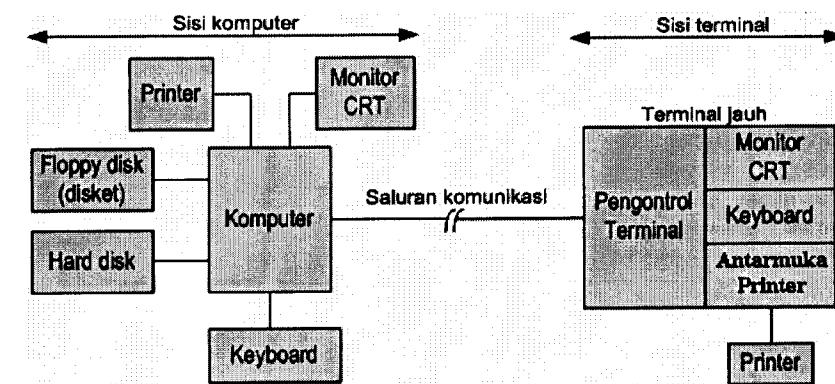
Pada tiga mode pertama (Compatible, Nibble, dan Byte), transfer data dilakukan berdasarkan programmed I/O mode/interrupt mode. I/O driver menangani protokol handshaking (serupa dengan SPP) dan memungkinkan kecepatan transfer data hingga 50–100 Kbyte per detik. Mode EPP dan ECP berdasarkan pada metode hardware transfer data. Software melakukan operasi-operasi minimum seperti inisiasi dan pengontrol printer menangani handshaking dan transfer data tanpa perlakuan software.

9.7 PORT SERIAL

Antarmuka serial digunakan untuk menghemat biaya untuk komunikasi jarak jauh. Standar RS 232C adalah antarmuka serial yang lazim. Kebutuhan untuk transfer data antara komputer dan terminal jauh (*remote terminal*) memunculkan pengembangan komunikasi serial. Komunikasi serial melakukan transfer data bit demi bit pada saluran komunikasi tunggal seperti yang ditunjukkan pada Gambar 9.42. Biaya perangkat keras komunikasi menjadi rendah karena hanya ada satu kawat/saluran tunggal yang diperlukan untuk bit-bit data tersebut.

Bila suatu sinyal digital ditransmisikan melalui saluran transmisi yang panjang, maka sinyal akan dipengaruhi dua hal yaitu:

1. Sinyal teredam yakni level tegangan turun/drop karena adanya resistansi yang terdistribusi.
2. Saluran transmisi mempunyai induktansi dan kapasitansi terdistribusi karena sinyal terdistorsi yaitu adanya tebing naik dan tebing turun yang lambat.



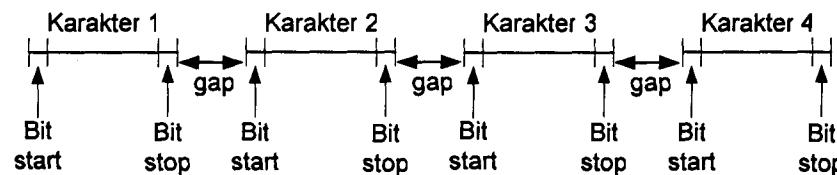
Gambar 9.42 Komunikasi data ke suatu terminal jauh

Redaman dan distorsi sinyal digital meningkat sejalan dengan panjang saluran. Karena itu, sinyal tidak dapat ditransmisikan secara efektif pada jarak yang jauh. Untuk mengatasi hal ini, digunakan sebuah MODEM (MODulator-DEModulator) dan peralatan komunikasi untuk transfer data

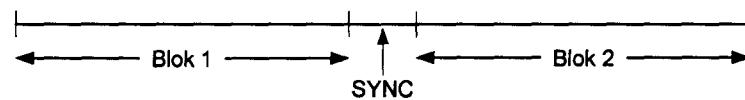
melalui saluran telepon. Sepasang MODEM diperlukan untuk melakukan hubungan komunikasi dua terminal.

9.7.1 Komunikasi Asinkron dan Sinkron

Komunikasi data serial umumnya menggunakan skema komunikasi asinkron atau sinkron. Dua metode ini menggunakan teknik yang berbeda untuk sirkuit sinkronisasi pada sisi pengirim dan penerima. Pada skema asinkron, setiap karakter termasuk bit start dan bit stop (Gambar 9.43). Pada skema sinkron, setelah sebuah byte data ditetapkan, maka sebuah pola bit yang disebut SYNC dikirimkan (Gambar 9.44). Pada komunikasi asinkron dapat dibuat jarak (gap) antara karakter yang berdekatan sedangkan pada komunikasi sinkron tidak ada gap. Pada komunikasi sinkron, aliran bit-bit data dengan kecepatan tetap yang tiba adalah kontinyu. Laju di mana bit-bit data dikirimkan disebut *baud rate* yang dinyatakan dalam BPS (*bit per second*). Pada skema komunikasi asinkron, bit-bit dalam sebuah frame karakter (termasuk bit start, bit parity, dan bit stop) dikirim pada baud rate.



Gambar 9.43 Komunikasi Asinkron



Gambar 9.44 Komunikasi sinkron

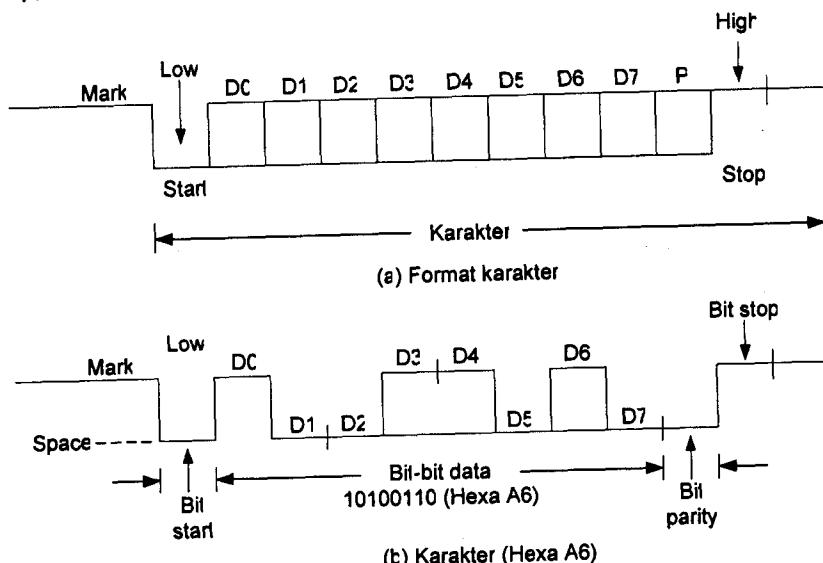
Komunikasi sinkron umumnya digunakan bila dua komputer berkomunikasi satu sama lain atau bila sebuah terminal buffer berkomunikasi ke komputer. Komunikasi asinkron digunakan bila peripheral kecepatan rendah berkomunikasi dengan komputer.

Format serial dari sebuah karakter dalam komunikasi asinkron ditunjukkan pada Gambar 9.45 dalam negative logic. Gambar ini juga menunjukkan format data serial untuk data A6. Bila tidak ada transmisi maka saluran tetap pada level tegangan high yang disebut MARK. Bit start pada saluran berada dalam level low yang disebut SPACE dan dia mengindikasikan bahwa satu karakter ditransmisikan untuk start. Durasi bit start sama dengan satu periode baud.

LSB mengikuti bit start dan kemudian berikutnya diikuti bit-bit lainnya. Durasi setiap bit data sama dengan satu periode baud. Saluran menjadi high atau low bergantung pada bit-bit data. Jumlah bit-bit data dalam suatu karakter dapat dari lima hingga delapan seperti yang diinginkan oleh ujung pengirim dan penerima.

Bit parity dikirim setelah bit signifikan (MSB) data. Bit parity bisa ganjil atau genap. Bit parity dapat juga dihilangkan. Durasi bit parity sama dengan satu periode baud.

Bit stop menjadikan saluran ke level high (mark) dan durasi bit stop bisa 1; 1,5 atau 2 periode baud. Jumlah bit stop dapat diprogram. Setelah bit-bit stop, saluran tetap dalam keadaan level high (mark).

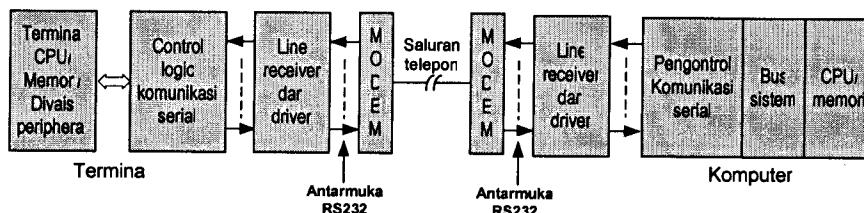


Gambar 9.45 Format karakter dalam komunikasi asinkron

Standar baud rate adalah: 50; 110; 134,5; 150; 300, 1200; 2400; 4800; 9600; 19200 dan 38400. Dalam suatu sistem komunikasi, baud rate dipilih bergantung pada kualitas saluran transmisi dan kemampuan peralatan ujung pengirim dan penerima. Pada sistem transmisi sederhana, baud rate adalah sama dengan Bit Per Second (BPS) yaitu kecepatan transfer data dan jumlah bit yang ditransmisikan per detik pada saluran adalah sama. Pada sistem komunikasi di mana level modulasi jamak digunakan, maka baud rate dan bit per second tidaklah sama, yaitu kecepatan transfer data tidak sama dengan jumlah bit yang ditransmisikan per detik pada saluran.

Antarmuka Serial

Komponen-komponen dasar suatu sistem komunikasi melibatkan sebuah komputer dan terminal seperti yang ditunjukkan pada Gambar 9.46. Terminal dapat diganti dengan komputer lain atau sebuah printer. Dengan menggunakan sebuah modem, sinyal digital diubah menjadi sinyal analog. Saluran telepon digunakan untuk mentrasmisikan sinyal analog. Pada ujung penerima, sinyal analog ini diubah kembali menjadi sinyal digital. Modem dapat menggunakan teknik modulasi yang beragam: Modulasi Amplituda (AM), Modulasi Frekuensi (FM), Modulasi Frequency Shift Keying (FSK), dan sebagainya. Metode FSK merupakan metode yang umum digunakan dalam modem.



Gambar 9.46 Komponen-komponen sistem komunikasi serial

Antarmuka RS-232

Antarmuka RS-232 adalah antarmuka standar yang dinyatakan oleh *Electronics Industries Association* (EIA) (RS singkatan dari *Recommended Standard*). Spesifikasi standar RS-232C sama dengan rekomendasi standar CCITT V-5.

Antarmuka RS-232 mengharapkan sebuah modem yang dikoneksikan pada ujung pengirim dan penerima. Modem disebut DCE (*Data Communication Equipment*) sedangkan komputer, terminal atau printer tempat di mana modem diantarmuka disebut DTE (*Data Terminal Equipment*). DCE dan DTE dihubungkan via kabel yang panjangnya harus tidak melebihi 50 feet. Walaupun kurang andal dia tidak memengaruhi komunikasi jika kecepatan transfer data dikurangi ketika jarak dinaikkan. DTE mempunyai konektor jantan (male) tipe-D 25 pin dan DCE mempunyai konektor betina (female) tipe-D 25 pin. Namun beberapa sistem menggunakan konektor 9 pin.

Level Sinyal RS-232

RS-232 mengikuti logika negatif. Logika 1 direpresentasikan dengan sebuah tegangan negatif dan logika 0 direpresentasikan dengan tegangan positif. Level 1 (high) bervariasi dari -3 sampai -15 Volt dan level 0 (low) bervariasi dari +3 sampai +15 Volt. Sirkuit hardware yang digunakan secara praktis untuk antarmuka RS-232 menetapkan level sinyal pada +12 Volt (logika 0) dan pada -12 Volt (logika 1).

Sinyal-Sinyal RS-232

Pada Tabel 9.5 ditunjukkan sinyal-sinyal RS-232. TXD membawa bit-bit data yang dikirim oleh DTE. Modem menerima sinyal TXD dan menggunakan untuk pemodulasi sinyal pembawa. RXD adalah data dari DCE ke DTE. RXD dibangkitkan oleh modem dengan melakukan demodulasi sinyal yang diterima dari ujung modem yang lain.

Sebelum pengiriman data DTE meminta izin pada modem melalui sinyal RTS. Bila modem mendapatkan jalur komunikasi (terdiri atas saluran telepon, ujung modem lain dan DTE) siap untuk melakukan komunikasi, maka dia memberikan sinyal CTS ke DTE sebagai suatu persetujuan untuk RTS.

DTE memberikan sinyal DTR bila dia dihidupkan, bebas-error dan siap. Modem memberikan sinyal DSR untuk mengindikasikan bahwa dia dihidupkan dan bebas-error.

Sinyal RI dan RLSD digunakan dengan modem dial. Bila saluran telepon adalah suatu saluran yang di-share, maka modem dial yang digunakan dan

telepon set tersambung ke modem. Bila suatu DTE pada satu ujung menginginkan berkomunikasi dengan DTE pada ujung yang lain maka dia melakukan dial. Modem pada sisi pengirim mengirimkan sebuah dial tone. Ketika respon, modem pemanggil mengirimkan sinyal RI ke DTE, dan mengirim suatu tone jawaban dalam 2 detik ke modem pemanggil. Kemudian modem pemanggil mengirimkan tone dengan durasi 8 ms pada saluran telepon. Sekarang modem pemanggil mengirimkan CD ke DTE. CD adalah suatu indikasi ke DTE bahwa dia akan segera menerima data yang dikirim oleh ujung DTE lain.

TABEL 9.5 Antarmuka RS-232

No	Sinyal	Nama Sinyal	Sumber	Tujuan
1	—	<i>Frame ground</i>	—	—
2	TXD	Kirim data	DTE	DCE
3	RXD	Terima data	DCE	DTE
4	RTS	<i>Request to send</i>	DTE	DCE
5	CTS	<i>Clear to send</i>	DCE	DTE
6	DSR	<i>Data set ready</i>	DCE	DTE
7	SG	<i>Signal ground</i>	—	—
8	RLSD atau CD	<i>Receive line signal detect</i>	DCE	DTE
9	DTR	<i>Data transmit ready</i>	DTE	DCE
10	RI	<i>Ring indicator</i>	DCE	DTE

9.8 ANTARMUKA I/O SERIAL MODERN

Dua antarmuka serial modern yaitu USB (universal serial bus) dan Firewire merupakan antarmuka serial yang mempunyai fitur menarik karena mempunyai kinerja tinggi dan juga karena fleksibilitasnya. Mereka juga mendukung perangkat peripheral yang luas mulai dari keyboard hingga kamera digital.

9.8.1 Universal Serial Bus

Universal Serial Bus (USB) lebih unggul dari port serial RS-232 dalam beberapa aspek: mudah diinstalasi, transfer rate yang lebih cepat, pengkabelannya sederhana dan koneksi multi perangkat. Fitur-fitur USB sebagai berikut:

1. Multi perangkat: Dapat dikoneksikan hingga 127 macam perangkat ke bus USB tunggal.
2. Transfer rate: USB standar yang pertama mendukung transfer rate 12 Mbps. USB 2.0 mendukung transfer rate yang lebih tinggi.
3. Mendukung berbagai peripheral: perangkat bandwidth rendah seperti keyboard, mouse, joystick, game pad, floppy disk drive, zip drive, printer, scanner, dan lain sebagainya.
4. Arsitektur Hub: setiap perangkat dikoneksikan ke hub USB. Hub USB adalah sebuah unit pintar yang berinteraksi dengan PC pada satu sisi dan perangkat-perangkat peripheral pada sisi lain.
5. *Hot pluggability*: Perangkat USB dapat dikoneksikan tanpa mematikan PC. Fitur "Plug and Play" dalam BIOS dan perangkat-perangkat USB menjaga/mengurus pendekripsi, mengenali dan menangani perangkat. Pemakai secara total dibebaskan dari prosedur konfigurasi.
6. Alokasi daya. Pengontrol USB pada PC mendekripsi keberadaan (*attachment*) atau ketidakberadaan (*detachment*) perangkat-perangkat USB dan mengalokasikan level-level daya listrik yang sesuai.
7. Mudah diinstalasi: Hanya terdapat satu kabel. Kabel dengan 4-pin membawa sinyal-sinyal seperti yang ditunjukkan pada Tabel 9.6.

TABEL 9.6 Antarmuka RS-232

Nomor pin	Sinyal
1	Power
2	Sinyal -
3	Sinyal +
4	Ground

8. Host centric: CPU/software menginisiasi setiap transaksi pada bus USB. Karena itu, overhead pada software meningkat bila jumlah peripheral banyak, melibatkan banyak transaksi yang terkoneksi.

9.8.2 Firewire (IEEE 1394)

Firewire pertama kali diperkenalkan oleh PC Apple dan kemudian distandarisasi dengan IEEE 1395. Firewire menjadi suatu pilihan yang menarik untuk berbagai kecepatan peripheral termasuk perangkat-perangkat multimedia seperti kamera video digital. Fitur-fitur Firewire sebagai berikut:

1. Hot pluggability: hal ini mirip dengan USB
2. Multi perangkat hingga 63
3. Koneksi kancing (snap): tidak memerlukan device ID, jumper, DIP switch terminator dan sebagainya
4. Power sourcing
5. Rekonfigurasi dinamik
6. Kecepatan lebih tinggi: 400 Mbps; bandwidth 30 kali lebih tinggi dari USB
7. Antarmuka peer-to-peer: Setiap perangkat pada bus Firewire membentuk sebuah node tidak seperti USB
8. Transfer data *isochronous*: Firewire mendukung transfer data isochronous yang cocok untuk video digital dan time-critical media lainnya. Transfer data isochronous menjamin pengiriman data antara node-node dengan tepat waktu. Bus secara otomatis mengalokasikan 10 MB/detik untuk overhead perintah serial dan selebihnya untuk perangkat. Ketika bandwidth bus Firewire terpakai semuanya (dialokasikan sepenuhnya), maka perangkat yang lainnya tidak dikenali lagi.
9. Mendukung DMA: Firewire mendukung transfer akses memori langsung (DMA), tidak seperti USB dan IDE. Cocok untuk perangkat-perangkat berikut:
 - Kamera digital
 - Scanner
 - Hard disk drive
 - Removable disk drive
 - Printer
 - Tape back up
 - Video tape
 - Video disk
 - Music system

9.9 SIKLUS BUS

Karakteristik kunci suatu bus adalah penggunaan media transmisi secara bersama dalam waktu yang tidak bersamaan. Perangkat yang mentransmisikan sebuah sinyal pada bus, dapat diterima oleh semua perangkat lain yang terkoneksi pada bus tersebut. Jika ada dua perangkat yang transmit dalam satu periode waktu yang sama, maka sinyal-sinyal tersebut akan saling tumpang tindih dan menjadi rusak. Jadi, pada satu waktu hanya ada satu perangkat yang dapat ditransmisikan. Hanya ada satu perangkat yang dapat menjadi master bus (pengontrol bus), dan perangkat lainnya menjadi slave. Master bertugas mengontrol bus, dan dapat mengirim atau menerima.

Siklus bus adalah urutan kejadian pada bus dalam mentransfer satu word informasi antara CPU dan memori atau port I/O. Urutan operasi dimonitor oleh CPU (master). Bus kontrol digunakan untuk mengontrol akses saluran data dan saluran alamat, karena saluran data dan saluran alamat pemakaiannya di-share oleh semua komponen. Sinyal-sinyal kontrol mentransmisikan informasi pewaktuan/penjadwalan serta perintah di antara modul-modul sistem. Sinyal-sinyal pewaktuan mengindikasikan validitas data dan informasi alamat. Pada Tabel 9.7 diberikan beberapa macam siklus bus khas pada saluran kontrol.

TABEL 9.7 Beberapa siklus bus

Kontrol	Penjelasan
▪ <i>Memory write</i>	menyebabkan data pada bus ditulis ke lokasi memori yang teralamat
▪ <i>Memory read</i>	menyebabkan data dari lokasi memori yang teralamat ditempatkan pada bus
▪ <i>I/O write</i>	menyebabkan data pada bus diberikan ke I/O port yang teralamat
▪ <i>I/O read</i>	menyebabkan data dari I/O port yang teralamat ditempatkan pada bus
▪ <i>Transfer ACK</i>	mengindikasikan bahwa data telah diterima dari atau ditempatkan pada bus
▪ <i>Bus request</i>	mengindikasikan bahwa suatu modul memerlukan perolehan kontrol bus
▪ <i>Bus grant</i>	mengindikasikan bahwa suatu permintaan modul

Kontrol	Penjelasan
	telah diberikan kontrol bus
▪ <i>Interrupt request</i>	mengindikasikan bahwa suatu interupsi ditunda.
▪ <i>Interrupt ACK</i>	pengakuan bahwa interupsi yang tertunda telah dikenali
▪ <i>Clock</i>	digunakan untuk operasi-operasi sinkronisasi
▪ <i>Reset</i>	menginisialisasi semua modul

9.10 ORGANISASI BUS

Pengontrol I/O dapat dikoneksikan secara langsung ke bus sistem. Solusi efisien untuk maksud tersebut adalah menggunakan satu atau lebih bus ekspansi. Antarmuka bus ekspansi menyangga transfer data antara bus sistem dan pengontrol-pengontrol I/O pada bus ekspansi. Susunan ini membolehkan sistem untuk mendukung berbagai perangkat I/O secara luas dan pada saat yang sama mengisolasi trafik antara memori ke prosesor dari trafik I/O.

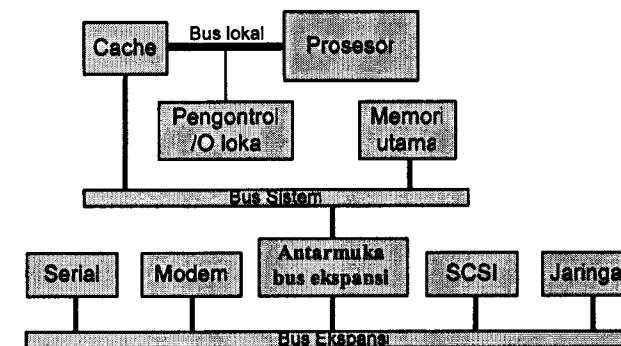
Gambar 9.47 menunjukkan beberapa contoh khas perangkat-perangkat I/O yang dapat dicantolkan ke bus ekspansi. Koneksi-koneksi jaringan termasuk LAN (*local area network*) misalnya Ethernet 10-Mbps dan koneksi ke WAN (*wide area network*) misalnya packet-switching network. SCSI (*small computer system interface*) merupakan suatu tipe bus yang digunakan untuk mendukung local disk drive dan peripheral lainnya. Sebuah port serial dapat digunakan untuk mendukung printer atau scanner.

Arsitektur bus tradisional ini layak dari segi efisiensi tetapi mulai tidak mampu dalam mendukung perangkat-perangkat I/O dengan kinerja yang lebih tinggi. Untuk merespons pertumbuhan tuntutan ini, maka industri mulai membangun bus berkecepatan tinggi yang dapat diintegrasikan dengan sistem yang ada yang hanya memerlukan suatu jembatan antara bus prosesor dengan bus kecepatan tinggi tersebut. Susunan ini biasa disebut arsitektur loteng tengah (*mezzanine bus*).

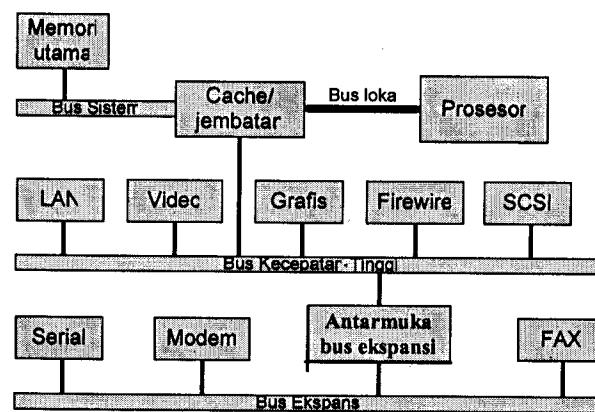
Gambar 9.48 menunjukkan realisasi khas dari pendekatan mezzanine bus tersebut. Sekali lagi, terdapat suatu bus lokal yang mengkoneksikan prosesor ke suatu pengontrol cache, yang pada akhirnya dikoneksikan ke bus sistem yang mendukung memori utama. Pengontrol cache diintegrasikan ke suatu jembatan (*bridge*), atau perangkat penyangga yang

mengoneksikannya ke bus kecepatan tinggi. Bus ini mendukung koneksi dengan LAN kecepatan tinggi seperti Fast Ethernet 100 Mbps, video dan pengontrol-pengontrol workstation grafis, serta pengontrol-pengontrol antarmuka ke bus-bus peripheral lokal termasuk SCSI dan Firewire. Perangkat-perangkat kecepatan rendah tetap didukung oleh suatu bus ekspansi dengan suatu antarmuka yang menyangga trafik antara bus ekspansi dan bus kecepatan-tinggi.

Keuntungan susunan ini adalah bahwa bus kecepatan-tinggi membawa perangkat-perangkat tuntutan-tinggi terintegrasi lebih dekat dengan prosesor dan pada saat yang sama tidak bergantung pada prosesor. Jadi, perbedaan pada prosesor dan kecepatan-kecepatan bus kecepatan-tinggi dan penentuan-penentuan saluran sinyal dapat ditoleransi. Perubahan dalam arsitektur prosesor tidak memengaruhi bus kecepatan-tinggi dan sebaliknya.



Gambar 9.47 Konfigurasi arsitektur bus tradisional



Gambar 9.48 Konfigurasi arsitektur bus kecepatan-tinggi

9.11 TEKNIK-TEKNIK ARBITRASI BUS

Arbitrasi bus adalah proses penentuan master bus yang mengontrol bus pada suatu waktu yang diberikan ketika ada suatu permintaan dari satu atau lebih master bus. Suatu master bus dapat diinisiasi dan melakukan suatu siklus bus untuk transfer data. Sebuah slave tidak dapat menjadi master bus dan karena itu tidak dapat melakukan inisiasi sebuah siklus bus. Pada kebanyakan komputer, prosesor dan pengontrol DMA adalah merupakan master bus sedangkan memori dan pengontrol I/O merupakan slave. Pada beberapa sistem, pengontrol I/O intelijen tertentu juga merupakan master bus. Pada beberapa sistem, prosesor jamak melakukan sharing pada sebuah bus.

Bila lebih dari satu master bus secara simultan memerlukan bus, maka hanya salah satu dari mereka yang mengontrol bus dan menjadi master bus aktif, yang lainnya harus menunggu gilirannya. Pengawas bus (*bus arbiter*) memutuskan yang mana yang akan menjadi *master bus* sekarang. Ada dua jenis pengawas bus:

1. Arbitrasi bus terpusat, di mana suatu pengawas didedikasikan mempunyai fungsi sebagai pengawas bus.
2. Arbitrasi bus terdistribusi, di mana semua master bus bekerja sama melakukan pengawasan. Dalam hal ini setiap master bus mempunyai sebuah seksi pengawasan.

9.11.1 Metode Arbitrasi Bus

Ada tiga skema populer untuk arbitrasi bus yang digunakan dalam sistem komputer:

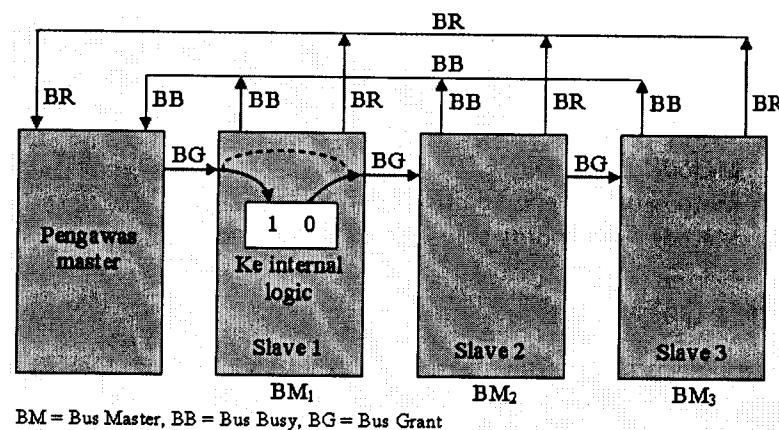
1. Metode *daisy-chain*
2. Metode *polling* atau *rotating priority*
3. Metode *fixed priority* atau *independent request*

Metode Daisy-Chain

'Bus Request' (BR) memberikan sinyal pada semua master yang membentuk satu line bus, BR. Dengan cara yang sama 'Bus Busy' (BB) mengeluarkan semua master membentuk satu line bus lainnya, BB. Bila tidak ada master bus yang menggunakan bus, maka BB tidak aktif. Kapanpun satu atau lebih permintaan bus terjadi, maka BR menjadi aktif. Pengawas (*arbiter*) melakukan inisiasi arbitrasi dengan mentransmisikan sinyal BG (bus grant). Sinyal ini mencari master pertama. Jadi sekarang ada dua kemungkinan:

1. Master pertama tidak memerlukan bus dan tidak membangkitkan BR. Dia hanya melewatkkan sinyal BG ke master bus berikutnya.
2. Master pertama memerlukan penggunaan bus sekarang dan membangkitkan sinyal BR. Dia menjadi master bus sekarang dengan memberikan sinyal BB dan membuang sinyal BR. Namun, BR dapat tetap aktif jika beberapa master lainnya juga telah membangkitkan BR. Master bus sekarang mempertahankan bus hingga dia menyelesaikan operasi busnya. Dengan melihat BB aktif, semua master bus lainnya mengetahui bahwa beberapa master bus telah mengambil alih bus dan karena itu tetap menunggu. Segera master bus yang sekarang menyelesaikan penggunaan busnya. Dia membuang sinyal BB. Jika BR tetap aktif, arbiter mengirimkan lagi sinyal BG.

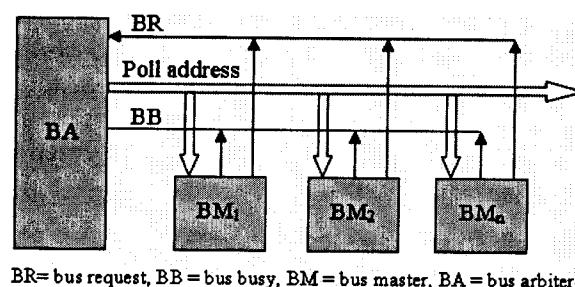
Gambar 9.49 mengilustrasikan metode daisy-chain.



Gambar 9.49 Arbitrasi bus daisy chain

Metode Polling

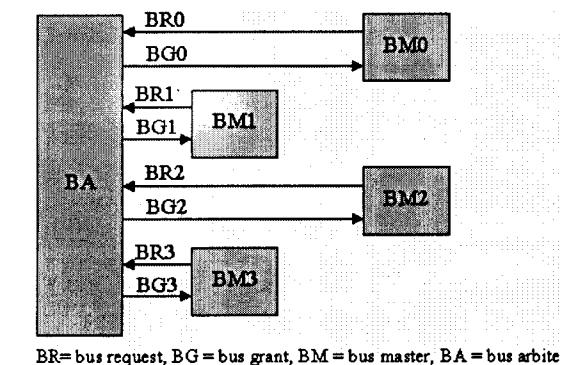
Pada metode *polling* (Gambar 9.50), sinyal BR dan BB sama dengan metode daisy chain. Pertama, BR aktif, arbiter melakukan polling ke master-master satu per satu, mentransmisikan alamat master pada saluran alamat. Master yang mempunyai alamat yang sesuai dengan alamat yang diberikan oleh arbiter menjadi master bus sekarang dengan membangkitkan sinyal BB. Metode ini memerlukan saluran lebih karena alamat harus ditransmisikan. Tetapi urutan pengalamanan master (urutan *polling*) dapat diprogram.



Gambar 9.50 Arbitrasi bus metode polling

Metode Arbitrasi Independent Request

Gambar 9.51 menunjukkan metode *independent request* atau *fixed priority*. Setiap master mempunyai saluran keluaran BR dan saluran masukan BG yang didedikasikan khusus. Jika ada n master, arbiter mempunyai n masukan BR dan n keluaran BG. Arbiter mengikuti suatu urutan prioritas dengan level prioritas berbeda untuk setiap master. Pada satu waktu yang diberikan, arbiter memberikan bus grant ke master prioritas tertinggi di antara master-master yang telah membangkitkan bus request. Skema ini memerlukan hardware yang lebih, tetapi membangkitkan respons yang cepat. Urutan prioritas juga dapat diprogram.



Gambar 9.51 Arbitrasi bus metode independent request atau fixed priority

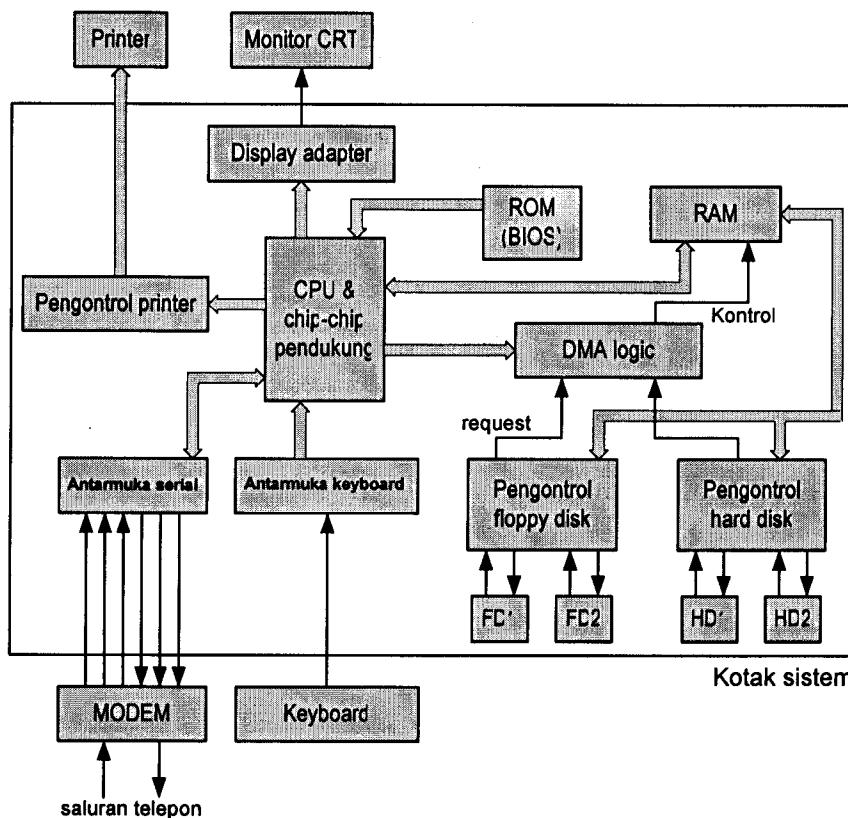
9.12 JENIS BUS KOMPUTER

Komunikasi dalam sistem komputer modern dilakukan dengan berbagai jenis bus, yaitu:

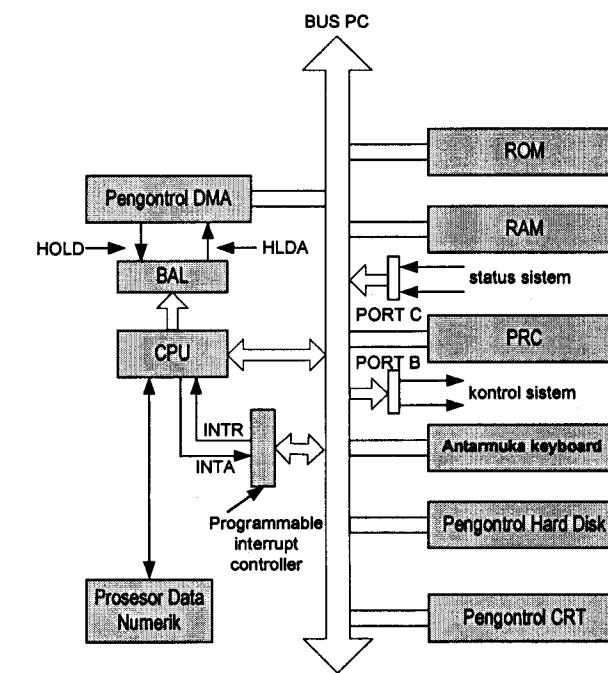
1. Bus Lokal
2. Bus Sistem
3. Bus I/O
4. Bus Loteng Tengah (*mezzanine bus*)

9.12.1 Konsep Bus Tunggal

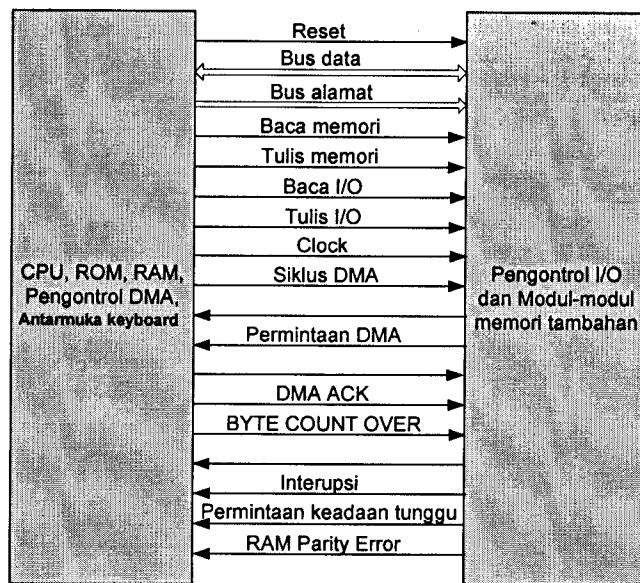
Gambar 9.52 menunjukkan diagram blok fungsional PC dan Gambar 9.53 mengilustrasikan organisasi bus untuk PC. Gambar 9.54 memamerkan sinyal-sinyal pada bus PC. Komputer IBM PC pertama beroperasi pada 4,77 MHz. dengan nol keadaan tunggu untuk memori motherboard dan satu keadaan tunggu untuk port-port I/O motherboard. Waktu akses RAM adalah 150 ns. Jumlah keadaan tunggu yang dipersyaratkan bergantung pada waktu akses dan kecepatan clock.



Gambar 9.52 Diagram blok fungsional komputer PC

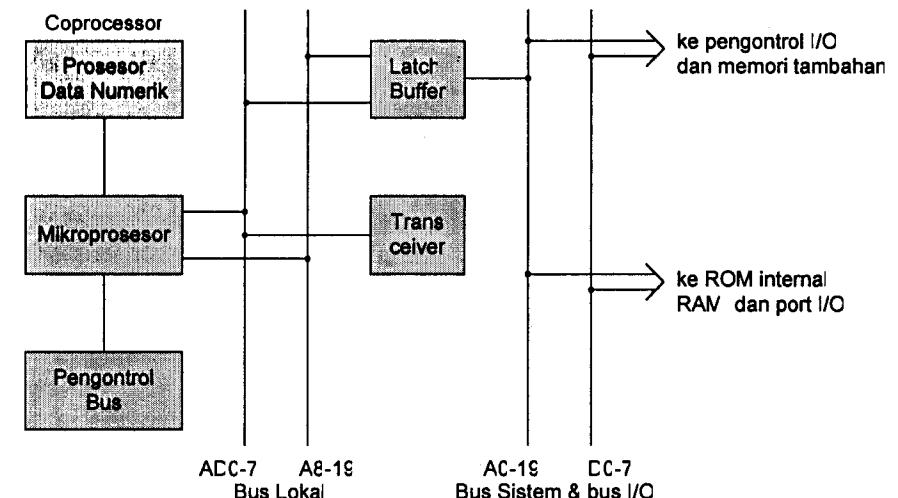


Pada PC-8088, bus sistem dan bus I/O adalah sama. Gambar 9.55 menunjukkan organisasi bus pada motherboard PC original. Koprosesor floating point dikoneksikan ke bus lokal 8088. Semua subsistem lainnya dihubungkan ke mikroprosesor melalui buffer. Subsistem-subsistem *systemboard* (chip RAM, ROM, dan I/O) dikoneksikan ke bus X (XA, XD, XC). *Daughterboard* dikoneksikan via slot-slot I/O dengan bus A, D dan C.

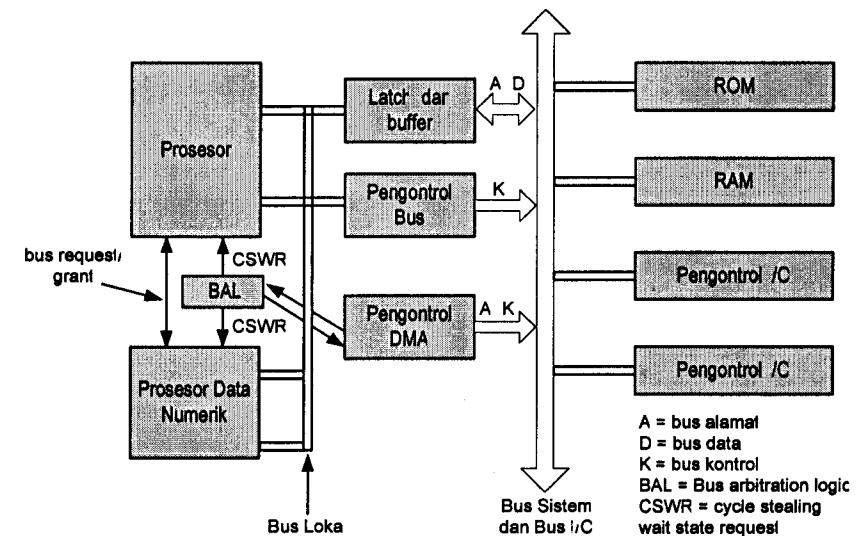


Gambar 9.54 Sinyal-sinyal bus PC

Jadi bus X dan bus D digunakan sebagai bus sistem dan bus I/O tanpa ada perbedaan. Pada pendekatan ini, CPU berkomunikasi dengan subsistem I/O dan memori dengan kecepatan yang sama. Secara konseptual, hanya ada satu bus yang ditampilkan pada Gambar 9.56. Pemisahan bus X dan bus D adalah untuk pendistribusian beban elektrik dan interkoneksi modul fisik antara motherboard (*systemboard*) dan expansion board. Pikirkan suatu keadaan ketika CPU membaca dari memori utama dan pada saat yang sama pengontrol DMA melakukan pembacaan memori utama. Jelas pengontrol DMA diprioritaskan untuk menghindari kemacetan data. Penambahan memori cache membantu mengurangi prosesor melakukan pengaksesan memori utama, tetapi memori cache juga menggunakan bus yang sama. Masalah ini diselesaikan dengan memisahkan bus memori dan bus I/O yang akan dijelaskan pada bagian berikut.



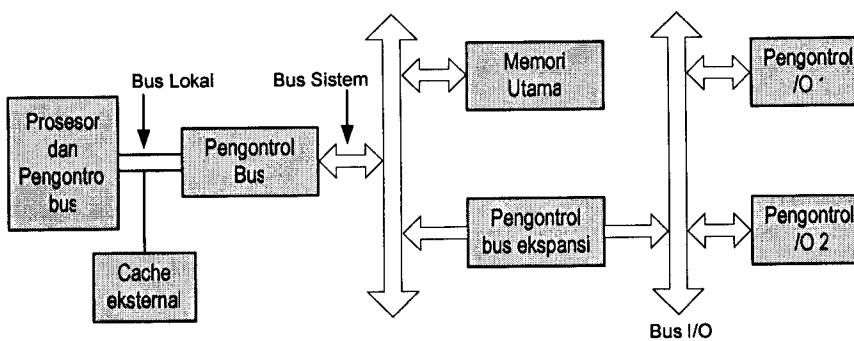
Gambar 9.55 Struktur bus tunggal pada PC-8088



Gambar 9.56 Konsep bus tunggal pada PC-8088

9.12.2 Bus I/O dan Isolated Memory

Konsep bus tunggal lebih sederhana dalam desain dan implementasi, tetapi kekurangannya bandwidth-nya menjadi terbatas karena peripheral-peripheral lambat. Hal ini memengaruhi komunikasi CPU – memori. Hal ini tidak terlalu serius dirasakan jika kecepatan prosesor lebih lambat. Ketika kecepatan prosesor tinggi yang tersedia, maka desainer sistem mendesain bus memori terpisah dengan bus I/O (Gambar 9.57). Bandwidth bus I/O lebih kecil dari bus memori. Karena itu komunikasi CPU – memori lebih cepat daripada komunikasi CPU – I/O dan komunikasi memori – I/O. Dan juga jika digunakan memori cache eksternal, membantu peningkatan kinerja CPU.



Gambar 9.57 Bus sistem dan bus I/O terpisah

Bus Lokal

Diperlukan suatu bus untuk mendukung peripheral kecepatan tinggi seperti HDD cepat, pengontrol display tingkat lanjut, prosesor-prosesor grafik dan sebagainya. Suatu solusi yang telah lama dikenal adalah koneksi pengontrol-pengontrol I/O pada bus lokal agar pengontrol-pengontrol ini dapat berkomunikasi dengan memori utama secara langsung tanpa melalui bus I/O ekspansi. Umumnya pengontrol CRT ditempatkan pada slot bus lokal. Jalan pintas ini memberikan transfer data tinggi untuk pengontrol video dan karena itu kinerja video menjadi lebih baik. Namun, adaptor bus lokal dapat memengaruhi trafik CPU ke memori. Karena adaptor bus lokal ditempatkan secara langsung pada ujung mikroprosesor, maka mereka

berkomunikasi dengan memori pada kecepatan yang lebih tinggi. Setiap adaptor harus didesain dengan cermat agar tidak membebani sinyal-sinyal bus lokal secara berlebihan. Jika terjadi pelanggaran maka akan memengaruhi fungsi sistem. Terlalu banyak adaptor tidak direkomendasikan. Bus lokal standar yang populer yang digunakan secara luas pada sistem berbasis prosesor 80486 adalah bus VESA (*Video Electronics Standard Association*).

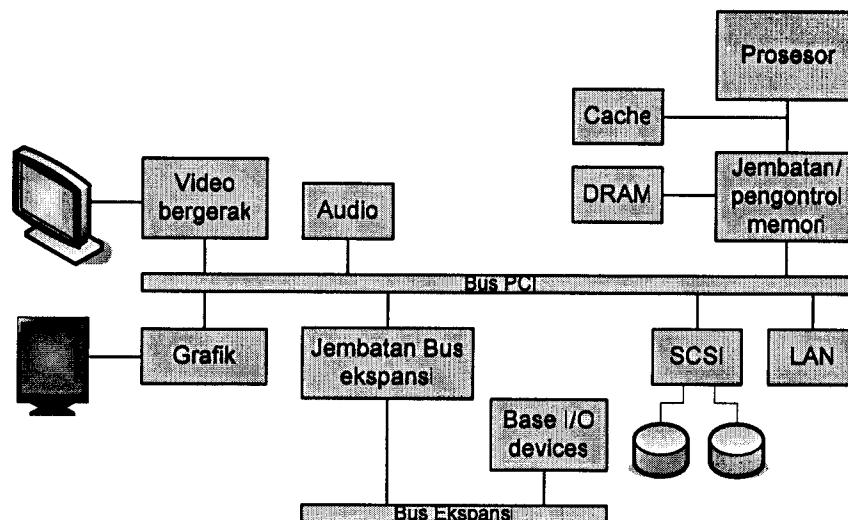
Bus PCI

Konsep bus lokal telah digantikan dengan bus PCI (*Peripheral Component Interconnect*) Intel. Bus PCI bukan merupakan bus I/O nyata. Dia merupakan suatu bus mezzanine yang bertindak sebagai bus pertengahan antara bus sistem dan bus I/O. Pengontrol PCI (*bridge-jembatan*) menghubungkan bus sistem dengan bus PCI. Bus PCI mengisolasi subsistem I/O kecepatan rendah dari prosesor tetapi pada saat yang sama pengontrol I/O dibawa mendekati memori. Bus PCI adalah bus dengan kecepatan tinggi. Pengontrol I/O pada bus PCI mempunyai keuntungan lain yaitu mereka dapat digunakan dengan suatu sistem tanpa mengganggu penggunaan prosesor eksak karena mereka tidak diantarmuka ke prosesor. Mereka hanya melihat bus PCI.

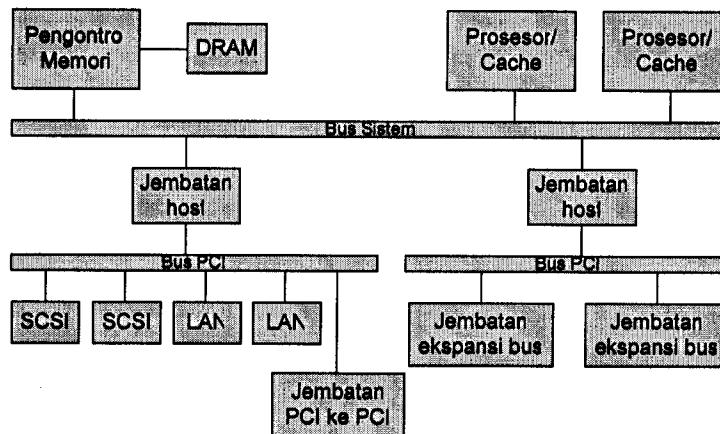
Bus PCI yang pertama melakukan transfer 32 bit setiap siklus dan beroperasi pada kecepatan 33 MHz untuk total bandwidth 133 MB/detik. Untuk menggalakkan pemakaiannya, Intel mematenkan bus PCI tersebut dan kemudian menetapkan semua paten tersebut untuk digunakan oleh publik, sehingga perusahaan apa saja dapat membuat peripheral-peripheral untuk bus ini tanpa membayar royalti.

Pada Gambar 9.58 ditunjukkan penggunaan bus PCI tipikal dalam suatu sistem prosesor-tunggal. Gabungan pengontrol DRAM dan jembatan ke bus PCI menyediakan koneksi dengan prosesor dan kemampuan untuk mengirimkan data pada kecepatan tinggi. Jembatan berfungsi sebagai penyanga data atau buffer agar kecepatan bus PCI dapat berbeda dengan kemampuan yang dimiliki I/O prosesor. Sedangkan pada Gambar 9.59 menunjukkan suatu sistem multiprosesor di mana satu atau lebih konfigurasi PCI dapat dikoneksikan oleh jembatan ke bus sistem prosesor. Bus sistem hanya mendukung unit-unit prosesor/cache, memori utama, dan jembatan-jembatan PCI. Penggunaan jembatan-jembatan sekali lagi adalah

untuk menjaga PCI agar tidak terikat dengan kecepatan prosesor yang juga menyediakan kemampuan untuk menerima dan mengirim data dengan cepat.



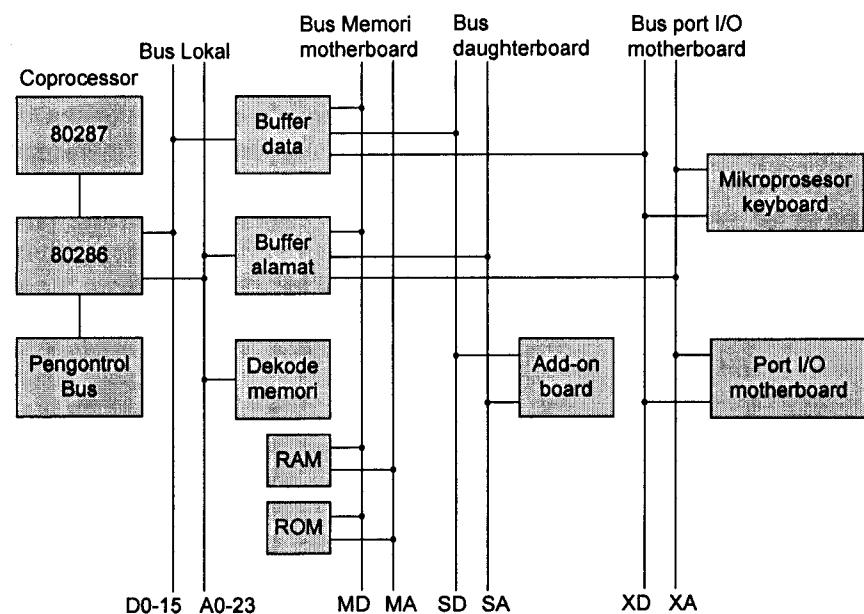
Gambar 9.58 Konfigurasi bus PCI pada sistem PC-desktop khas



Gambar 9.59 Konfigurasi bus PCI pada sistem server khas

Suatu fitur menarik dari desain IBM untuk PC dan PC/AT adalah desain ‘terbuka’ untuk spesifikasi bus PC (ISA bus) agar pabrik-pabrik lainnya juga dapat menggunakan desain bus yang sama tanpa mengeluarkan biaya royalti ke IBM. ISA (*Industry Standard Architecture*) adalah bus dengan lebar 16 bit dan beroperasi maksimum pada 8 MHz. IBM mengubah strateginya untuk sistem PS/2 dengan bus MCA yang disediakan hanya untuk pembayaran yang signifikan. Tetapi beberapa pabrik PC secara bersamaan membuat spesifikasi bus EISA untuk sistem 386 sebagai rival bus MCA. Bus MCA dan EISA merupakan intelligent bus dengan bandwidth yang besar.

Ketika sistem 486 datang, suatu bus lokal yang disebut bus VESA (bus VL) diperkenalkan untuk mendukung transfer kecepatan tinggi ke adaptor display, segera setelah Intel memperkenalkan bus PCI menjadi standar sistem Pentium. Dengan Pentium pro, FSB telah menjadi bus sistem karena arsitektur dual bus Pentium pro (dan diikuti berikutnya pada keluarga Pentium). Pentium 4 secara dramatis telah mengembangkan sistem bus dengan kecepatan hingga 400 MHz dan yang lebih tinggi lagi.



Gambar 9.60 Organisasi bus PC-AT/286

Pada Gambar 9.60 ditunjukkan organisasi bus PC-AT/286. Bus alamat adalah dua-arah pada bus ISA. Daughterboard adalah sebuah intelligent board dan dapat juga menjadi master bus. Beberapa sinyal kontrol khusus pada bus ISA memberitahukan motherboard apakah transfer bus sekarang adalah sebuah transfer 8-bit atau transfer 16-bit. Di dalam respons, prosesor melakukan satu siklus bus 16-bit atau dua siklus bus 8-bit. Sinyal MASTER digunakan oleh daughterboard untuk mengontrol sistem secara temporer. Untuk tujuan ini, dia membangkitkan sinyal DRQ dan setelah menerima pengakuan dari pengontrol DMA, maka dia mengaktifkan sinyal MASTER. Sekarang dia dapat mengakses memori atau I/O secara langsung di dalam sistem. Fitur ini dapat digunakan oleh suatu master bus untuk mengontrol bus.

Suatu master bus ISA adalah non-preemptible. Hingga dia melepaskan bus (dengan deaktivasi sinyal DRQ dan MASTER), master bus ISA melanjutkan menahan bus. Pada saat yang sama, dia bertanggungjawab pada master bus ISA ini untuk menjaga penyegaran DRAM juga bila dia menggunakan bus.

Bus EISA (*Extended Industry Architecture*) pertama kali distandardkan untuk sistem berbasis prosesor 80386. Seperti bus MCA, bus EISA juga bus yang pintar. Pada bus EISA, semua saluran alamat dibawa ke slot-slot I/O dalam keadaan tidak di-latch. Karena itu, suatu unit dapat merespons dengan cepat (ketika dialamat) dalam suatu siklus bus dan mengindikasikan tipe transfer, yaitu 32-bit atau 16-bit. Pada bus EISA, saluran interupsi dapat di-share dengan banyak kartu EISA. Setiap kartu EISA mempunyai sebuah kode identifier produk yang unik yang dapat dibaca oleh software. Board ekspansi EISA mempunyai dua mode operasi:

1. Mode master
2. Mode slave

Pada mode master, master bus EISA dalam board ekspansi EISA melakukan siklus-siklus bus untuk mengakses memori sistem atau port I/O. Pada mode slave, port I/O atau memori dalam board ekspansi EISA diakses oleh board sistem EISA; dalam keadaan ini, CPU dalam board sistem atau suatu master bus EISA dalam beberapa board ekspansi merupakan master bus. Board ekspansi EISA menggunakan sinyal MREQ (*EISA bus master request*) dan MAK (*EISA bus master acknowledgement*) untuk protokol master bus. Tidak seperti pada ISA, EISA tidak dapat menggunakan sinyal-

sinyal DMA (DRQ/DACK) untuk permintaan bus dan persetujuan. Sebagai gantinya dibangun komunikasi langsung dengan EISA BAL. Karena master bus EISA terhubung ke bus lokal, maka dia dapat melakukan *burst bus cycles*.

RINGKASAN

Antarmuka antara perangkat peripheral ke inti sistem dilakukan menggunakan pengontrol perangkat. Pengontrol perangkat dihubungkan ke CPU atau memori melalui bus dan ke antarmuka perangkat di sisi lain. Untuk suatu sistem yang ada, antarmuka sistem untuk semua Pengontrol I/O adalah sama, sedangkan antarmuka perangkat bergantung pada jenis perangkat. Beberapa contoh antarmuka perangkat antara lain: RS-232C, antarmuka centronics, SA450, ST-506, SCSI, IDE, USB, Firewire. Ada dua jenis antarmuka perangkat: antarmuka serial dan antarmuka paralel. Pada antarmuka serial, hanya ada satu saluran data dan bit-bit data dalam suatu byte ditransmisikan satu per satu secara serial. Antarmuka paralel menyediakan delapan saluran data secara paralel supaya delapan bit data (byte) ditransmisikan dari sistem secara serempak/simultan.

I/O driver merupakan program yang melakukan berbagai operasi I/O dengan memberikan serangkaian perintah yang sesuai ke pengontrol peripheral. Pengontrol membangkitkan sinyal-sinyal yang tepat bergantung pada perintah yang diterima dari driver CPU/perangkat. Setiap sinyal status melaporkan suatu status khusus dari perangkat.

Ada empat metode yang populer perancangan hardware pengontrol I/O:

- a) Desain hardwired (random logic)
- b) Desain berbasis LSI fungsi khusus
- c) Desain berbasis mikroprosesor
- d) Custom built IC atau gate array.

Port I/O adalah sebuah unit hardware yang dapat dialami program di mana CPU dapat melakukan transfer informasi. Setiap port mempunyai alamat sendiri-sendiri yang digunakan ketika berkomunikasi satu sama lain. Sebuah port bisa berupa unit hardware yang berdiri sendiri (*independen*) atau berupa bagian dari suatu unit hardware. Pada skema direct I/O, port I/O diperlakukan berbeda dengan lokasi-lokasi memori dan program menggunakan instruksi IN atau OUT untuk berkomunikasi. Program membaca data dari suatu port input dengan menggunakan instruksi IN. Instruksi OUT melakukan transfer data dari register prosesor ke port output. Pada skema memory mapped I/O, port-port I/O diperlakukan sebagai lokasi memori. Karena itu tidak digunakan instruksi IN dan OUT.

Bus adalah sekumpulan jalur yang berisi sejumlah sinyal untuk pengoperasian unit-unit hardware. Sinyal-sinyal ini ada tiga jenis yaitu:

- (a) Sinyal-sinyal bus alamat
- (b) Sinyal-sinyal bus data
- (c) Sinyal-sinyal bus kontrol

Siklus bus adalah suatu rangkaian kejadian untuk mentransfer satu word informasi. Bila dua unit berkomunikasi satu sama lain, maka salah satunya bertindak sebagai master dan yang lainnya sebagai slave. Pada metode transfer sinkron, unit-unit pengirim dan penerima mensuplai clock yang sama. Setiap aksi disinkronisasikan terhadap tebing naik atau tebing turun sinyal clock. Pada transfer asinkron, tidak ada clock bersama dan master serta slave dapat mengikuti urutan protokol persetujuan/pengakuan (*acknowledgment protocol*) selama rentetan transfer data. Interupsi adalah sebuah sinyal atau kejadian dalam sistem komputer, di mana komputer melakukan penundaan pekerjaan yang ada sementara dan mengeksekusi program lain yang dikenal dengan rutin layanan interupsi (ISR, *interrupt service routine*). Penanganan interupsi dilakukan melalui kerja sama antara hardware dan software. Ada tiga aksi yang secara otomatis dikerjakan oleh CPU ketika merasakan adanya interupsi:

- (a) Menyimpan status CPU
- (b) Menemukan penyebab interupsi
- (c) Bercabang ke ISR

CPU dapat melakukan penghalangan interupsi dengan menggunakan IE flag. Selama sebuah rutin layanan interupsi dieksekusi, CPU dapat mengizinkan interupsi yang lain, hal ini disebut interupsi sarang. Pengontrol interupsi menetapkan sebuah prioritas tetap untuk sejumlah permintaan interupsi. Jika program bermaksud untuk hanya mengizinkan interupsi tertentu dan menghalangi yang lainnya maka dia dapat melakukan penghalangan selektif. Pola penghalang diberikan oleh program yang memutuskan interupsi mana yang dihalangi pada satu waktu. *Non-Maskable Interrupt* (NMI) dibangkitkan untuk perhatian penting dari CPU (ISR). Prosesor melakukan NMI dengan segera. Pengindraan suatu NMI tidak bergantung pada IE flag.

Istilah "transfer data" merujuk pada pertukaran informasi (program dan data) antara CPU/memori dan perangkat I/O. Transfer data umumnya

dilakukan dalam langkah-langkah satu word pada satu waktu. Metode software transfer data ada dua jenis yaitu:

- (a) Programmed mode (polling)
- (b) Interrupt mode

Programmed mode transfer data adalah sebuah tugas yang dikhususkan dilakukan pada suatu stretch (dari awal byte pertama sampai akhir byte terakhir) oleh software. Pada interrupt mode, software (CPU) diminta oleh hardware hanya ketika transfer data aktual dari setiap byte dan pada saat yang lain dia bebas melakukan tugas lain. DMA mode adalah sebuah metode hardware. Pengontrol DMA atau data channel menjaga transfer data antara memori dan perangkat I/O. Software menyampaikan informasi relevan tentang operasi I/O ke pengontrol DMA. Dia mempunyai dua keuntungan terhadap programmed mode dan interrupt mode yaitu:

- (a) Mendukung perangkat peripheral kecepatan tinggi
- (b) Pencapaian paralelisme antara pemrosesan CPU dan operasi I/O

Kinerja dan fleksibilitas tinggi merupakan fitur yang menarik dari antarmuka serial modern seperti Universal Serial Bus (USB) dan Firewire. Mereka juga mendukung perangkat peripheral yang luas—dari keyboard sampai kamera digital.

Arbitrasi bus adalah suatu proses penentuan bus master (siapa yang akan mempunyai kontrol terhadap bus) bila ada sebuah permintaan untuk bus mastership dari sejumlah bus master. "Bus arbiter" memutuskan siapa yang akan menjadi bus master. Sementara yang lainnya menunggu gilirannya. Pada kebanyakan komputer, prosesor dan pengontrol DMA adalah bus master, sedangkan memori dan pengontrol I/O merupakan slave. Pada beberapa sistem, pengontrol I/O pintar juga merupakan bus master. Komunikasi pada suatu sistem komputer modern terdiri atas jenis-jenis bus: Local Bus, System Bus, I/O Bus, dan Mezzanine Bus.

SOAL-SOAL ULANGAN

1. Setiap komputer didukung oleh sejumlah perangkat peripheral. Untuk menggunakan perangkat peripheral, dibutuhkan dua modul yaitu (1) pengontrol I/O dan (2) _____.

2. Fungsi _____ adalah melakukan antarmuka perangkat peripheral ke inti sistem (CPU/memori).
3. Fungsi _____ adalah menyampaikan berbagai perintah ke pengontrol I/O untuk melakukan sejumlah operasi I/O.
4. Istilah _____ diartikan sebagai sinyal-sinyal yang terdapat antara dua subsistem dan protokol komunikasi antara subsistem.
5. Ada dua jenis antarmuka perangkat yaitu: (1) antarmuka serial dan (2) _____.
Pada antarmuka serial hanya ada satu saluran data dan bit-bit data dalam suatu byte ditransmisikan _____.
Setiap sinyal kontrol menentukan informasi/tindakan kontrol yang khusus pada perangkat. Pada dasarnya ada tiga jenis sinyal kontrol yaitu: (1) aksi perangkat internal, (2) transfer data, dan (3) _____.
Setiap sinyal status mengindikasikan suatu kondisi spesifik perangkat. Ada tiga jenis sinyal status yaitu: (1) pemilihan antarmuka, (2) transfer data, dan (3) _____.
Ada beberapa metode yang digunakan dalam desain/perancangan hardware pengontrol I/O: (1) desain hardwired (random logic), (2) desain berbasis LSI fungsi khusus, (3) desain berbasis mikroprosesor, dan (4) _____.
10. Terminal hardware yang dapat mengirim dan menerima data, disebut _____.
11. Port I/O yang diperlukan seperti layaknya lokasi-lokasi memori, disebut _____.
12. Pada skema port I/O yang diperlukan berbeda dengan lokasi memori dan program menggunakan instruksi IN dan OUT untuk berkomunikasi dengan port-port, disebut _____.
13. Ada dua jenis transfer data berdasarkan pada mekanisme dari pewaktuan data yaitu: (1) sinkron, dan (2) _____.
14. Pada transfer metode sinkron, pengiriman dan penerimaan unit adalah disuplai dengan sinyal _____ yang sama.
15. Pada transfer sinkron, bila master mengirim data ke slave, data ditransmisikan oleh master tanpa mengharapkan _____ dari slave sebagai tanda terima/respons.
16. _____ adalah suatu sinyal atau suatu event yang digunakan untuk meminta CPU supaya menunda program yang sedang berjalan dan mengambil program lain.

17. Program yang dijalankan oleh CPU karena adanya kejadian/event penundaan program utama dikenal dengan nama _____.
18. Secara umum ada dua jenis interupsi yaitu: (1) interupsi _____, dan (2) interupsi _____.
19. Ada tiga aksi berbeda yang dilakukan secara otomatis oleh hardware CPU pada pengindraan (*sense*) sebuah interupsi yaitu: (1) penyimpanan status CPU, (2) menemukan penyebab interupsi, dan (3) _____.
20. Jika CPU tidak ingin melayani interupsi karena alasan penundaan, maka digunakan sebuah flag yang dikenal dengan _____ yang digunakan sebagai pengabaian atau penundaan pelayanan interupsi tersebut.
21. Interupsi yang berada dalam interupsi (interupsi berlapis) disebut _____.
22. Ada dua jenis transfer data yaitu: (1) transfer dari perangkat input ke memori, dan (2) transfer dari _____.
23. Ada dua teknik pelaksanaan transfer data yaitu: (1) tanpa melalui CPU (metode hardware), dan (2) _____.
24. Pada transfer data tanpa melalui CPU (metode hardware), program mendelegasikan tanggung jawab pelaksanaan operasi I/O ke unit hardware lain yang disebut _____.
25. Teknik transfer data melalui CPU terbagi dua yaitu: (1) programmed I/O mode, dan (2) _____.
26. _____ adalah proses penentuan master bus yang mengontrol bus pada suatu waktu yang diberikan ketika ada suatu permintaan dari satu atau lebih master bus.
27. Bus arbiter memutuskan yang mana yang akan menjadi master bus sekarang. Ada dua jenis pengawas bus yaitu: (1) _____, di mana suatu pengawas didedikasikan mempunyai fungsi sebagai pengawas bus. Dan (2) _____, di mana semua master bus bekerja sama melakukan pengawasan. Dalam hal ini setiap master bus mempunyai sebuah seksi pengawasan.
28. Ada tiga skema populer untuk arbitrasi bus yang digunakan dalam sistem komputer yaitu: (1) metode daisy-chain, (2) metode polling atau rotating priority, dan (3) metode _____.
29. Komunikasi dalam sistem komputer modern dilakukan dengan berbagai jenis bus, yaitu: (1) Bus Lokal, (2) Bus Sistem, (3) Bus I/O, dan (4) _____.

30. Bus PCI merupakan suatu bus lantai tengah (*mezzanine bus*) yang bertindak sebagai bus pertengahan antara (1) bus sistem, dan (2) bus _____.

SOAL-SOAL LATIHAN

1. Sebuah komputer berbasis Mikroprosesor Intel 80286 beroperasi pada *zero-wait state* pada frekuensi clock 12 MHz. Hitung bandwidth yang ditawarkan oleh sistem. Setiap siklus bus mentransfer dua byte dan dua state (periode clock) dalam satu siklus bus karena itu kita mempunyai sistem *zero-wait state*.
2. Sebuah Sistem PCI 64 bit beroperasi pada 66 MHz. Hitung bandwidth.

BAB 10

PERIPHERAL KOMPUTER

Sasaran bab ini:

1. Keyboard
2. Mouse
3. Tampilan CRT
4. Tampilan LCD
5. Printer
6. Disk Magnetik
7. Pita Magnetik
8. Disk Optik
9. Modem
10. Scanner
11. Plotter

Perangkat I/O yang dicantolkan ke komputer disebut juga peripheral. Di antara peripheral yang paling umum adalah keyboard, unit display, dan printer. Peripheral yang menyediakan penyimpanan cadangan untuk sistem adalah disk dan pita magnetik. Peripheral merupakan perangkat elektromekanik dan elektromagnetik yang agak kompleks.

Subsistem input-output suatu komputer yang disebut I/O menyediakan mode komunikasi yang efisien antara sistem sentral dan lingkungan luar. Program dan data harus dimasukkan ke memori komputer untuk pemrosesan dan hasil komputasi yang diperoleh harus direkam atau ditampilkan untuk pemakaian. Komputer tidak akan berguna tanpa kemampuan menerima informasi dari sumber luar dan mentransmisikan hasilnya dalam bentuk yang dapat dimanfaatkan.

Di sisi lain, unit CPU merupakan perangkat yang dapat melakukan operasi/pemrosesan yang paling cepat. Bila masukan informasi yang ditransfer ke prosesor melalui keyboard yang lambat, maka prosesor akan banyak membuang waktu ketika menunggu informasi tiba. Untuk menggunakan komputer secara efisien, maka program dan data yang besar harus dipersiapkan sebelumnya dan ditransmisikan ke dalam media penyimpanan seperti pita magnetik atau disk. Informasi dalam disk kemudian ditransfer ke dalam memori komputer pada rate yang cepat. Hasil-hasil program juga ditransfer ke dalam media penyimpanan yang cepat seperti disk, di mana mereka dapat ditransfer kemudian ke printer untuk menyediakan hasil print-out.

Monitor Video merupakan peripheral yang sangat umum yang terdiri atas keyboard sebagai perangkat input dan unit display sebagai perangkat output. Ada beberapa jenis monitor video yaitu CRT (*cathode ray tube*) dan LCD (*liquid crystal display*).

Pada bagian ini kita akan membahas peripheral yang merupakan salah satu peralatan yang dibutuhkan untuk berkomunikasi dengan dunia luar dan perangkat pelengkap lainnya.

10.1 KEYBOARD

Cara memasukkan informasi/data ke dalam komputer yang paling dikenal adalah melalui sebuah typewriter seperti keyboard yang memungkinkan manusia untuk memasukkan informasi alphanumerik secara langsung. Setiap sebuah tombol ditekan, terminal mengirimkan sebuah karakter yang dikodekan secara biner ke dalam komputer. Kecepatan pemasukan informasi yang tercepat dengan cara ini sangat tergantung pada kecepatan pengetikan manusia.

Keyboard digunakan oleh user untuk memasukkan data atau karakter ke dalam komputer secara manual. Input tersebut kemudian ditransmisikan untuk ditampilkan pada monitor. Keyboard terdiri atas saklar-saklar tekan (tombol). Bila tombol ditekan maka akan terjadi kontak yang mengaktifkan salah satu karakter melalui encoding pada sirkuit elektronika. Selanjutnya kode 8-bit standar dibangkitkan dan dikirim ke komputer. Keyboard serial mengirimkan data dengan cara bit demi bit. Komputer kemudian mengubahnya menjadi data paralel 8-bit (byte).

Pada umumnya tombol keyboard dihubungkan dengan sebuah matriks baris dan kolom. Setiap tombol mempunyai koordinasi tetap dengan nomor baris dan nomor kolom. Fungsi-fungsi yang dikerjakan oleh keyboard elektronik meliputi:

- a) Pengindraan tombol yang ditekan
- b) Encoding
- c) Mengirimkan kode ke komputer

Keyboard mengikuti teknik standar pengenalan karakter yang dikenal dengan *scanning*. Ada beberapa jenis tombol keyboard, misalnya:

- a) *Mechanical keyswitch*
- b) *Membrane keyswitch*
- c) *Capacitive keyswitch*

10.1.1 Mechanical Keyswitch Keyboard

Keyboard jenis ini memiliki sebuah pegas yang ketika ditekan maka tombol akan terhubung dengan dua titik pada sebuah matriks keyboard yang mengaktifkan sebuah sinyal yang akan dirasakan oleh encoder. Pada saat tombol dilepas, pegas akan kembali ke posisi semula dan akan memutuskan kembali koneksi yang terjadi sebelumnya. Keyboard jenis ini ukurannya besar mudah melakukan perbaikan jika terjadi kegagalan.

10.1.2 Membrane Keyswitch Keyboard

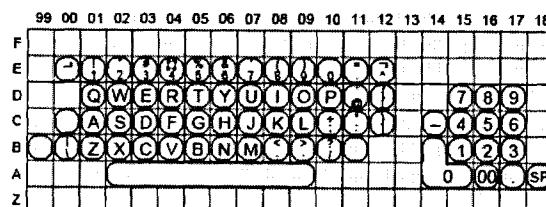
Pada keyboard *membrane switch*, terdapat sejumlah saklar membran yang berada di bawah tombol dan tidak mempunyai pegas. Pada saat tombol ditekan, sebuah sirkuit elektronika terhubung oleh dua kontak metal, dan pada saat dilepas maka sirkuit tersebut akan terputus. Keuntungan keyboard ini tidak bising.

10.1.3 Capacitive Keyswitch Keyboard

Pada keyboard jenis ini, kapasitansi antara dua plat penghantar diaktifasi oleh tombol. Plat-plat ditempatkan bersama sebuah celah kecil yang berisi lembaran milar. Pada saat tombol ditekan, pergerakan plunger mendorong plat yang mengurangi celah antaranya. Hal ini mengubah kapasitansi antar plat. Keyboard ini ukurannya kecil dan lebih tahan, tetapi jika terjadi kerusakan maka tidak mudah membetulkannya.

10.1.4 Sistem Tata Letak Keyboard

Keyboard yang umum digunakan adalah yang mempunyai tata letak tombol-tombol berdasarkan standar ECMA-23 (edisi kedua) seperti yang diperlihatkan pada Gambar 10.1. Tata letak QWERTY menyesuaikan dengan tata letak mesin ketik tradisional.



Gambar 10.1 Tata letak keyboard standar ECMA-23 (edisi kedua)

TABEL 10.1 Kode ASCII

MSD LSD	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	'	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	AC	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	-	o	DEL

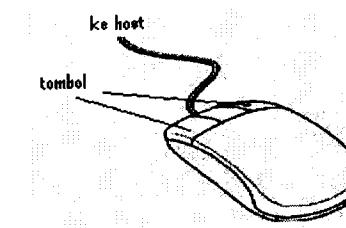
Bila sebuah tombol atau karakter ditekan, maka sebuah pola bit dibangkitkan dan ditransmisikan ke komputer. Kode 7-bit ASCII merupakan kode teks yang umum digunakan (lihat Tabel 10.1). Pola bit-bit yang dapat direpresentasikan hanya 128 pola ($2^7=128$), tetapi kebanyakan keyboard memperluas standar ECMA-23 dengan menambahkan tombol-tombol modifikasi (*shift*, *escape*, dan *control*) dan juga pola 7-bit tidak cukup panjang. Banyak alternatif yang memungkinkan, tetapi satu metode yang telah diterima secara luas adalah dengan menyediakan satu pola bit untuk tombol modifikasi dan pola bit lainnya tetap. Kode ASCII 7-bit merepresentasikan 5 macam kelompok kode:

1. Huruf kapital dan huruf kecil
2. Angka/numeral
3. Tanda baca
4. Simbol-simbol khusus
5. Karakter kontrol

Kode ASCII 7-bit direpresentasikan dalam sebuah byte di mana bit MSB (most significant bit, D7) adalah 0. Bit D7 ini dimungkinkan untuk digunakan sebagai parity bit atau untuk perluasan kode.

10.2 MOUSE

Mouse adalah perangkat input yang dapat digenggam yang terdiri atas bola karet pada bagian bawah dan satu atau lebih tombol yang terdapat pada bagian atas seperti yang diilustrasikan pada Gambar 10.2. Ketika mouse digerakkan maka suatu pointer kecil pada monitor juga akan bergerak yang mengizinkan pemakai untuk memilih atau menunjuk item atau menu yang ada di layar monitor.

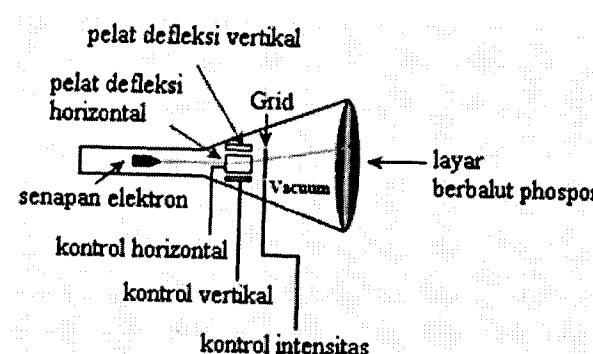


Gambar 10.2 Mouse

Jenis mouse lainnya adalah mouse optik yang tidak mempunyai roda atau bola karet melainkan menggunakan sebuah LED (*light emitting diode*) dan sebuah foto detektor pada bagian bawahnya. Mouse optik menggunakan *mousepad* khusus yang bersifat reflektif dan absorptif yang terdiri atas garis-garis horizontal dan vertikal. Mouse dideteksi melalui transisi antara daerah reflektif dan absorptif tersebut.

10.3 TAMPILAN CRT

Tampilan video atau monitor terbuat dari perangkat tampilan *luminescent* seperti CRT (*cathode ray tube*), atau suatu panel kristal cair (*liquid crystal display, LCD*) dan sirkuit kontrol. Pada CRT terdapat sebuah senapan elektron yang dapat melontarkan seberkas sinar elektron pada layar berbalut phosphor. Dalam CRT, pelat defleksi horizontal dan vertikal mengemudikan berkas elektron yang menyapu (*sweep*) layar tampilan dengan cara raster (satu baris setiap saat, dari kiri ke kanan, dimulai dari atas). Konfigurasi CRT ditunjukkan pada Gambar 10.3. Tegangan positif pada grid (kisi) akan mengakselerasi elektron yang bermuatan positif ke arah layar, sebaliknya tegangan negatif akan menolak elektron menjauhi layar. Warna yang dihasilkan pada layar ditentukan oleh karakteristik phosphor. Pada CRT warna, terdapat tiga jenis phosphor (merah, hijau dan biru) yang di-interleave pada suatu pola teratur, dan tiga senapan elektron yang menghasilkan tiga berkas sinar yang didefleksikan ke layar secara bersamaan.



Gambar 10.3 CRT dengan satu senapan elektron

Penulisan informasi pada layar dikontrol oleh "dot clock", yang membangkitkan sebuah berkas 1 dan 0 bergantian secara kontinu pada kecepatan yang sesuai dengan waktu terbaru (*update time*) untuk sebuah titik tunggal pada layar. Sebuah titik tunggal disebut *picture element* atau pixel.

Mata manusia relatif lambat jika dibandingkan dengan kecepatan sebuah perangkat elektronika, dan tak dapat mengamati sebuah gerakan terputus yang terjadi pada kecepatan sekitar 25 Hz atau lebih. Layar komputer hanya memerlukan update 25 sampai 30 kali dalam satu detik untuk mengamati sebuah citra kontinu. Sedangkan video monitor untuk aplikasi-aplikasi komputer dapat mempunyai suatu kecepatan scan yang diinginkan oleh perancang monitor dan kartu antarmuka video. Pada aplikasi televisi, kecepatan scan harus standarisasi. Di Eropa untuk televisi standar digunakan 25 Hz dan di Amerika Utara digunakan 30 Hz. Jenis phosphor yang digunakan pada layar tidak mempunyai tingkat kontinuitas yang panjang, dan karena itu scan baris diperbarui (*update*) secara bergantian untuk mengurangi kedipan (*flicker*). Sehingga, layar diperbarui pada kecepatan 50 Hz di Eropa dan 60 Hz di Amerika Utara. Banyak peneliti percaya bahwa frekuensi 50 Hz Eropa merupakan pilihan yang buruk, karena banyak gambar yang mengalami kedipan.

Kecepatan data antara komputer dan video monitor dapat ditingkatkan sangat tinggi. Misalnya sebuah monitor 24 bit per pixel dengan resolusi 1024 x 768 pixel dan kecepatan refresh 60 Hz akan memerlukan sebuah bandwidth (yakni, jumlah informasi yang dapat di bawah pada suatu periode waktu) $3 \text{ byte/pixel} \times (1024 \times 768) \text{ pixel} \times 60 \text{ Hz}$ atau kira-kira sama dengan 140 MB/sec.

Keuntungan monitor CRT

- Warna akurat—CRT memberikan warna yang lebih kaya dalam berbagai macam dibandingkan LCD
- Waktu respons—dapat merespons lebih cepat dari sejumlah LCD yang berkaitan dengan video dan permainan game
- Sudut pandang—CRT menyediakan kecerahan (*brightness*) dan warna dari berbagai sudut pandang.
- Multi-resolusi
- Harga—CRT lebih murah dari LCD

Kekurangan monitor CRT

- Sangat berat dan besar
- Menggunakan energi yang besar
- Menghasilkan panas yang lebih

10.4 TAMPILAN LCD

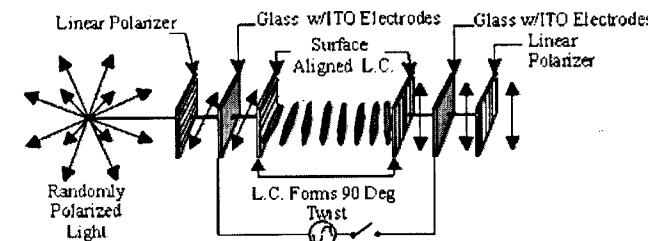
Saat ini teknologi LCD (tampilan kristal cair) semakin berkembang dan penggunaannya sudah semakin meluas dan populer. Penggunaan LCD misalnya selain terdapat di dalam jam tangan, kalkulator, televisi, sekarang ini banyak digunakan untuk komputer notebook. Teknologi LCD untuk monitor komputer ini selain ringan dan tipis juga hemat daya listrik dibandingkan dengan pendahulunya CRT.

Perkembangan kinerja LCD ditandai yang pertama adalah dengan munculnya jenis segmen yang hanya dapat menampilkan angka, kemudian diikuti dengan sistem *dot-matrix* yang mampu menampilkan karakter dan grafik. Tampilan LCD kemudian berkembang dari monokrom menjadi berwarna, mulai dari hanya gambar citra diam sampai citra bergerak, dan dari ukuran kecil hingga layar besar. Ada tiga evolusi teknologi LCD: (1) *Drive systems*; (2) *LCD systems*; (3) *Peripheral technologies*.

Karena teknologi LCD sangat kompleks dan memiliki banyak variasi dan mengalami perkembangan yang sangat cepat maka bahasan mengenai LCD ini perlu disederhanakan.

10.4.1 Operasi Dasar LCD

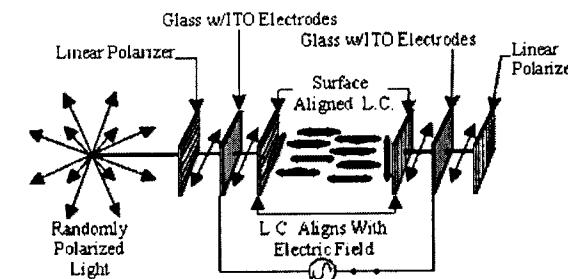
LCD terdiri atas susunan atau lapisan seperti *sandwich*. LCD terdiri atas dua substrat yang membentuk sebuah “*flat bottle*” yang berisi campuran kristal cair. Di dalam permukaan *bottle* atau sel dibungkus dengan polimer yang disangga untuk meluruskan/menyejajarkan molekul-molekul kristal cair. Molekul-molekul kristal cair disejajarkan di atas permukaan dalam arah *buffing*. Untuk *twisted nematic device*, dua permukaan disangga secara ortogonal satu sama lain, membentuk sudut 90 derajat dua kali lipat dari permukaan satu terhadap yang lain (lihat Gambar 10.4).



Gambar 10.4 Kristal cair kembar 90 derajat

Struktur *helical* mempunyai kemampuan untuk mengontrol cahaya. Sebuah pemolarisasi (*polarizer*) digunakan pada bagian depan dan *analyzer/reflector* digunakan pada bagian belakang sel. Bila cahaya polarisasi-acak melewati pemolarisasi depan (*front polarizer*) maka akan dipolarisasi secara linier. Kemudian selanjutnya melewati glass depan dan dirotasi oleh molekul-molekul kristal cair dan melewati glass belakang. Jika analyzer berotasi 90 derajat terhadap pemolarisasi, maka cahaya akan melewati analyzer dan terpantul kembali melalui sel. Pengamat akan melihat latar-belakang display, dalam hal ini adalah *silver gray* dari reflektor.

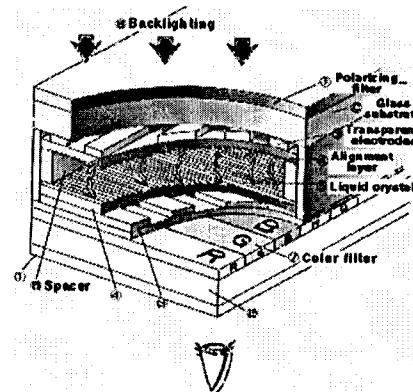
Bila sinyal kemudi yang tepat dikenakan pada elektroda sel, maka medan listrik di set-up melintasi sel. Molekul-molekul kristal cair akan berotasi dalam arah medan listrik. Cahaya terpolarisasi linier yang datang melalui sel tidak terpengaruh dan diserap oleh bagian belakang analyzer. Pengamat melihat sebuah karakter hitam pada latar-belakang silver gray (lihat Gambar 10.5). Bila medan listrik mati (*off*), maka molekul-molekul relaksasi kembali ke struktur kembarnya 90 derajat. Hal ini disebut sebagai citra positif (*positive image*), *reflective viewing mode*.



Gambar 10.5 Kristal cair dengan medan listrik

10.4.2 Struktur LCD dan Proses Produksi

Pada Gambar 10.6 ditunjukkan deskripsi sederhana dari struktur material LCD dan proses produksi dari sebuah *simple matrix LCD*.



Gambar 10.6 Struktur sandwich

LCD warna mempunyai struktur yang komponen-komponennya dibentuk seperti susunan sebuah *sandwich*:

1. *Polarizing filter*. Filter ini mengontrol cahaya yang masuk dan yang keluar.
2. *Glass substrate*. Substrat ini menghentikan pemfilteran secara elektrik dari elektroda
3. *Transparent electrodes*. Elektroda ini mengemudikan (*drive*) LCD. Material transparan tinggi yang digunakan tidak akan mengganggu kualitas dari *image's integrity*.
4. *Alignment layer*. Film digunakan untuk meluruskan molekul-molekul dalam suatu arah yang tetap.
5. *Liquid crystals*
6. *Spacer*. Menjaga dan mempertahankan *uniform space* di antara pelat glass.
7. *Color filter*. Warna dinyatakan melalui penggunaan filter R, G, B
8. *Backlighting*

10.4.3 Jenis-Jenis Tampilan

Ada tiga metode dasar yang digunakan untuk menampilkan huruf, angka, dan grafik:

- **Sistem segmen**
Unit-unit tampilan panjang disusun membentuk gambar angka '8' untuk menampilkan angka.
- **Sistem Dot matrix (tampilan karakter)**
Unit-unit tampilan disusun dalam baris kolom untuk membentuk karakter.
- **Sistem dot matrix (tampilan grafis)**
Unit-unit tampilan disusun dalam baris dan kolom untuk membentuk grafis.



10.4.4 Prinsip-Prinsip Tampilan Warna

Tampilan warna dibuat dengan menempatkan filter warna pada setiap unit tampilan. Pada sistem dot matrix, dot merah, hijau, dan biru diperoleh melalui penggunaan filter pada masing-masing tiga warna utama Red (R), Green (G), dan Blue (B). Warna-warna yang lain dapat dibentuk dengan menggabungkannya.

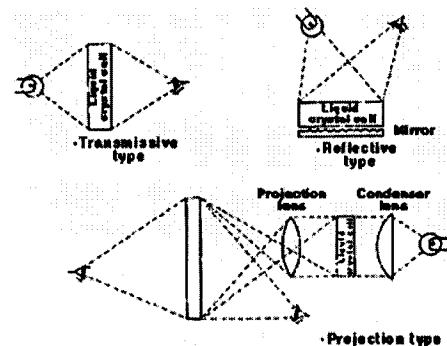
Pada LCD berwarna seperti monitor terdapat banyak sekali titik sinar (pixel) yang terdiri atas satu buah kristal cair sebagai sebuah titik sinar. Meski disebut sebagai titik sinar, namun kristal cair ini tidak memancarkan sinar sendiri. Sumber sinar di dalam sebuah perangkat LCD adalah lampu neon berwarna putih di bagian belakang susunan kristal cair tadi. Titik sinar yang jumlahnya puluhan ribu bahkan jutaan inilah yang membentuk tampilan citra. Kutub kristal cair yang dilewati arus listrik akan berubah karena pengaruh polarisasi medan magnetik yang timbul dan karenanya akan hanya membiarkan beberapa warna diteruskan sedangkan warna lainnya tersaring.

10.4.5 Konfigurasi Tampilan

Cahaya yang melalui kristal cair adalah hanya cahaya natural (alami) atau cahaya ambien buatan (*artificial ambient light*). Konfigurasi tampilan

dikategorisasi oleh posisi relatif dari sumber cahaya. Ada tiga jenis (Gambar 10.7):

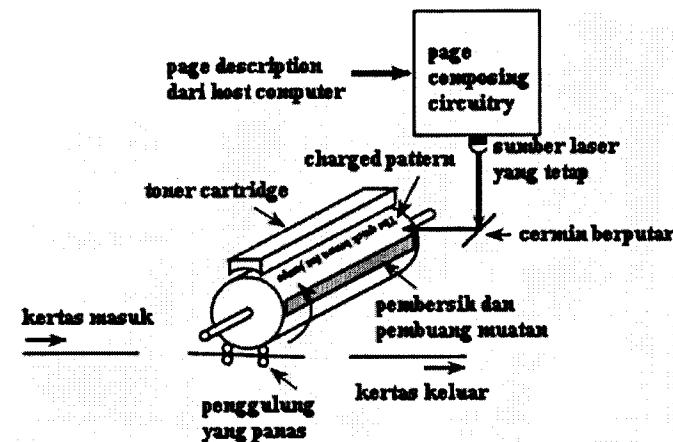
1. *Transmissive type* (TV LCD)
2. *Reflective type* (kalkulator LCD, Jam)
3. *Projection type* (LCD projection)



Gambar 10.7 Konfigurasi LCD

10.5 PRINTER

Printer adalah peripheral yang bertujuan untuk mencetak karakter, grafis atau citra di atas kertas. Ada dua kelompok printer yaitu *impact printer* dan *non-impact printer*. Pada *impact printer*, karakter dicetak berdasarkan kontak fisik (atau tekanan) antara *head* (hammer, pin, atau font) dengan pita tinta ke atas kertas. Misalnya printer *dot matrix*, printer *daisy wheel*, printer *drum*, dan sebagainya. Sedangkan pada non-impact printer tidak ada kontak fisik antara *head* dengan kertas atau pita. Misalnya printer *termal*, printer *ink jet*, printer *laser*, dan sebagainya. Pada bagian ini kita membahas printer *laser* yang merupakan teknologi yang lebih maju.



Gambar 10.8 Diagram skematik printer laser

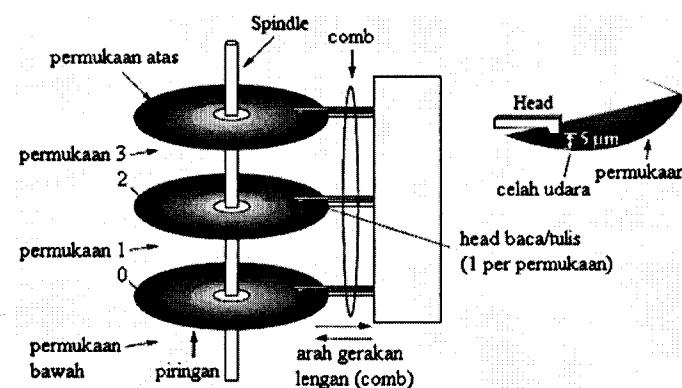
Printer laser terbuat dari drum bermuatan di mana laser membuat muatannya pada daerah yang dipilih yang berkenaan dengan representasi bit yang dipetakan pada suatu halaman yang akan dicetak. Daerah yang bermuatan memungut bubuk toner yang sensitif secara elektrostatis. Drum berputar, toner ditransfer ke kertas dan dipanaskan agar melekat di atas kertas. Drum dibersihkan dari berbagai residu dan proses yang sama berulang ke halaman berikutnya. Diagram skematik printer laser ditunjukkan pada Gambar 10.8. Karena toner berupa plastik, bukan tinta, maka dia tidak terabsorpsi ke dalam kertas, melainkan menempel di atas permukaan.

10.6 DISK MAGNETIK

Disk magnetik adalah perangkat untuk penyimpanan informasi yang mendukung kapasitas penyimpanan yang besar dan mempunyai waktu akses yang relatif cepat. Disk magnetik mempunyai head yang dapat bergerak pada tumpukan satu atau lebih piringan yang berjarak beberapa milimeter satu dengan yang lainnya dan ditopang oleh sebuah *spindle* seperti yang ditunjukkan pada Gambar 10.9. Setiap piringan mempunyai dua permukaan yang terbuat dari aluminium atau *glass* (lebih tahan panas) yang dibungkus dengan partikel-partikel kecil dari bahan magnetik seperti besi oksida.

Bahan dasar (substrat) yang terbuat dari glass mempunyai keuntungan:

1. Permukaannya lebih merata sehingga memberikan keandalan yang lebih baik.
2. Kecacatan/kerusakan pada permukaan lebih sedikit sehingga mengurangi error yang terjadi saat pembacaan/penulisan.
3. Lebih baik dalam menangani/mendukung *flying height*.
4. Kekakuannya (*stiffness*) lebih baik untuk mengurangi/menjaga pengaruh dinamis disk.
5. Lebih tahan terhadap guncangan dan kerusakan.
6. Lebih tahan panas.



Gambar 10.9 Disk magnetik dengan tiga piringan

10.6.1 Mekanisme Pembacaan dan Perekaman

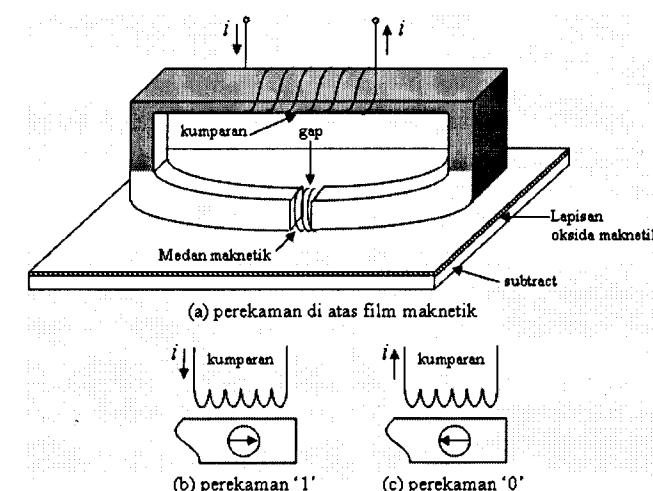
Sebuah head dideendasikan pada setiap permukaan. Ada enam head yang digunakan dalam Gambar 10.9 pada enam permukaan. Permukaan atas pada piringan atas dan permukaan bawah pada piringan bawah kadang-kadang tidak digunakan pada disk multipiringan karena mereka lebih mudah terkena kontaminasi dibanding dengan permukaan bagian dalam. Head diikatkan pada sebuah lengan bersama (disebut juga *comb*) yang bergerak keluar masuk untuk mencapai bagian-bagian permukaan.

Pada head, terdapat arus listrik i di dalam kumparan (*coil*) yang ditunjukkan pada Gambar 10.10(a) akan menghasilkan medan magnetik melintasi suatu celah (*gap*). Bagian medan magnetik ini akan menyebar ke material/bahan magnetik di bawah celah dan material akan dimagnetisasi

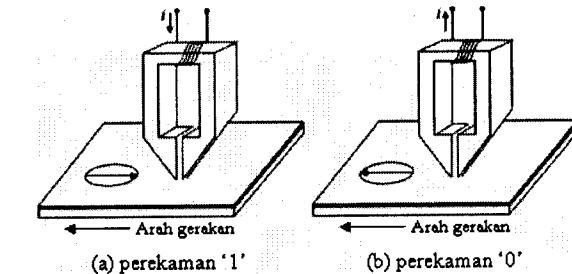
pada suatu arah yang tetap. Bila arus listrik dihilangkan, maka bagian titik yang termagnetisasi akan tetap tidak berubah, seperti yang ditunjukkan pada Gambar 10.10(b). Jadi, informasi akan tersimpan. Jika arah arus listrik dibalik, maka bagian titik agak dimagnetisasi kembali tetapi dengan arah yang berlawanan, seperti yang ditunjukkan pada Gambar 10.10(c). Dari sini jelas tergambar sebuah sistem biner yang dapat digunakan untuk menyimpan informasi. Misalnya bagian b dari Gambar 10.10 sebagai '1' (high) dan bagian c sebagai '0' (low). Memberikan arus listrik i artinya sama dengan merekam '0' atau '1' sebagai data.

Sekarang jika suatu titik tetap yang termagnetisasi dengan arah tertentu dilewati oleh celah head (Gambar 10.11a), maka arus listrik dengan arah yang ditunjukkan akan mengimas/menginduksi kumparan. Tetapi jika suatu bagian titik termagnetisasi dengan arah berlawanan dilewati oleh celah head, maka arus listrik akan menginduksi dengan arah berlawanan seperti yang ditunjukkan pada Gambar 10.11(b). Pendekatan arah bagian titik yang termagnetisasi dengan pengukuran arus induksi merupakan pembacaan informasi ('1' atau '0').

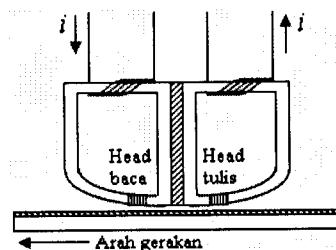
Head magnetik baca/tulis yang sama pada Gambar 10.10(a) dapat digunakan untuk menulis data digital atau membaca data digital. Namun, saat ini head yang umum digunakan adalah *dual head* yang terintegrasi sebagai head baca dan head tulis seperti pada Gambar 10.12.



Gambar 10.10 Perekaman disk magnetik



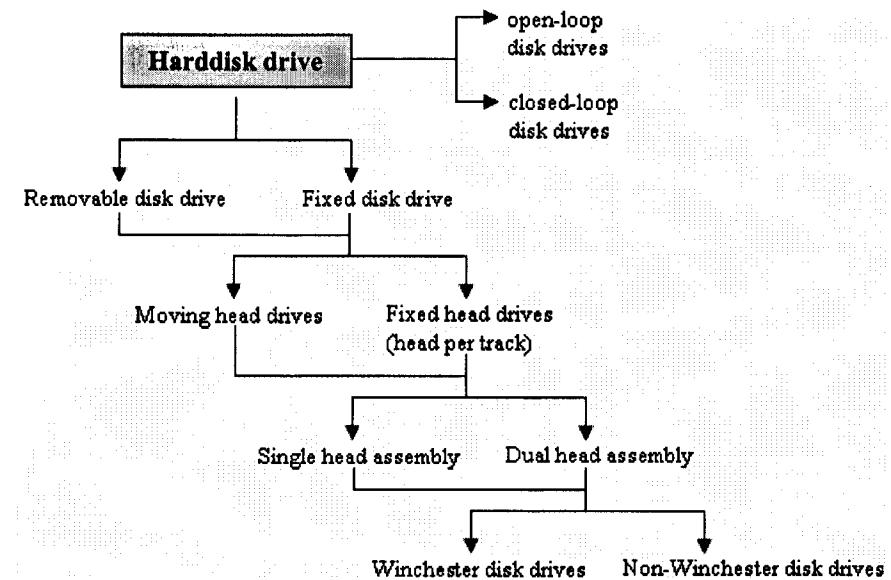
Gambar 10.11 Perekaman dengan head terpisah



Gambar 10.12 Perekaman dengan dual head

Pada *hard disk drive*, piringan berputar pada kecepatan konstan khas 3600 sampai 15.000 revolusi per menit (rpm). Pada *floppy disk drive* rotasi yang khas antara 300 dan 600 rpm. Head membaca atau menulis data dengan memagnetisasi bahan magnetik tepat ketika head berada di atas permukaan yang diinginkan. Hanya satu head yang digunakan ketika membaca atau menulis, sehingga data disimpan dengan cara serial walaupun head pada prinsipnya dapat digunakan untuk membaca atau menulis beberapa bit secara paralel. Salah satu alasan mengapa mode operasi paralel tidak umum digunakan karena pada head dapat terjadi *misaligned* (tidak sejajar), yang mengakibatkan hilangnya data ketika dibaca atau ditulisi.

Hard disk drive mempunyai banyak jenis seperti yang ditunjukkan pada Gambar 10.13.



Gambar 10.13 Jenis-jenis hard disk drive

Pada *removable harddisk drive*, disk dapat dipindahkan/dilepaskan jika tidak digunakan. Karena itu dapat digunakan untuk *disk pack* yang lain, selain itu data yang dibuat pada satu disk drive dapat ditransfer ke drive/kompute lain. Pada *fixed disk*, disk tidak dapat dilepaskan dari disk drive.

Pada *moving head disk drive*, head baca/tulis bergerak dari satu track ke track lain seperti yang diperintahkan oleh komputer. *Moveable head* hanya terdapat satu pada setiap permukaan piringan. Pada *fixed disk drive*, head baca/tulis tetap tidak dapat berpindah. Karena itu harus satu head untuk satu track.

Pada *single head*, ada satu head untuk setiap permukaan. Semua ini dicantolkan pada sebuah spindle dan selalu bergerak bersama. Pada saat yang sama semua head berada pada silinder yang sama.

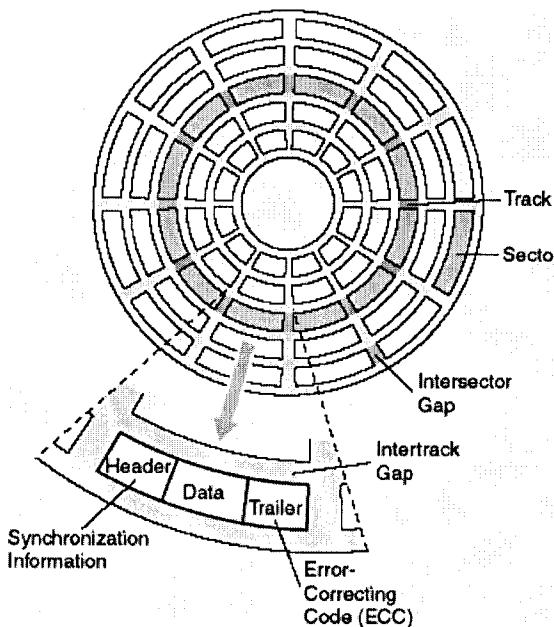
Pada *dual head*, ada dua head untuk setiap permukaan. Bila satu set (kumpulan) head berada dalam posisi awal, yaitu silinder 0, maka kumpulan head lainnya ditempatkan di atas silinder nomor pertengahan. Ketika head kumpulan pertama tiba pada silinder nomor pertengahan, maka head

kumpulan kedua meraih atau menuju ke silinder terakhir. Jadi satu kumpulan head berada pada silinder paruh pertama dan kumpulan head kedua berada pada silinder paruh kedua.

Istilah Winchester merujuk pada suatu teknologi baru yang diperkenalkan oleh IBM pada tahun 1973. Semua disk drive di pasaran hingga tahun 1973 merupakan *non-Winchester disk drive*. Teknologi Winchester dianggap suatu revolusi atau terobosan baru dalam harddisk drive. Fitur utama teknologi Winchester adalah:

1. Head baca/tulis dan disk ditempatkan dalam suatu tempat yang tersegel.
2. Head melayang berada sangat dekat dengan hard disk sekitar lebih kecil dari 19 mikro inch.
3. Head diparkir pada zona parkir (*landing zone*) ketika disk tidak berputar. Head akan melayang atau mendarat pada lapisan tipis di udara ketika disk mulai start. Tidak ada data yang direkam pada *landing zone*.
4. Permukaan disk diberi pelumas untuk mencegah kerusakan pada head atau track.

Harddisk Winchester tidak memerlukan perawatan preventif. Densitas track yang lebih tinggi dan densitas bit-bit diperoleh pada teknologi Winchester. Walaupun terdapat sistem preventif untuk melawan kerusakan head, yaitu sentuhan head disk, suatu kecelakaan head-crash (kerusakan head) dapat merusak data yang tersimpan dalam disk. Karena itu pemakai diharuskan mempunyai back-up data.

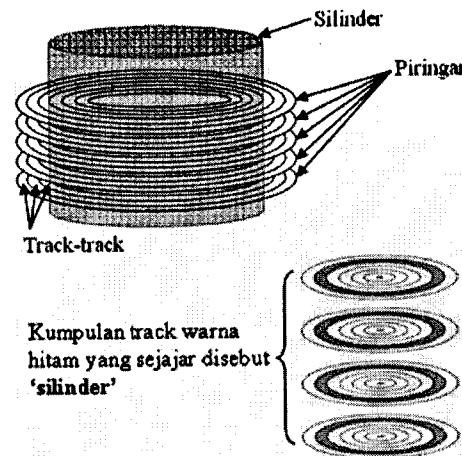


Gambar 10.14 Ilustrasi track dan sektor

10.6.2 Track dan Spot

Pada Gambar 10.14 ditunjukkan ilustrasi track dan sektor. Permukaan disk dibagi dalam lingkaran-lingkaran konsentrik (lingkaran dalam lingkaran) yang disebut track. Pada setiap permukaan terdapat ribuan track. Track yang lebih tebal menyimpan lebih besar. Bit-bit data direkam sebagai titik magnetik kecil di atas track. Spot yang lebih kecil, mempunyai bit per inch lebih besar dan penyimpanan yang lebih besar.

Kumpulan semua track yang berada dalam posisi yang relatif sama/sejajar pada susunan piringan disebut silinder (lihat Gambar 10.15).



Gambar 10.15 Susunan piringan dan track

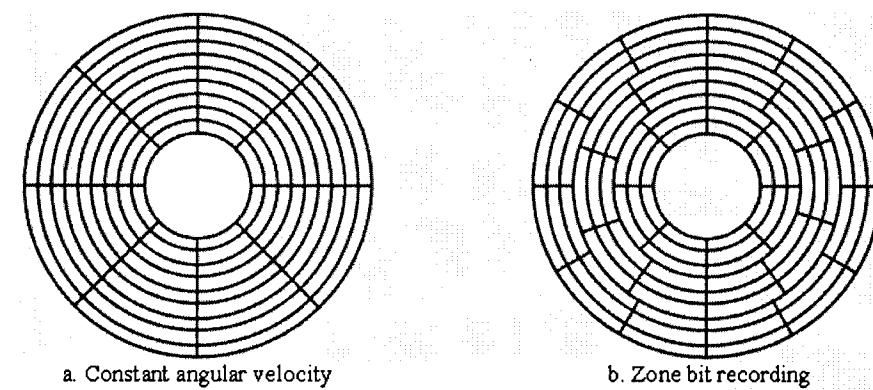
10.6.3 Sektor

Lebih jauh track dibagi menjadi sektor-sektor, yang menyimpan sebuah blok data yang dibaca atau ditulsi dalam satu waktu; mislanya, READ SECTOR 782, WRITE SECTOR 5448. Pada setiap track terdapat ratusan sektor. Untuk memperbarui disk, satu atau lebih sektor dibaca ke dalam komputer, diubah dan ditulsi kembali ke dalam disk. Sistem operasi yang mengatur bagaimana menetapkan data ke dalam ruang tetap ini.

Di antara track terdapat gap (pemisah), demikian halnya pada sektor (Gambar 10.14). Hal ini bertujuan mencegah atau mengurangi error karena adanya *misalignment* (kesemrawutan) atau ketidak lurusan head atau adanya interferensi/gangguan medan-medan magnetik.

Semakin mendekati pusat disk, maka ukuran fisik sektor semakin kecil. Karena jumlah bit per sektor konstan, ukuran sebuah bit juga berkurang. Bit yang dekat dengan pusat disk yang berputar melewati sebuah titik tetap (seperti head baca/tulis) kecepatannya lebih lambat daripada bit terluar. Karena itu, beberapa cara harus dicari untuk mengompensasi variasi kecepatan agar head dapat membaca semua bit dengan kecepatan yang sama. Hal ini dapat dilakukan dengan meningkatkan ruang spasi antara bit-bit informasi yang direkam pada segmen-segmen disk sehingga informasi kemudian dapat di-scan dengan kecepatan yang sama pada kecepatan

rotasi disk yang tetap. Cara ini dikenal dengan kecepatan sudut tetap, *constant angular velocity* (CAV). Tata letak disk yang menggunakan CAV dapat dilihat pada Gambar 10.16. Disk dibagi menjadi sejumlah sektor berbentuk kue pie ke dalam serangkaian track-track konsentrasi. Keuntungan menggunakan CAV adalah bahwa blok-blok data individu dapat langsung dialami oleh track dan sektor. Untuk menggerakkan head dari lokasi saat ini ke suatu alamat khusus, maka dia hanya membutuhkan gerakan head yang pendek ke suatu track khusus dan waktu tunggu yang singkat bagi sektor yang tepat untuk berputar di bawah head. Tetapi penggunaan CAV mempunyai kelemahan karena jumlah data yang dapat disimpan pada track terluar sama dengan jumlah yang dapat disimpan pada track pendek yang terdalam di pusat disk.



10.16 Perbandingan organisasi sektor

Karena kapasitas penyimpanan disk dari sebuah sistem CAV dibatasi oleh kepadatan (*density*) perekaman maksimum yang dapat dicapai pada track terluar, maka untuk meningkatkan kepadatan tersebut, digunakan teknik yang lebih modern yang dikenal dengan *zone bit recording*, di mana permukaan disk dibagi menjadi sejumlah zona-zona. Jumlah bit per track dalam sebuah zona adalah konstan. Zona terjauh dari pusat disk mempunyai jumlah bit yang lebih banyak daripada zona yang dekat dengan pusat disk. Hal ini mengizinkan kapasitas penyimpanan keseluruhan yang lebih besar tetapi dibayar dengan sirkuit yang agak lebih kompleks. Dengan head yang bergerak dari satu zona ke zona lain, maka panjang bit-bit

individu berubah yang menyebabkan perubahan waktu pembacaan maupun penulisan.

10.6.4 Parameter-Parameter Kinerja Disk

Parameter-parameter yang menentukan kinerja suatu disk meliputi *seek time*, *rotational delay*, *access time*, dan *transfer time*.

Disk berotasi atau berputar secara konstan ketika disk drive dioperasikan. Untuk melakukan pembacaan atau penulisan, maka head harus ditempatkan pada posisi sektor di atas track yang diinginkan. Pemilihan track yang tepat/sesuai dengan yang diinginkan dilakukan oleh sistem *movable head*. Pada sistem *fixed head*, pemilihan satu *head* dilakukan secara elektronik. Pada sistem *movable head* waktu yang dibutuhkan oleh pergerakan head untuk mencari track yang sesuai disebut *seek time*. Sedangkan waktu yang dibutuhkan oleh pergerakan putaran piringan untuk mencapai sektor yang sesuai disebut *rotational delay* atau disebut juga *rotational latency*. Jumlah/total waktu antara seek time jika ada dan rotational delay disebut dengan *access time* yang merupakan waktu untuk mulai melakukan pembacaan atau penulisan. Ketika head telah berada pada posisi yang sesuai, maka operasi baca atau tulis pada sektor dilakukan, dan ketika bagian data diperoleh dan ditransfer ke head disebut sebagai *transfer time*.

Transfer time ke atau dari disk bergantung pada kecepatan rotasi disk, yang dinyatakan dengan:

$$T = \frac{b}{Nr}$$

di mana T = *transfer time*; b = jumlah byte yang ditransfer; N = jumlah byte pada suatu track dan r = kecepatan rotasi dalam revolusi per detik. Jadi rata-rata jumlah *access time* dapat dinyatakan dengan:

$$T_a = T_s + \frac{1}{2r} + \frac{b}{Nr}$$

di mana T_s adalah *seek time* rata-rata. Perlu dicatat bahwa pada drive yang ber-zona, jumlah byte per track bervariasi sehingga memunculkan persoalan kalkulasi yang lebih kompleks atau menyulitkan.

Contoh 10.1 Waktu transfer hard disk

Anggap bahwa sebuah disk berputar sekali setiap 16 ms. *Seek time* untuk menggerakkan di antara head yang berdekatan adalah 2 ms. Ada 32 sektor per track yang menyimpan dengan urutan linier (*non-interleaved*), dari sektor 0 ke sektor 31. Head melihat sektor-sektor dalam urutan tersebut. Anggap head baca/tulis mulai diposisikan pada sektor 1 track 12. Ada memory buffer yang cukup besar untuk menyangga semua track. Data ditransfer antara lokasi-lokasi disk dengan membaca data sumber ke dalam memori, menempatkan head baca/tulis pada lokasi tujuan, dan menulis data ke tujuan.

- Berapa waktu dibutuhkan untuk mentransfer sektor 1 track 12 ke sektor 1 track 13?
- Berapa waktu dibutuhkan untuk mentransfer semua sektor pada track 12 ke sektor yang sesuai pada track 13? Perhatikan bahwa sektor-sektor tidak harus ditulisi dalam urutan yang sama ketika mereka dibaca.

Solusi:

Waktu untuk mentransfer sebuah sektor dari satu track ke track berikutnya dapat dibagi menjadi bagian-bagian: waktu baca sektor, waktu pergerakan head, delay rotasi, dan waktu tulis sektor.

Waktu untuk membaca atau menulis sebuah sektor adalah $(16 \text{ ms/track}) \times (1/32 \text{ track/sektor}) = 0,5 \text{ ms/sektor}$. Pada contoh ini, waktu pergerakan head hanya 2 ms karena head bergerak antara track yang berdekatan. Setelah pembacaan sektor 1 track 12, maka akan memakan waktu 0,5 ms. Ditambah dengan delay rotasi 15,5 ms yang dibutuhkan head berpindah pada sektor 1 kembali. Waktu pergerakan head 2 ms tumpang tindih (overlap) dengan delay rotasi 15,5 ms, dan juga hanya dua kali lebih besar (15,5 ms) dari yang digunakan.

Kita menjumlahkan waktu-waktu tersebut dan diperoleh: $0,5 \text{ ms} + 15,5 \text{ ms} + 0,5 \text{ ms} = 16,5 \text{ ms}$ untuk mentransfer sektor 1 track 12 ke sektor 1 track 13.

Waktu untuk mentransfer semua track 12 ke track 13 dihitung dengan cara yang sama. Memory buffer dapat menyimpan semua track, dan juga waktu untuk membaca atau menulis semua track mempunyai delay rotasi yang kecil untuk sebuah track yaitu 16 ms. Waktu pergerakan head 2 ms, yang juga adalah waktu untuk empat sektor lewat di bawah head (0,5 ms per sektor). Jadi, setelah pembacaan sebuah dan reposisi head,

maka head sekarang berada pada track 13, pada empat sektor yang lalu, awal sektor telah dibaca pada track 12.

Sektor dapat ditulisi dalam urutan yang berbeda dengan urutan ketika mereka dibaca. Jadi track 13 dapat ditulisi dimulai pada sektor 5. Waktu untuk menulis track 13 adalah 16 ms, dan waktu untuk semua transfer menjadi: $16\text{ ms} + 2\text{ ms} + 16\text{ ms} = 34\text{ ms}$. Catatan bahwa delay rotasi adalah nol untuk contoh ini karena head mendarat pada awal sektor pertama yang akan ditulisi.

10.6.5 Floppy Disk Drive versus Hard Disk Drive

Disk drive yang umum digunakan adalah *floppy disk drive* (FDD) dan *hard disk drive* (HDD). Pada HDD, media yang digunakan berupa piringan sirkular yang kaku. Pada FDD, medianya berupa disk sirkular yang lebih fleksibel yang dikenal dengan *diskette*. HDD memberikan kinerja yang lebih baik daripada FDD karena beberapa alasan:

1. Kapasitas penyimpanan data yang lebih besar
2. Waktu pengaksesan data lebih cepat
3. Kecepatan transfer data (*rate*) lebih besar
4. Keandalan operasi lebih baik
5. Kerusakan/hilang data atau error lebih sedikit

Beberapa konsep umum antara FDD dan HDD:

1. Data ditulis dalam bit demi bit pada disk
2. Untuk penambahan bit data, bit clock juga ditulis pada medium
3. Data direkam pada track sirkular yang konsentrik
4. Rotasi disk pada kecepatan tetap
5. Head merupakan head yang dapat bergerak

Beberapa perbedaan konsep antara FDD dan HDD:

1. Head pada floppy disk menyentuh permukaan media ketika operasi baca/tulis. Pada HDD, head baca/tulis tidak menyentuh permukaan media ketika operasi baca/tulis. Head melayang di atas disk pada jarak dekat, disebut sebagai *flying height*.
2. FDD mempunyai maksimum dua head baca/tulis karena diskette mempunyai dua permukaan. Hard disk dapat mempunyai banyak piringan yang ditopang oleh sebuah poros spindle sehingga HDD akan mempunyai multi-head baca/tulis. Setiap piringan mempunyai

dua permukaan, atas dan bawah. Jadi, total jumlah head baca/tulis yang diperlukan dalam HDD sama dengan dua kali jumlah piringan yang dimilikinya.

3. Mekanisme penempatan head pada FDD menggunakan sebuah motor stepper. Pada HDD tersedia dua opsi yaitu:
 - a) Mekanisme motor stepper (*open-loop disk drive*)
 - b) Mekanisme *voice coil servo* (*closed-loop disk drive*)
4. Diskette pada FDD diputar pada kecepatan rendah, biasanya pada 300 rpm atau 360 rpm. Sedangkan piringan pada HDD diputar pada kecepatan tinggi, biasanya pada 2400 rpm, 3600 rpm, 4800 rpm atau 7200 rpm.
5. Jumlah track pada floppy disk lebih sedikit—biasanya 40 atau 80. Kerapatan/densitas track biasanya 48 TPI (track per inch) atau 96 TPI. Pada HDD, densitas track memungkinkan lebih tinggi dan bahkan bisa sampai 1000 TPI. Jumlah track bergantung pada ukuran hard disk, 14", 8", 5 1/4", 3 1/2", atau 2 1/2".
6. Karena kecepatan rotasi hard disk lebih tinggi dari floppy diskette, maka densitas perekaman BPI (bit per inch) dalam HDD adalah lebih tinggi dari densitas pada FDD.

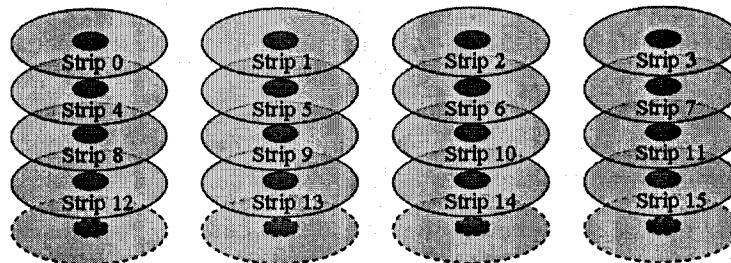
10.7 RAID

Pada tahun 1988, David Patterson, Garth Gibson dan Randy Katz dari Universitas California di Berkeley mengusulkan sistem penyimpanan yang berbasis banyak disk, yang disebutnya RAID (*Redundant Array of Inexpensive Disks*). Karena istilah "*inexpensive*" adalah relatif dan dapat menyesatkan, maka diusulkan mengganti akronim tersebut yang sekarang sudah umum diterima yaitu "*Redundant Array of Independent Disks*".

Pada makalah tersebut, Patterson, Gibson dan Katz menetapkan lima tipe (disebut level) RAID tetapi tidak menunjukkan hierarki. Setiap level mempunyai karakteristik kinerja dan keandalan yang berbeda-beda. Level-level yang orisinal adalah level 1 sampai level 5. Definisi RAID level 0 dan level 6 dikenal belakangan. Berbagai vendor (penjaja) menemukan level-level yang lain yang kemudian juga menjadi standar. Mereka umumnya mengombinasikan (*hybrid*) level-level RAID tersebut yang dapat diterima.

10.7.1 RAID Level 0

RAID level 0 atau RAID-0 menempatkan blok-blok data pada stripe dalam beberapa permukaan disk agar sebuah record menempati sektor atau beberapa permukaan disk seperti yang ditunjukkan pada Gambar 10.17. Metode ini disebut juga dengan *drive spinning*, *block interleave data striping* atau *disk striping*. Tidak seperti level RAID lainnya, RAID-0 tidak menawarkan *redundancy* sehingga RAID-0 memberikan kinerja yang paling baik khususnya jika kontrolernya dipisahkan dan digunakan cache untuk setiap disk. RAID-0 sangat mahal. Masalah dengan RAID-0 terletak pada fakta bahwa keandalan sistem keseluruhan hanya pada sebuah bagian yang akan diharapkan pada sebuah disk tunggal. Secara spesifik jika array berisi lima disk, masing-masing dengan desain umur 50.000 jam (atau sekitar 6 tahun), sistem keseluruhan mempunyai harapan desain umur $50.000/5 = 10.000$ jam (atau sekitar 14 bulan). Jika jumlah disk meningkat, probabilitas kegagalan meningkat pada titik di mana dia mencapai kepastian. RAID-0 tidak menawarkan *fault tolerance* karena tidak ada redundancy. Karena itu, RAID-0 hanya menawarkan keuntungan kinerja. Keandalannya benar-benar kurang. RAID-0 direkomendasikan untuk data yang tidak kritis (atau data yang sering mengalami perubahan dan di-back up secara reguler) yang memerlukan pembacaan dan penulisan kecepatan tinggi, dan harga murah dan yang digunakan pada aplikasi seperti video atau pengeditan citra.

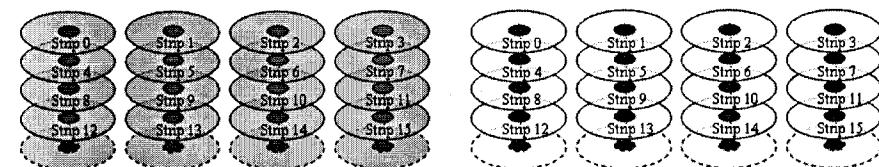


Gambar 10.17 RAID-0 (nonredundant)

10.7.2 RAID Level 1

RAID level 1 atau RAID-1 (dikenal juga dengan *disk mirroring*) memberikan proteksi kegagalan yang terbaik dari semua skema RAID. Setiap saat data ditulis dalam duplikasi pada set kedua dari drive yang

disebut *mirror set*, atau *shadow set* (seperti yang ditunjukkan pada Gambar 10.18). Susunan ini menawarkan kinerja yang dapat diterima, khususnya jika mirror drive disinkronisasikan 180° di luar rotasi dengan drive-drive utama. Walaupun kinerja pada penulisan lebih lambat dari RAID-0 (karena data harus ditulis dua kali), pembacaan lebih cepat karena sistem dapat membaca dari lengan disk yang terjadi dekat dengan sektor target. Hal ini mengurangi *rotational delay* setengahnya pada pembacaan. RAID-1 sangat cocok untuk orientasi transaksi, high-availability environment, dan aplikasi lain yang memerlukan fault tolerance tinggi seperti akuntansi atau daftar gaji.



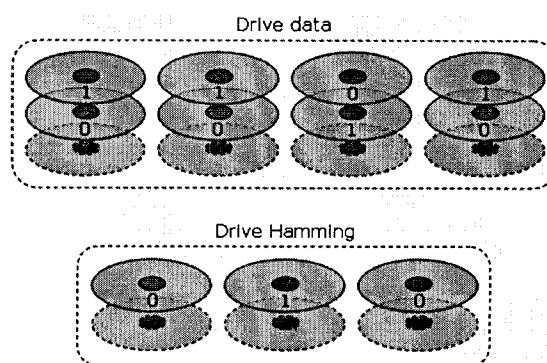
Gambar 10.18 RAID-1 (disk mirroring)

10.7.3 RAID Level 2

Masalah utama pada RAID-1 adalah harganya yang mahal; diperlukan jumlah disk dua kali lipat untuk menyimpan sejumlah data. Sebuah cara yang lebih baik adalah mencurahkan satu atau lebih disk untuk penyimpanan informasi pada disk yang lain. RAID-2 menetapkan salah satu dari dua metode ini.

RAID-2 memberikan ide *data striping* yang ekstrem. Sebagai pengganti penulisan data pada blok-blok dengan ukuran berubah-ubah, RAID-2 menulis satu bit per strip (seperti yang ditunjukkan pada Gambar 10.19). Hal ini memerlukan delapan permukaan yang hanya untuk mengakomodasi data. Drive-drive tambahan digunakan untuk informasi koreksi error yang dibangkitkan menggunakan sebuah kode Hamming. Jumlah drive kode Hamming diperlukan untuk membetulkan *single-bit error* yang sebanding dengan catatan jumlah drive-drive data yang akan diproteksi. Jika satu dari salah satu drive pada array gagal, maka word-word kode Hamming dapat digunakan untuk rekonstruksi drive yang gagal. (Jadi jelas, drive Hamming dapat dikonstruksi menggunakan drive-drive data.)

Karena satu bit ditulis per drive, keseluruhan disk set RAID-2 bertindak seolah-olah berada pada satu disk data yang besar. Jumlah total penyimpanan yang tersedia merupakan penjumlahan kapasitas penyimpanan dari drive-drive data. Semua drive, termasuk drive-drive Hamming—harus disinkronisasikan secara tepat, sebaliknya data menjadi teracak (*scramble*) dan drive-drive Hamming tidak baik. Pembangkitan kode Hamming memakan banyak waktu; jadi RAID-2 terlalu lambat untuk implementasi komersial pada umumnya. Saat ini hard drive mempunyai koreksi error CRC yang terpadu (*built-in*). Namun, RAID-2 membentuk jembatan teoretis antara RAID-1 dan RAID-3, di mana keduanya digunakan dalam dunia nyata.



Gambar 10.19 RAID-2 (*redundancy melalui kode Hamming*)

10.7.4 RAID-Level 3

RAID level 3 atau RAID-3 disusun mirip dengan RAID-2. Perbedaannya adalah RAID-3 hanya memerlukan sebuah redundant disk tunggal seberapa pun besarnya disk larik (array). RAID-3 menggunakan hanya satu drive untuk menyimpan *parity bit* sederhana seperti yang ditunjukkan pada Gambar 10.20. Kalkulasi parity dapat dilakukan dengan cepat dalam hardware menggunakan operasi exclusive OR (XOR, \oplus) pada setiap bit data (disimbolkan sebagai b_n) sebagai berikut:

$$\text{Parity} = b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7$$

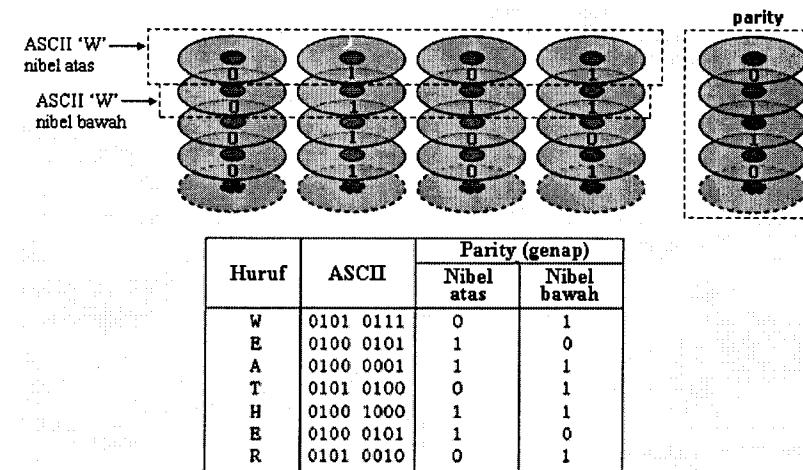
yang ekivalen dengan

$$\text{Parity} = b_0 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 \pmod{2}$$

Suatu drive yang gagal dapat direkonstruksi menggunakan kalkulasi yang sama. Misalnya, anggap drive nomor 6 gagal dan diganti. Data pada tujuh drive yang lain dan drive parity digunakan sebagai berikut:

$$b_6 = b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus \text{Parity} \oplus b_7$$

RAID-3 memerlukan duplikasi yang sama dan sinkronisasi seperti RAID-2, tetapi lebih ekonomis daripada RAID-1 atau RAID-2 karena dia hanya menggunakan drive untuk proteksi data. RAID-3 telah digunakan beberapa tahun pada sejumlah sistem komersial, tetapi tidak cocok untuk aplikasi-aplikasi *transaction-oriented*. RAID-3 lebih berguna untuk lingkungan di mana blok-blok besar dari data akan dibaca atau ditulis, seperti pada pemrosesan video atau citra.



Gambar 10.20 RAID-3, *Bit interleave data striping* dengan *parity disk*

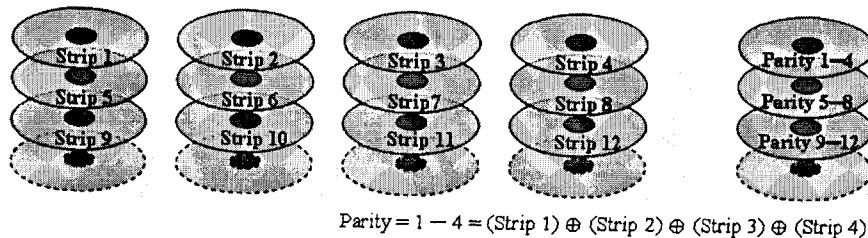
10.7.5 RAID-Level 4

Sebuah array RAID-4 seperti RAID-3, terdiri atas grup disk data dan disk parity. Sebagai pengganti penulisan data satu bit pada satu waktu pada semua drive, RAID-4 menulis data pada strip dengan ukuran yang sama, membuat sebuah strip pada semua drive seperti gambaran pada RAID-0. Bit-bit pada data strip di-XOR-kan satu sama lain untuk membuat parity strip.

Anda dapat memikirkan RAID-4 seperti RAID-0 dengan parity. Namun, penambahan hasil-hasil parity dalam sebuah penalti kinerja mendasar mengalami pertentangan dengan disk parity. Misalnya, anggap kita akan menulis ke Strip 3 dari sebuah stripe spanning lima drive (empat data, satu parity) seperti yang ditunjukkan pada Gambar 10.21. Pertama, kita harus membaca data yang sekarang menduduki Strip 3 serta parity strip. Data yang lama di-XOR-kan dengan data baru untuk memberikan parity baru. Data strip kemudian ditulis bersama dengan parity yang diperbarui (update).

Bayangkan apa yang terjadi jika ada permintaan penulisan yang menunggu sementara kita sedang memutar-mutar bit dalam blok parity, katakanlah satu permintaan penulisan untuk Strip 1 dan satu untuk Strip 4. Jika kita menggunakan RAID-0 atau RAID-1, kedua permintaan yang tertunda ini dapat terlayani bersamaan dengan menulis ke Strip 3. Jadi, drive parity menjadi macet (*bottleneck*), merampas sistem dari semua peningkatan kinerja potensial yang ditawarkan oleh sistem-sistem multidisk.

Beberapa penulis menyarankan bahwa kinerja RAID-4 dapat ditingkatkan jika ukuran stripe dioptimasi dengan ukuran record data yang akan ditulis. Lagi pula, ini dapat menjadi baik untuk aplikasi-aplikasi (seperti pemrosesan video atau voice) di mana data menempati record-record dengan ukuran yang sama. Namun, aplikasi-aplikasi basisdata pada umumnya melibatkan record-record berbagai ukuran, sehingga tidak mungkin memperoleh ukuran ‘optimum’ untuk suatu jumlah record substansial dalam basisdata. Karena kinerja yang diharapkan rendah, maka tidak ada implementasi komersial dari RAID-4.



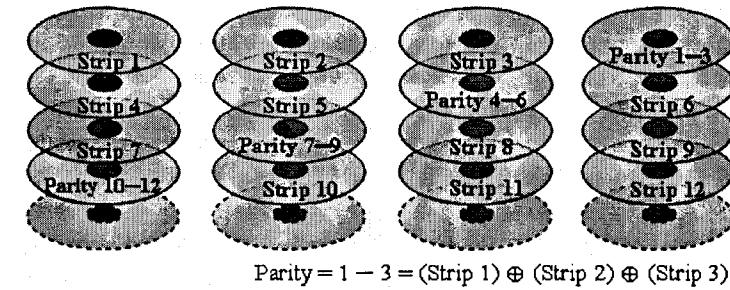
Gambar 10.21 RAID-4, Block interleave data striping dengan satu parity disk

10.7.6 RAID-Level 5

Sebagian besar setuju bahwa RAID-4 akan menawarkan proteksi yang cukup terhadap kegagalan disk tunggal. Kemacetan disebabkan oleh drive-drive parity, namun, membuat RAID-4 tidak cocok untuk digunakan pada lingkungan-lingkungan yang memerlukan throughput transaksi tinggi. Tentunya, throughput akan lebih baik jika kita dapat memengaruhi beberapa pengurutan penyeimbangan beban, menuliskan parity ke beberapa disk sebagai pengganti yang hanya satu. RAID-5 adalah RAID-4 dengan disk-disk parity yang menyebar melalui seluruh larik, seperti yang ditunjukkan pada Gambar 10.22.

Karena beberapa permintaan dapat dilayani secara bersamaan, RAID-5 menyediakan throughput pembacaan terbaik dari semua model-model parity dan memberikan throughput yang dapat diterima pada operasi-operasi penulisan. Misalnya, pada Gambar 10.22 array dapat melayani sebuah penulisan ke drive 4 Strip 6 secara bersamaan dengan sebuah penulisan drive 1 Strip 7 karena permintaan-permintaan ini melibatkan set-set lengan disk yang berbeda untuk parity dan data. Namun, RAID-5 memerlukan pengontrol disk yang sangat kompleks dari semua level yang ada.

Bandingkan dengan sistem-sistem RAID lainnya, RAID-5 menawarkan proteksi terbaik untuk harga termurah. Dengan ini menjadikan RAID-5 sukses secara komersial sampai saat ini. Aplikasi-aplikasi yang direkomendasikan termasuk file dan server-server aplikasi, email dan server-server pemberitaan, server-server basisdata dan server-server Web.



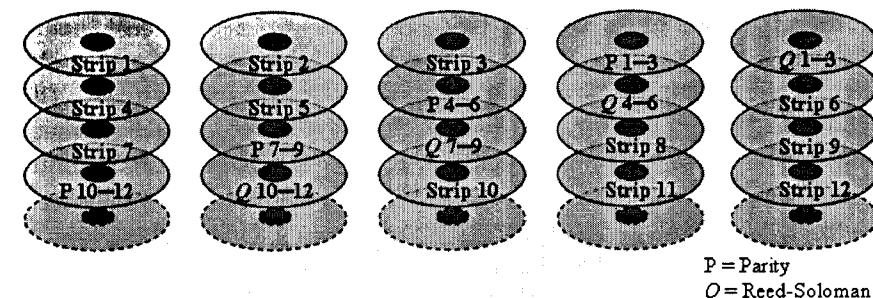
Gambar 10.22 RAID-5, Block interleave data striping dengan parity terdistribusi

10.7.7 RAID-Level 6

Pada umumnya sistem RAID hanya dibahas bahwa dapat tahan terhadap kegagalan satu disk pada satu waktu. Masalahnya adalah kegagalan drive pada sistem-sistem besar cenderung untuk masuk dalam cluster-cluster. Terdapat dua alasan untuk hal ini, pertama pabrik disk drive memperkirakan waktu yang sama mencapai akhir umur manfaat yang diharapkan kira-kira bersamaan. Jadi, jika Anda mengatakan bahwa disk drive baru kepunyaan Anda memiliki umur manfaat sekitar enam tahun, Anda dapat berharap masalah-masalah dalam enam tahun, kemungkinan terjadi kegagalan-kegagalan secara bersamaan. Alasan yang kedua, kegagalan disk drive seringkali disebabkan oleh kejadian yang merupakan bencana besar seperti sentakan power. Sebuah sentakan power menghantam semua drive pada waktu yang singkat, ini pelemahan pertama terjadi, diikuti dengan pelemahan-pelemahan berikutnya dan seterusnya. Kegagalan-kegagalan disk beruntun seperti ini dapat terjadi dalam searian atau mingguan. Jika mereka terjadi dalam MTTR (*mean time to repair*), termasuk *call time* dan *travel time*, disk yang kedua dapat gagal sebelum disk pertama diganti, dengan demikian salinan semua array tidak dapat diservis dan tidak berguna.

Sistem yang memerlukan ketersediaan (*availability*) tinggi harus mampu tahan terhadap lebih dari satu kegagalan drive secara bersamaan, khususnya jika MTTR adalah sebuah bilangan yang besar. Jika sebuah array dirancang untuk tahan terhadap kegagalan bersama dari dua drive, diperoleh MTTR efektif dua kali lipat. RAID-1 menawarkan ketahanan jenis ini; kenyataannya selama sebuah disk dan mirror-nya keduanya tidak terhapus, array RAID-1 dapat bertahan kehilangan separuh dari disk-nya.

RAID-6 menyediakan suatu jawaban ekonomi terhadap masalah kegagalan multidisk. Hal ini dilakukan dengan menggunakan dua set strip *error-correction* untuk setiap *rank* (atau baris horizontal) drive. Sebuah level proteksi kedua ditambahkan dengan menggunakan kode-kode *Reed-Soloman error-correction* di samping parity. Mempunyai dua *error-detecting strip per stripe* meningkatkan biaya penyimpanan. Jika *unprotected data* dapat disimpan pada N drive, penambahan proteksi RAID-6 memerlukan $N + 2$ drive. Karena parity dua dimensi, RAID-6 menawarkan kinerja penulisan yang sangat buruk. Konfigurasi RAID-6 ditunjukkan pada Gambar 10.23.

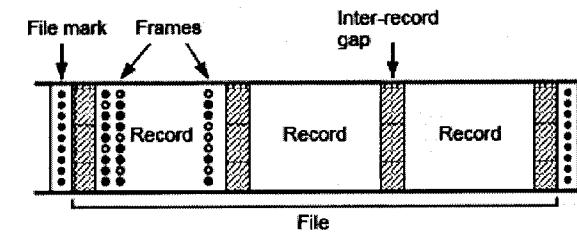


Gambar 10.23 Block interleave data striping dengan proteksi dual error

Hingga saat ini, tidak ada penyebaran komersial RAID-6. Ada dua alasan. Pertama, terdapat sebuah *sizeable overhead penalty* yang dilibatkan dalam pembangkitan kode Reed-Soloman. Kedua, operasi baca/tulis sebanyak dua kali diperlukan untuk memperbarui kode-kode *error-correcting* yang tersimpan pada disk. IBM yang pertama yang membawa RAID-6 ke pasaran dengan RAMAC RVA 2 *turbo disk array*-nya.

10.8 PITA MAGNETIK

Pita magnetik biasanya (*typical*) mempunyai head baca/tulis tunggal di mana head untuk pembacaan dan penulisan dipisahkan. Bila dilakukan penulisan, lilitan pita plastik (*mylar*) dengan selimut magnetik yang melewati head akan memagnetisasi pita tersebut. Dan jika dilakukan pembacaan, maka data yang di-sense (terdeteksi) akan disimpan. Pita magnetik lebih murah untuk penyimpanan data yang besar, tetapi lambat karena pada waktu mengakses bagian pita tertentu, head harus melewati bagian-bagian pita lain yang ada sebelumnya secara serial.



Gambar 10.24 Bagian-bagian pita magnetik

Informasi yang disimpan pada pita dilakukan dengan dua-dimensi seperti yang diilustrasikan pada Gambar 10.24. Bit-bit disimpan pada selebar pita dalam suatu frame dan sepanjang pita dalam bentuk record-record. File dibuat dengan kumpulan record-record. Record adalah data yang terkecil yang dapat dibaca dari atau ditulis ke pita. Pita magnetik cocok untuk penyimpanan data dalam jumlah besar seperti backup dari disk atau citra hasil pindai, tetapi tidak cocok untuk pembacan dan penulisan akses acak.

10.9 DISK OPTIK

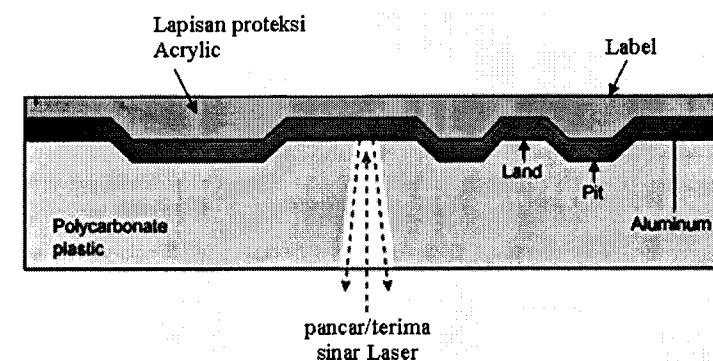
Beberapa teknologi baru memberikan keuntungan pada optik sebagai penyimpanan dan pengambilan kembali data. Pada bahasan ini akan dikemukakan *Compact Disc* (CD) dan *Digital Versatile Disc* (DVD) yang menggunakan cahaya laser untuk membaca data yang terkode pada permukaan reflektif.

10.9.1 Compact Disc

Compact disc pertama kali ditemukan pada tahun 1983 sebagai media untuk memainkan musik. *Compact disc* mempunyai kapasitas penyimpanan 74 menit audio dalam format stereo digital (2-kanal). Audio dicuplik (*sampling*) pada 2×44000 16-bit cuplikan per detik atau dengan kapasitas yang setara sekitar 700 MB. Karena ditemukan pada 1983, maka teknologi ini telah mengalami perkembangan dalam hal kerapatan, keandalan dan harga, yang menuju pada pengembangan CD ROM (*CD read only memory*) untuk komputer yang juga mempunyai kapasitas sama 700 MB. Harga yang cukup rendah ditambah dengan keandalan yang baik dan kapasitas yang tinggi membuat media CD ROM sebagai salah satu pilihan untuk pendistribusian software komersial menggantikan floppy disk.

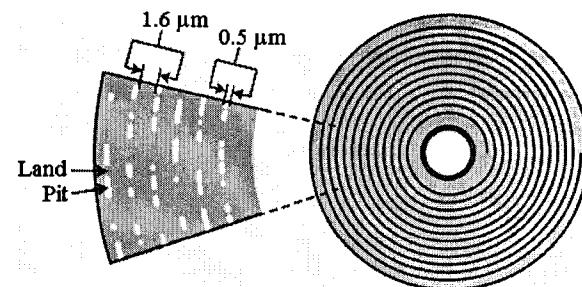
CD ROM terdiri atas aluminium yang dibungkus plastik, atau bahan dari resin seperti polycarbonate yang memantulkan cahaya berbeda pada *land* (tonjolan) dan *pit* (cekungan). Informasi direkam secara digital (musik atau data komputer) yang dicetak seperti rangkaian pit mikroskopik pada permukaan polycarbonate. Hal ini dilakukan sekaligus dan dengan laser intensitas tinggi yang difokuskan dengan baik akan menghasilkan disk master. Selanjutnya master digunakan untuk membuat salinan yang dicap di atas polycarbonate. Permukaan pit kemudian dibungkus dengan permukaan yang mempunyai sifat pantulan tinggi, biasanya dari aluminium

atau emas. Permukaan yang bening/terang ini kemudian dibalut dengan selimut akhir dari bahan acrylic bening untuk melindungi dari debu dan goresan. Terakhir label dapat dibuat di atas permukaan acrylic.



Gambar 10.25 Prinsip operasi compact disc (CD)

Informasi yang dibaca dari suatu CD atau CD ROM menggunakan sinar laser daya rendah yang terdapat dalam *optical-disc player* atau *drive unit*. Pada Gambar 10.25 diilustrasikan sinar laser akan melalui polycarbonate ketika motor memutar disk melewatiinya. Intensitas cahaya pantulan dari laser berubah begitu mengenai pit. Secara khusus, jika kerak laser jatuh di atas pit yang mempunyai permukaan agak kasar, cahaya berhambur dan terpantul kembali ke sumber dengan intensitas yang rendah. Daerah di antara pit disebut land. Land merupakan permukaan yang halus yang memantulkan kembali dengan intensitas lebih tinggi. Perubahan antara pit dan land dideteksi oleh sebuah photosensor dan diubah menjadi sinyal digital. Sensor akan menguji permukaan dengan interval teratur. Awal atau akhir sebuah pit direpresentasikan sebagai 1; bila tidak terjadi perubahan elevasi antara interval maka akan direkam sebagai 0.



Gambar 10.26 Format penyimpanan spiral pada CD

Informasi dalam track CD direkam dalam format spiral (menggunakan *constant linear velocity*, CLV) yang berbeda dengan disk magnetik yang direkam dalam track konsentrik dengan rotasi disk menggunakan *constant angular velocity* (CAV), artinya jumlah bit per track tetap. CD berisi track berbentuk spiral yang berawal dari pusat dan membentuk spiral keluar mendekati tepi disk. Sektor yang dekat dengan sisi luar disk sama panjangnya yang dekat dengan sisi dalam. Pit mempunyai spasi yang sama antara ujung satu dengan lainnya pada spiral tersebut (Gambar 10.26).

Kecepatan rotasi pada awalnya 30 rpm yang sama dengan floppy disk yang diatur agar disk bergerak lebih lambat ketika head berada di tepi dibandingkan bila berada di pusat. Jadi CD ROM memiliki rugi waktu akses yang lama seperti floppy disk karena rotational latency yang tinggi. Teknologi CD ROM sangat tepat untuk pendistribusian data dalam jumlah besar yang murah bila dibuat dalam jumlah copy yang banyak, karena biaya pembuatan master dan pembuatan copy tersebar/ter tutupi pada biaya copy yang murah. Jika hanya beberapa copy saja yang dibuat, maka harga setiap disk menjadi tinggi karena CD tidak ekonomis jika diproduksi dalam jumlah sedikit. CD juga tidak dapat ditulisi setelah diproduksi. Teknologi baru yang mengatasi masalah ini adalah disk optik WORM (*write once read many*), di mana dengan intensitas laser yang rendah dalam CD controller dapat menulis disk optik (tetapi hanya sekali untuk setiap lokasi bit). Proses penulisan biasanya lebih lambat dari proses pembacaan, dan controller serta media lebih mahal dibandingkan pada CD ROM.

10.9.2 Digital Versatile Disc

Media penyimpanan disk optik yang terbaru adalah *Digital Versatile Disc* (DVD) yang dikenal juga dengan *Digital Video Disk* dikembangkan tahun 1997 untuk keperluan industri movie dan data komputer. DVD menggunakan diameter piringan yang sama dengan CD yaitu 120mm/4,75 inch dan ada juga (jarang) 80mm (3,15 inch). DVD tersedia dalam dua versi yaitu sisi tunggal dan sisi ganda dengan satu-layer atau dual-layer informasi per sisi. DVD sisi tunggal telah menjadi media standar untuk perekaman gambar bergerak, menggantikan videotape di pasaran. Sisi ganda, versi *dual-layer* dapat menyimpan informasi sekitar 30 kali lebih besar dari CD standar. DVD dibuat dalam format sebuah ROM serta dapat dihapus (DVD-E) dan yang dapat ditulisi sekali (DVD-R). Ada standar industri untuk DVD-Audio, DVD-Video dan DVD-ROM serta penyimpan data DVD-RAM. Bila digunakan DVD sisi tunggal, kapasitas penyimpanannya dapat mencapai hingga 4,7 GB. DVD standar juga memasukkan kemampuan penyimpanan data pada dua sisi dalam dua layer setiap sisi, kapasitas totalnya 17 GB. Teknologi DVD berkembang dari CD, dan dalam kenyataannya *DVD player* (pemutar DVD) pada dasarnya *backward compatible*—pemutar DVD dapat digunakan untuk memainkan CD dan CD-ROM sama baiknya dengan DVD.

Ukuran pit-pit DVD kira-kira setengah dari ukuran pit CD (0.4 sampai 2.13 micron) dan track pitch-nya 0.74 micron.

10.10 MODEM

Kata modem merupakan singkatan dari MODulator DEModulator. Sebuah perangkat input/output modern melakukan link sebuah komputer ke saluran telepon untuk dapat berkomunikasi dengan komputer lainnya. Sistem telepon mentransmisikan suara dan data sebagai sinyal-sinyal analog.

Bila sebuah komputer mengirim data ke komputer lain, modem memberikan data digital dari komputer, memodulasinya (mentransformasi) menjadi tegangan-tegangan analog yang dapat ditransmisikan melalui saluran telepon. Pada sisi penerima, modem yang lain mengubah tegangan-tegangan analog tersebut menjadi data digital. Pertumbuhan internet menyebabkan banyaknya pemakaian modem yang melintasi satu komputer ke komputer lainnya.

10.11 SCANNER

Scanner adalah suatu perangkat input yang mengubah gambar dan teks menjadi sebuah data stream. Ada empat jenis scanner yaitu:

4. *Drum Scanner*
5. *Flatbed Scanner*
6. *Sheetfed Scanner*
7. *Hand-held Scanner*

Drum scanner. Adalah produk yang mempunyai kualitas dan harga yang tinggi, sedangkan hand-held merupakan kebalikan yang ekstrem. Sebuah drum scanner menggunakan sebuah tabung photo-multiplier yang merupakan perangkat pengindra cahaya. Drum scanner menawarkan sensitivitas tinggi dan signal-to-noise ratio yang tinggi. Citra yang di-scan ditangani oleh sebuah drum berputar.

Flatbed scanner. Material yang akan di-scan (gambar atau teks) diletakkan pada permukaan plat glass di dalam scanner. Sebuah sumber cahaya dan sebuah CCD (*charge coupled device*) ditancapkan pada suatu pembawa yang bergerak. CCD memungkinkan konversi cahaya (dan bayangan cahaya) menjadi pulsa-pulsa listrik. Scanning head terdiri atas CCD yang bergerak melintasi gambar-gambar.

Sheetfed scanner. Pada sheetfed scanner kertas ditarik pada scan head sebagai pengganti head yang dapat bergerak pada halaman. Ukurannya yang kecil merupakan keuntungan, tetapi mekanisme yang kurang tepat dapat mengakibatkan kertas yang di-scan miring.

Hand-held scanner. Merupakan scanner yang dapat digenggam dan digerakkan pada dokumen dengan manual. Harga yang murah dan portabilitas merupakan keuntungan, tetapi kualitasnya rendah.

10.12 PLOTTER

Plotter adalah suatu perangkat output grafik yang digunakan untuk membuat gambar pada kertas. Ada dua jenis plotter yaitu *pen plotter* dan *photo plotter*. Pen plotter adalah sebuah perangkat elektromekanik. Sebuah pena digerakkan pada dua dimensi (naik/turun dan kiri/kanan) melintasi media kertas atau film. Pena-pena bertinta biasanya digunakan dalam plotter. Pen plotter mempunyai banyak jenis: *drum plotter*, *microgrip*

plotter, *flatbed plotter*. Pada drum plotter, drum (silinder panjang) berputar. Sebuah pena bergerak secara horizontal. Pada micogrip plotter, medium dipegang pada bagian pinggir dan digerakkan balik dan seterusnya. Pada flatbed plotter, pena digerakkan sepanjang sumbu X dan Y. Untuk mengontrol pergerakan pada langkah digital pendek, maka digunakan motor stepper. Photo plotter menggunakan teknologi serat optik untuk menghasilkan citra pada kertas silver kering.

Plotter tersedia dalam berbagai ukuran mulai A4, A3 dan seterusnya. Walaupun plotter mahal, mereka menyediakan kualitas gambar yang sangat baik. Gambar-gambar teknik dapat disiapkan lebih cepat dan indah dengan plotter. Ada dua metode antarmuka yang dapat dilakukan pada plotter yaitu dengan antarmuka serial dan antarmuka paralel.

RINGKASAN

Perangkat-perangkat peripheral membolehkan interaksi antara pengguna dan komputer. Keyboard adalah sebuah perangkat input. Bila sebuah tombol ditekan, maka elektronika keyboard mentransmisikan sebuah kode ke CPU. Elektronika keyboard juga menangani masalah peralihan (*roll-over*) tombol-tombol dan masalah pelentingan tombol (*switch bouncing*). Mouse adalah perangkat penunjuk yang mendukung antarmuka pengguna grafis. Monitor CRT adalah sebuah perangkat output yang menampilkan informasi video. Sebuah berkas elektron menghantam layar CRT yang menyebabkan iluminasi. Proses scanning melibatkan pantulan balik beam yang melintasi layar yang membuat citra permanen.

Printer merupakan perangkat output populer yang menyediakan sebuah keluaran hard copy yang dapat terlihat mata. Dia menerima karakter-karakter dari komputer dan mencetaknya pada kertas. Printer dot matrix, printer laser, dan printer inkjet merupakan jenis-jenis printer yang populer. Printer laser dan inkjet adalah printer yang tidak melakukan hantaman langsung pada kertas (*non-impact printer*) sedangkan printer dot matrix merupakan impact printer. Pada printer inkjet, pencetakan dilakukan dengan penyemprotan tinta secara beruntun pada sebuah titik untuk membentuk sebuah citra.

Floppy disk drive dan *hard disk drive* adalah perangkat penyimpanan magnetik dengan melakukan pemutaran disk. Hard disk menggunakan sebuah flat keras sedangkan pada floppy disk menggunakan sebuah flexible diskette. Floppy head menyentuh media ketika melakukan operasi baca/tulis, tetapi hard disk head tetap di atas permukaan disk. Mekanisme penempatan head pada sebuah hard disk rumit karena melibatkan mekanisme servo. Floppy disk drive umumnya menggunakan mekanisme motor stepper. *Compact Disk (CD)* menggunakan teknologi optik. Berkas laser digunakan untuk penyimpanan informasi dalam sebuah track berbentuk spiral pada CD. CD-ROM adalah media penyimpanan read-only. CD-R membolehkan satu kali perekaman (penulisan) dan CD-RW membolehkan operasi baca/tulis yang mirip dengan floppy.

Sebuah modem berfungsi untuk mendukung komunikasi jarak jauh. Scanner, Plotter dan Light pen merupakan peripheral khusus yang digunakan dalam aplikasi yang berbeda.

SOAL-SOAL ULANGAN

1. Ada tiga fungsi utama yang dikerjakan oleh keyboard elektronik meliputi: (1) pengindraan tombol yang ditekan, (2) *encoding*, dan (3) _____.
2. Keyboard mengikuti teknik standar pengenalan karakter yang dikenal dengan _____.
3. Ada beberapa jenis tombol keyboard, misalnya: (1) mechanical keyswitch (2) *membrane keyswitch*, dan (3) _____.
4. Keyboard yang umum digunakan adalah yang mempunyai tata letak tombol-tombol berdasarkan standar _____.
5. Kode 7-bit yang umum digunakan sebagai kode teks adalah ASCII yang merupakan singkatan dari _____.
6. Tampilan/layar video yang menggunakan teknologi tabung hampa dinamakan _____.
7. Tampilan/layar video yang menggunakan teknologi kristal cair dinamakan _____.
8. Ada dua kelompok printer yaitu (1) *impact printer* dan (2) _____.
9. Jenis printer yang mencetak karakter berdasarkan kontak fisik (atau tekanan) antara head (hammer, pin atau font) dengan pita tinta ke atas kertas, disebut jenis _____.
10. Contoh printer untuk jenis impact printer yaitu: (1) printer dot matrix, (2) printer _____.
11. Printer laser merupakan printer jenis _____.
12. Bagian dari hard disk drive (disk magnetik) yang berfungsi untuk membaca/menulis data disebut _____.
13. Hard disk drive yang sifatnya dapat langsung dilepaskan jika tidak digunakan disebut _____.
14. Pada dasarnya dikenal dua posisi head yaitu (1) fixed head, dan (2) _____.
15. Kerugian fixed head karena pada setiap _____ harus terdapat satu head.
16. Permukaan disk dibagi dalam lingkaran-lingkaran konsentrik (lingkaran dalam lingkaran) yang disebut _____.
17. Kumpulan semua track yang berada dalam posisi yang relatif sama/sejajar pada susunan piringan disebut _____.
18. Track dibagi menjadi bagian-bagian yang disebut _____, yang berbentuk seperti kue pie.

19. Antara satu track dengan track di sebelahnya dipisahkan dengan sebuah _____. Demikian halnya pada sektor.
20. Ukuran fisik sektor semakin mendekati pusat cakram maka semakin _____.
21. Bit yang dekat dengan pusat disk yang berputar melewati sebuah titik tetap (seperti head baca/tulis) kecepatannya lebih _____ daripada bit terluar.
22. Organisasi sektor pada disk dikenal ada dua yaitu: (1) *constant angular velocity*, dan (2) _____.
23. Parameter-parameter yang menentukan kinerja suatu disk meliputi (1) *seek time*, (2) *rotational delay*, (3) *access time*, dan (4) _____.
24. Pada sistem *movable head* waktu yang dibutuhkan oleh pergerakan head untuk mencari track yang sesuai disebut _____.
25. Waktu yang dibutuhkan oleh pergerakan putaran piringan untuk mencapai sektor yang sesuai disebut _____.
26. Jumlah/total waktu antara seek time jika ada dan rotational delay disebut dengan _____ yang merupakan waktu untuk mulai melakukan pembacaan atau penulisan.
27. Ketika head telah berada pada posisi yang sesuai, maka operasi baca atau tulis pada sektor dilakukan, dan ketika bagian data diperoleh dan ditransfer ke head disebut sebagai _____.
28. Pada tahun 1988, David Patterson, Garth Gibson dan Randy Katz dari Universitas California di Barkeley mengusulkan sistem penyimpanan yang berbasis banyak disk, yang disebut _____.
29. Tidak seperti level RAID lainnya, RAID level _____ tidak menawarkan redundancy sehingga RAID ini memberikan kinerja yang paling baik khususnya jika kontrolernya dipisahkan dan digunakan cache untuk setiap disk.
30. RAID level _____ (dikenal juga dengan *disk mirroring*) memberikan proteksi kegagalan yang terbaik dari semua skema RAID.
31. Pita magnetik merupakan media penyimpanan yang relatif lambat dibandingkan dengan disk magnetik karena pengaksesannya dilakukan secara _____.
32. Disk optik menggunakan _____ untuk membaca data yang terkode pada permukaan reflektif.
33. CD ROM terdiri atas aluminium yang dibungkus plastik, atau bahan dari resin seperti polycarbonate yang memantulkan cahaya berbeda pada (1) *land* (tonjolan), dan (2) _____.

34. Untuk melakukan link dari sebuah komputer ke saluran telepon maka dibutuhkan perangkat yang melakukan proses modulasi dan demodulasi yang lazim disebut _____.
35. Scanner adalah suatu perangkat input yang mengubah gambar dan teks menjadi sebuah data stream. Ada empat jenis scanner yaitu: (1) *drum scanner*, (2) *flatbed scanner*, (3) *sheetfed scanner*, dan (4) _____.

SOAL-SOAL LATIHAN

1. Sebuah tampilan *bit mapped* lebar 1024 pixel dengan tinggi 1024 pixel. Kecepatan refresh 60 Hz, yang berarti setiap pixel ditulisi kembali ke layar 60 kali dalam satu detik, tetapi hanya satu pixel setiap saat. Berapakah waktu maksimum yang diizinkan untuk menulisi sebuah pixel tunggal?
2. Anggap sebuah disk drive mempunyai karakteristik berikut:
 - 4 permukaan
 - 1024 track per permukaan
 - 128 sektor per track
 - 512 byte per sektor
 - Seek time dari track ke track 5 ms
 - Kecepatan rotasi 5000 RPM
 a) Berapakah kapasitas disk?
 b) Berapakah waktu akses (*access time*)?
3. Anggap sebuah disk drive mempunyai karakteristik berikut:
 - 5 permukaan
 - 1024 track per permukaan
 - 256 sektor per track
 - 512 byte per sektor
 - Seek time dari track ke track 8 ms
 - Kecepatan rotasi 7500 RPM
 a) Berapakah kapasitas disk?
 b) Berapakah waktu akses (*access time*)?
 c) Manakah yang lebih cepat dibandingkan waktu akses pada soal No.2b?
4. Sebuah hard disk mempunyai dua permukaan. Area penyimpanan pada setiap permukaan mempunyai radius terdalam 1 cm dan radius terluar 5 cm. Setiap track menyimpan bit dengan jumlah yang sama, walaupun

setiap track berbeda ukurannya satu sama lain. Kepadatan (density) penyimpanan maksimum media adalah 10000 bit/cm. Jarak spasi antara titik-titik yang berhubungan pada track yang berdekatan adalah 0.1 mm, termasuk pemisah (gap) antar-track. Anggap pemisah antar-track diabaikan, dan bahwa sebuah track berada pada masing-masing tepi area penyimpan.

- a) Berapakah jumlah bit maksimum yang dapat disimpan pada disk?
- b) Berapakah kecepatan transfer data dari disk ke head dalam bit per detik?
- 5. Sebuah disk mempunyai 128 track dengan masing-masing 32 sektor, pada setiap permukaan delapan platter. Disk berputar pada 3600 RPM, dan menghabiskan waktu 15 ms untuk bergerak antar-track yang berdekatan. Berapakah waktu terpanjang yang dibutuhkan untuk membaca sebuah sektor yang lokasinya di dalam disk tidak tetap (berubah-ubah)?

BAB 11

PIPELINING DAN PEMROSESAN VEKTOR

Sasaran bab ini:

1. Tumpang Tindih dan Paralelisme
2. Pipelining
3. Pemrosesan Larik
4. Pemrosesan Vektor

Pencapaian kinerja tinggi suatu komputer merupakan salah satu tujuan utama dari seorang arsitek komputer. Implementasi konkurensi dalam organisasi sebuah komputer meningkatkan kinerjanya seperti halnya pada multioperasi yang dilakukan secara simultan. Desain unit kontrol menjadi kompleks karena beberapa datapath beroperasi secara simultan. Penyediaan teknik tumpang-tindih (*overlap*) dan paralelisme dalam komputer merupakan suatu teknik standar dalam meningkatkan kinerja komputer. Teknik lain yang digunakan adalah multiprosesor dalam sebuah sistem komputer. Pada bab ini akan dibahas konkurensi pada uniprosesor kecuali arsitektur superscalar. Dibahas juga secara detail tentang konsep *pipelining* dan yang berkaitan dengan masalah-masalah desain. Di samping itu, disinggung pula tentang pemrosesan vektor dan pemrosesan larik.

11.1 STRATEGI PENINGKATAN KINERJA

Kinerja komputer diukur dari waktu yang digunakan dalam mengeksekusi program. Eksekusi program melibatkan pelaksanaan siklus instruksi, termasuk dua macam aktivitas:

1. *Mikrooperasi-mikrooperasi internal* dilakukan dalam unit-unit fungsional hardware seperti prosesor, memori, pengontrol I/O, dan sebagainya.
2. *Transfer informasi* antara unit-unit fungsional hardware untuk pengambilan instruksi, pengambilan operand, operasi I/O, dan sebagainya.

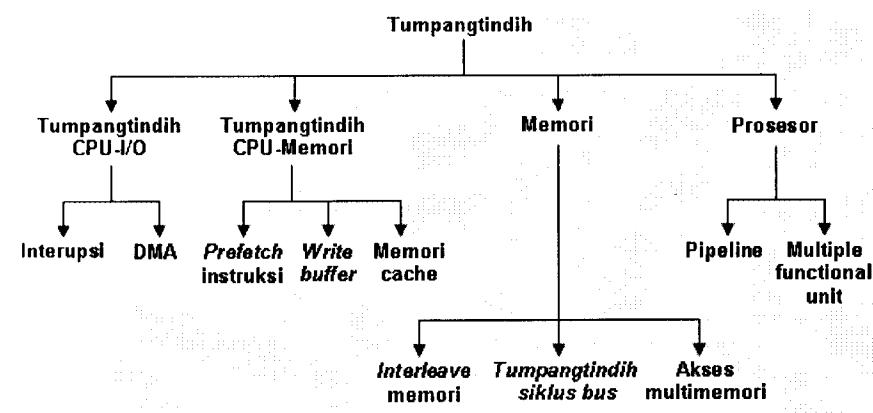
Kinerja optimal komputer bergantung pada jenis hardware yang digunakan. Untuk mengatasi kendala-kendala sistem, digunakan beberapa strategi peningkatan kinerja. Tujuannya di sini adalah meningkatkan jumlah operasi yang dieksekusi dalam suatu waktu yang diberikan. Ada dua strategi dasar yang digunakan untuk meningkatkan kinerja komputer:

1. *Tumpang tindih (overlap)*: Memecah tugas menjadi multi-tugas yang dapat dilaksanakan dengan cara tumpang tindih.
2. *Paralelisme*: Pengeksekusian lebih dari satu tugas secara paralel.

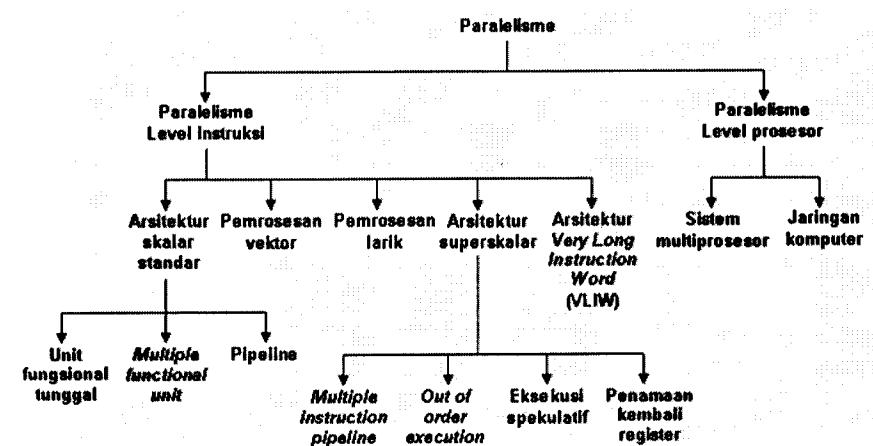
Strategi tumpang tindih menggunakan operasi-operasi konkuren melalui unit-unit fungsional heterogen, sedangkan paralelisme menggunakan operasi-operasi konkuren melalui unit-unit fungsional homogen.

Gambar 11.1 menunjukkan teknik-teknik tumpang tindih yang biasa digunakan, sedangkan pada Gambar 11.2 menunjukkan teknik-teknik paralelisme. Secara garis besar disebutkan ada dua level paralelisme:

1. Paralelisme level instruksi
2. Paralelisme level prosesor



Gambar 11.1 Teknik-teknik tumpang tindih



Gambar 11.2 Teknik-teknik paralelisme

Pada paralelisme level instruksi, paralelisme digunakan dalam sebuah prosesor tunggal. Arsitektur tradisional menggunakan berbagai teknik untuk mengekstrak paralelisme level instruksi dari aliran instruksi sebuah program aplikasi dan meningkatkan kinerja. Arsitektur vektor dan arsitektur larik (array) pendekatannya berbeda dan memanfaatkan keuntungan paralelisme data. Pada paralelisme level prosesor, terdapat beberapa prosesor yang melakukan share pada beban kerja (tugas-tugas). Hal ini

dapat dicapai dengan menggunakan lebih dari satu prosesor dalam sistem komputer tunggal (sistem multiprosesor) atau pendistribusian tugas-tugas di antara sejumlah sistem komputer yang di-link seperti cluster atau jaringan.

11.2 KLASIFIKASI PARALELISME (KLASIFIKASI FLYNN)

Pemrosesan paralel adalah istilah yang digunakan untuk menunjukkan suatu teknik tingkat tinggi yang digunakan untuk menyediakan tugas-tugas pemrosesan data secara simultan/bersama yang bertujuan meningkatkan kecepatan komputasi dari suatu sistem komputer. Teknik ini mengantikan pemrosesan yang dilakukan pada komputer konvensional di mana pemrosesan setiap instruksi dilakukan secara sekvensial. Sistem pemrosesan paralel dapat melakukan pemrosesan data secara bersama untuk memperoleh waktu eksekusi yang lebih cepat. Misalnya, ketika suatu instruksi sedang dieksekusi oleh ALU, maka instruksi berikutnya dapat dibaca dari memori. Sistem bisa mempunyai dua atau lebih ALU dan dapat mengeksekusi dua atau lebih instruksi pada saat yang sama. Selanjutnya, sistem dapat mempunyai dua atau lebih prosesor yang bekerja secara bersama. Tujuan pemrosesan paralel adalah untuk meningkatkan kemampuan kecepatan pemrosesan komputer dan meningkatkan throughput komputer, jadi sejumlah pemrosesan tersebut dapat dimanfaatkan ketika terjadi interval waktu (kekosongan). Jumlah hardware bertambah pada pemrosesan paralel, sehingga biaya sistem meningkat. Namun, perkembangan teknologi mampu mengurangi biaya hardware tersebut di mana teknik-teknik pemrosesan paralel masih layak secara ekonomi.

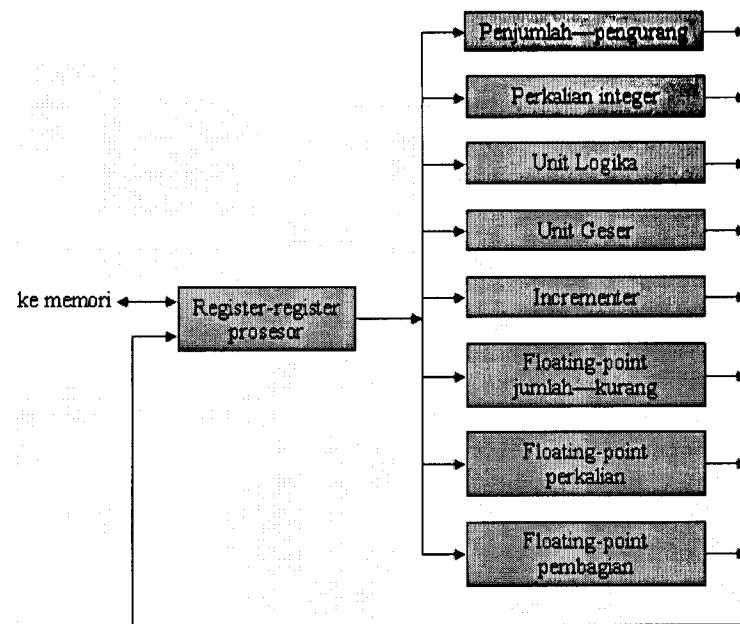
Pemrosesan paralel dapat ditinjau dari berbagai level kompleksitas. Pada level terendah, kita membedakan antara operasi paralel dan operasi serial berdasarkan penggunaan jenis register. Register geser bekerja dengan cara serial satu bit dalam satu waktu, sementara register paralel bekerja di mana semua bit diproses secara simultan. Pemrosesan paralel pada tingkat kompleksitas tinggi dapat dicapai dengan mempunyai unit fungsional perkalian yang melakukan operasi yang sama atau operasi yang berbeda secara simultan. Pemrosesan paralel dibangun dengan pendistribusian data antara unit-unit fungsional jamak. Misalnya, operasi aritmetika, logika, dan

geser dapat dipisah menjadi tiga unit dan mengubah tujuan operand-operand ke setiap unit di bawah pengawasan unit kontrol.

Pada Gambar 11.3 ditunjukkan suatu cara yang memungkinkan pemisahan unit eksekusi menjadi delapan unit fungsional yang bekerja secara paralel. Operand-operand dalam register diberikan ke salah satu unit bergantung pada operasi yang ditetapkan oleh instruksi yang sesuai dengan operand. Operasi dilakukan pada setiap unit fungsional ditunjukkan pada masing-masing blok dari diagram. Penjumlah dan pengali integer melakukan operasi aritmetika dengan bilangan integer. Operasi floating-point dipisahkan dalam tiga sirkuit yang bekerja secara paralel. Operasi-operasi logika, geser dan inkrement dapat dilakukan secara bersama pada data yang berbeda. Semua unit terpisah satu dengan yang lain, sehingga satu bilangan dapat digeser sementara bilangan lainnya sedang diinkrement. Organisasi multifungsional umumnya berhubungan dengan unit kontrol yang kompleks untuk mengoordinasi semua aktivitas di antara berbagai komponen.

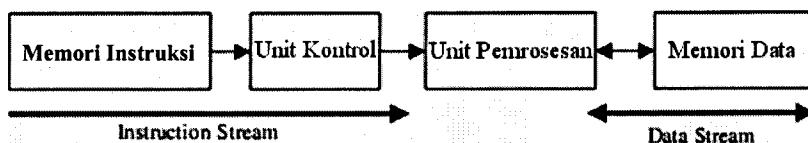
Klasifikasi pemrosesan paralel dibagi dalam beberapa cara. Dapat ditinjau dari organisasi internal prosesor, dari struktur interkoneksi antarprosesor, atau dari aliran informasi melalui sistem. Satu klasifikasi yang diperkenalkan oleh M.J. Flynn memandang organisasi sebuah sistem komputer pada jumlah instruksi dan data yang dimanipulasi secara simultan. Operasi normal suatu komputer adalah mengambil instruksi dari memori dan mengeksekusinya di dalam prosesor. Urutan instruksi dibaca dari memori yang merupakan bagian dari suatu *instruction stream*. Operasi-operasi dilakukan pada data dalam prosesor merupakan bagian dari suatu *data stream*. Pemrosesan paralel dapat terjadi dalam *instruction stream*, dalam *data stream*, atau keduanya. Klasifikasi Flynn membagi komputer menjadi empat kelompok utama:

1. *Single instruction stream, single data stream* (SISD)
2. *Single instruction stream, multiple data stream* (SIMD)
3. *Multiple instruction stream, single data stream* (MISD)
4. *Multiple instruction stream, multiple data stream* (MIMD)



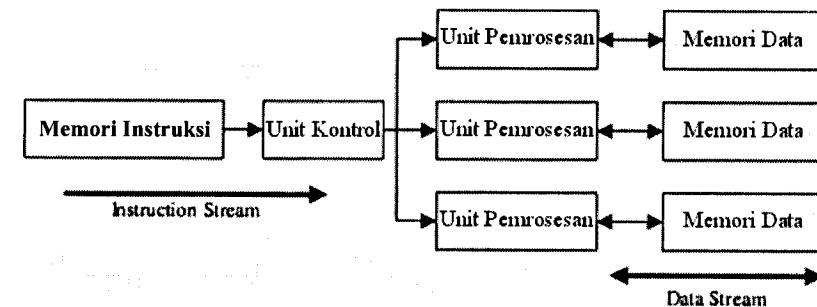
Gambar 11.3 Prosesor dengan multi-unit fungisional

Pemrosesan SISD (Gambar 11.4) merepresentasikan organisasi sebuah komputer tunggal yang mempunyai sebuah unit kontrol, sebuah unit pemrosesan, dan sebuah unit memori. Instruksi dieksekusi secara sekuensial dan sistem dapat atau tidak mempunyai kemampuan pemrosesan paralel internal. Pemrosesan paralel dalam kasus ini dapat dilakukan dengan menggunakan unit-unit multifungsional atau dengan pemrosesan pipeline.



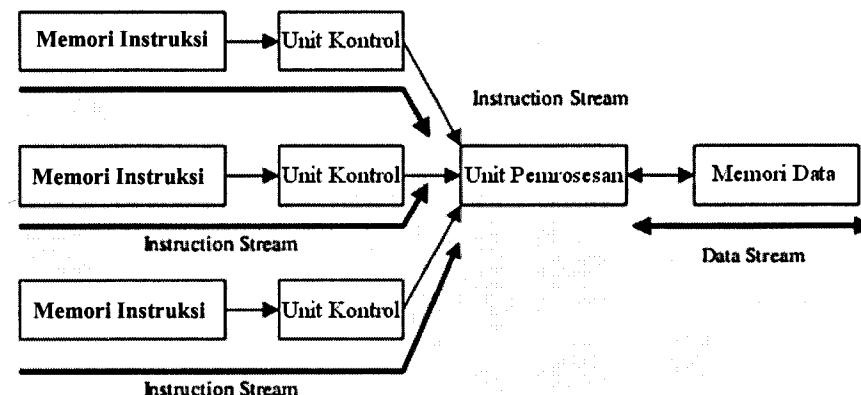
Gambar 11.4 Pemrosesan SISD

Pemrosesan SIMD (Gambar 11.5) merepresentasikan sebuah organisasi yang melibatkan banyak unit-unit pemrosesan di bawah pengawasan dari suatu unit kontrol bersama. Semua proses menerima instruksi yang sama dari unit kontrol tetapi bekerja pada item data yang berbeda. Unit memori berisi modul-modul jamak agar dapat berkomunikasi dengan semua proses secara simultan. Pemrosesan MISD (Gambar 11.6) hanya merupakan suatu struktur secara teoretis karena belum ada sistem praktis yang dibangun menggunakan organisasi ini. Organisasi MIMD (Gambar 11.7) merujuk pada suatu sistem komputer yang mampu memproses beberapa program pada waktu yang sama. Umumnya multiprosesor dan sistem multikomputer dapat diklasifikasikan dalam kategori ini.

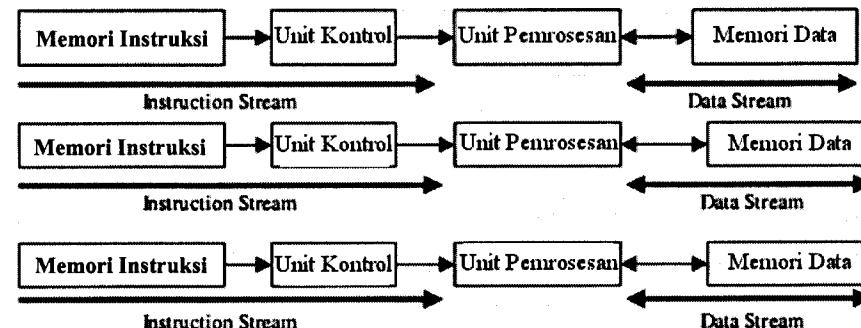


Gambar 11.5 Pemrosesan SIMD

Klasifikasi Flynn bergantung pada perbedaan antara kinerja unit kontrol dan unit pemrosesan data. Dia menekankan pada perilaku karakteristik sistem komputer daripada operasionalnya dan interkoneksi-interkoneksi struktural. Salah satu jenis pemrosesan paralel yang tidak sesuai dengan klasifikasi Flynn adalah pipelining. Hanya dua kategori yang digunakan dari klasifikasi ini yaitu prosesor-prosesor larik SIMD dan multiprosesor MIMD.



Gambar 11.6 Pemrosesan MISD



Gambar 11.7 Pemrosesan MIMD

11.2.1 Prosesor Skalar, Vektor, Superskalar, dan Prosesor yang di-Pipeline

Sebuah prosesor skalar sederhana hanya melakukan satu operasi aritmetika dalam satu waktu. Dia mengeksekusi satu instruksi pada waktu yang diberikan dan setiap instruksi mempunyai satu set operand. Pada prosesor vektor, setiap instruksi mempunyai multiset operand seperti elemen-elemen dua larik dan operasi yang sama dilakukan secara simultan pada set operand yang berbeda. Pada prosesor skalar *non-pipeline* (sekuensial), tidak ada tumpang tindih siklus-siklus instruksi berurutan. Pada prosesor *pipeline*, multi-instruksi beroperasi pada berbagai tingkat siklus instruksi secara simultan dalam bagian-bagian prosesor berbeda. Prosesor

larik mempunyai unit-unit eksekusi multi yang beroperasi secara simultan pada elemen-elemen sebuah vektor. Prosesor superskalar adalah prosesor skalar yang melakukan siklus instruksi jamak/multi secara simultan untuk instruksi-instruksi berurutan. Dua atau lebih instruksi diinisiasi secara simultan dan dalam sebuah siklus clock tunggal, operasi yang sama (fetch, dekode, eksekusi, dan sebagainya) dieksekusi pada dua atau lebih instruksi berurut dalam paralel. Untuk mencapai hal ini, beberapa unit-unit fungsional dibangun di dalam prosesor.

Klasifikasi arsitektur prosesor yang baru lainnya adalah *Very Long Instruction Word* (VLIW) yang memperbaiki arsitektur superskalar standar. VLIW memanfaatkan paralelisme level instruksi dalam program dengan mengeksekusi lebih dari satu instruksi dasar secara simultan.

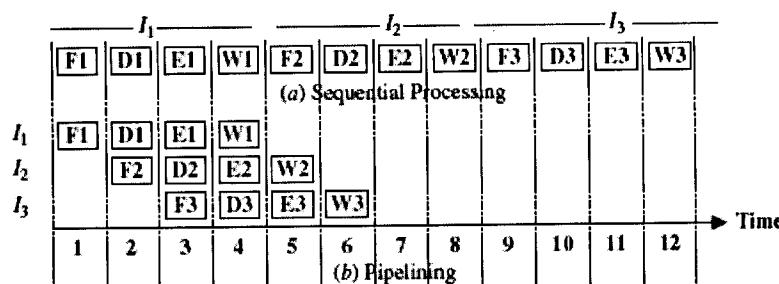
Pada bagian ini kita memandang pemrosesan paralel dengan tiga topik utama yaitu:

1. Pemrosesan Pipeline (*pipeline processing*)
2. Pemrosesan Vektor (*vector processing*)
3. Pemrosesan Larik (*array processing*)

Pemrosesan *pipeline* merupakan suatu teknik implementasi di mana sub-operasi aritmetika atau fase-fase dari suatu siklus instruksi komputer tumpang tindih ketika dieksekusi. Pemrosesan vektor berkenaan dengan komputasi yang melibatkan vektor-vektor besar dan matriks-matriks. Pemrosesan larik melakukan komputasi pada larik data yang besar.

11.3 PIPELINING

Pipelining adalah suatu teknik pemecahan satu pekerjaan/tugas menjadi beberapa sub-tugas, dan mengeksekusi sub-tugas tersebut secara bersamaan/paralel dalam unit-unit multihardware atau segmen-semen. Gambar 11.8 mengilustrasikan perbedaan dasar antara pengeksekusian empat sub-tugas suatu instruksi (dalam hal ini fetching F, decoding D, eksekusi E dan penulisan hasil W) menggunakan pipeline dan pemrosesan sekuensial. Jelas dari Gambar 11.8 terlihat total waktu yang dibutuhkan untuk memproses tiga instruksi (I_1 , I_2 dan I_3) hanya 6 unit waktu jika digunakan pipeline empat-tingkat dibandingkan 12 unit waktu jika digunakan pemrosesan sekuensial. Penghematan waktu eksekusi mencapai 50 % untuk tiga instruksi tersebut.



Gambar 11.8 Konsep pipelining

Tujuan yang ingin dicapai dalam pipeline adalah untuk meningkatkan throughput. Waktu yang digunakan untuk eksekusi setiap tugas sama dengan waktu yang digunakan untuk satu eksekusi non-pipeline. Tetapi karena eksekusi tugas yang berurutan dilakukan secara bersamaan, maka jumlah tugas yang dapat dieksekusi dalam suatu waktu yang disediakan lebih tinggi yaitu hardware pipeline menyediakan throughput yang lebih baik dibandingkan dengan hardware non-pipeline.

Pipeline ada dua jenis:

1. Pipeline instruksi. Pipeline instruksi memecah operasi-operasi siklus instruksi menjadi multilangkah yang dieksekusi satu per satu pada segmen-semen yang berbeda dari prosesor.
2. Pipeline aritmetika. Pipeline aritmetika membagi suatu operasi aritmetika seperti perkalian menjadi langkah-langkah multi-aritmetika yang masing-masing dieksekusi satu per satu pada segmen-semen aritmetika di dalam ALU.

11.3.1 Pipeline Instruksi

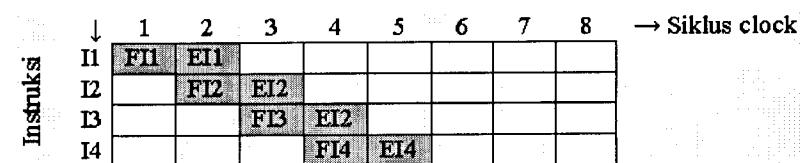
Pipeline instruksi terdiri atas multisegmen yang masing-masing melakukan salah satu fungsi dari siklus instruksi pada semua instruksi. Untuk keperluan ini, hardware prosesor disusun menjadi segmen-semen yang independen.



UF = unit fetch UE = unit eksekusi BI = buffer instruksi

Gambar 11.9 Pipeline dua tingkat

Kasus yang paling sederhana adalah pipeline dua tingkat seperti yang ditunjukkan pada Gambar 11.9. Siklus instruksi dipecah menjadi dua langkah—Fetch Instruksi dan Eksekusi Instruksi. Unit fetch (UF) melakukan langkah pengambilan instruksi, sedangkan unit eksekusi (UE) melaksanakan langkah eksekusi instruksi termasuk penyelesaian siklus instruksi. Buffer instruksi (BI) digunakan untuk menyimpan sementara instruksi yang diambil oleh UF. UF mentransfer instruksi ke dalam BI, dan UE menerima instruksi dari BI. Gambar 11.10 menunjukkan diagram pewaktuan dari pipeline instruksi dua tingkat selama mengeksekusi empat instruksi yaitu dari I₁ sampai I₄. Kita menganggap bahwa waktu yang digunakan oleh UF atau UE untuk operasi tersebut hanya satu siklus clock. Setiap instruksi diselesaikan dalam dua siklus clock. Karena itu, instruksi pertama diselesaikan pada akhir siklus clock kedua. Instruksi kedua diselesaikan pada akhir siklus clock ketiga. Dari gambar sebelumnya, terlihat bahwa jumlah siklus clock yang diperlukan untuk mengeksekusi empat instruksi adalah lima siklus clock. Dengan cara yang sama, jumlah siklus clock yang diperlukan untuk mengeksekusi tujuh instruksi adalah delapan siklus clock. Pada prosesor non-pipeline, delapan siklus clock diperlukan untuk menyelesaikan empat instruksi. Intel 8088 dan 8086 didesain dengan dua tingkat pipeline. UF pada 8088 mengambil/membaca empat byte dari aliran instruksi sedangkan 8086 mengambil enam byte. Karena itu, buffer instruksi adalah sebuah antrian FIFO (first in first out) di mana UF menyimpan instruksi-instruksi pre-fetch.



Gambar 11.10 Diagram pewaktuan pipeline dua tingkat

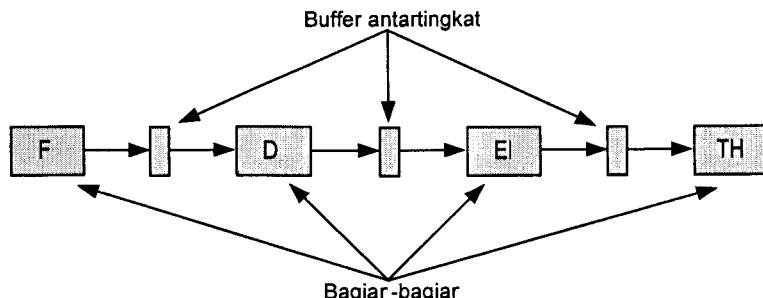
Gambar 11.11 menunjukkan suatu pipeline instruksi empat tingkat dan Gambar 11.12 menunjukkan diagram pewaktuan untuk eksekusi enam instruksi. Gambar 11.13 menampilkan pipeline instruksi enam tingkat.

Analisis berikut menunjukkan bahwa waktu yang digunakan untuk menyelesaikan m instruksi dalam suatu pipeline n tingkat diaproksimasi kira-kira sama dengan n .

Waktu yang digunakan instruksi pertama = nt_c di mana t_c adalah durasi dari satu siklus clock.

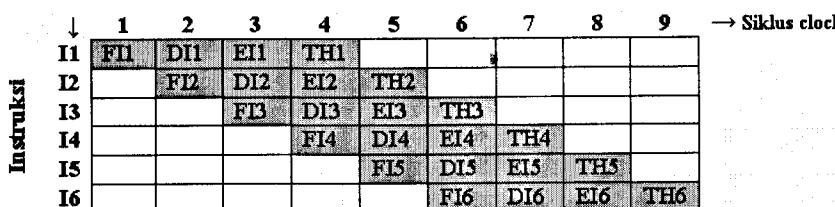
Waktu yang digunakan untuk sisa instruksi $(m - 1)$ adalah = $(m - 1)t_c$.

Total waktu yang digunakan untuk m instruksi = $(nt_c) + (m - 1)t_c = (n + m - 1)t_c$.

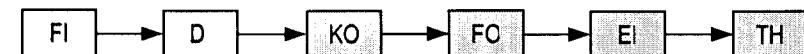


FI = Fetch instruksi DI = Dekode instruksi EI = Eksekusi instruksi TH = Tulis hasil

Gambar 11.11 Pipeline empat tingkat



Gambar 11.12 Diagram pewaktuan pada pipeline empat tingkat



FI = Fetch instruksi DI = Dekode instruksi KO = Kalkulasi alamat operand

FO = Fetch operand EI = Eksekusi instruksi TH = Tulis hasil

Catatan: buffer antartingkat tidak diperlukan

Gambar 11.13 Prosesor pipeline enam tingkat

Jika prosesor adalah prosesor non-pipeline, maka waktu yang digunakan untuk m instruksi dalam nmt_c dengan menganggap waktu siklus instruksi sama dengan nt_c .

Gain kinerja (speed-up) pipeline adalah waktu yang digunakan dalam mode non-pipeline dibagi waktu yang digunakan dalam mode pipeline, dapat dituliskan:

$$\text{Speed-up} = \frac{\text{waktu, pemrosesan non-pipeline}}{\text{waktu, pemrosesan pipeline}} = \frac{nmt_c}{(n+m-1)t_c} = \frac{nm}{n+m-1}$$

Untuk nilai m yang besar yang lebih besar dari $(n - 1)$, maka $(n + m - 1)$ mendekati m . Karena itu speed-up = $nm/m = n$.

Jadi, speed-up maksimum secara teoretis adalah sama dengan jumlah tingkat dalam pipeline. Jumlah siklus clock yang diperlukan untuk mengeksekusi m instruksi adalah m karena satu instruksi selesai setiap satu siklus clock.

Secara praktis, kehadiran instruksi cabang dalam program mengurangi perolehan speed-up pada pipelining dan meningkatkan jumlah siklus clock yang dibutuhkan untuk m instruksi.

$$\text{Throughput} = \text{jumlah tugas yang dieksekusi per unit waktu} = \frac{m}{(n+m-1)t_c}$$

$$\text{Efisiensi} = \text{rasio antara speed-up aktual dan speed-up maks.} = \frac{\text{speed-up}}{n} = \frac{m}{n+m-1}$$

Contoh 11.1

Sebuah program menggunakan 500 ns untuk eksekusi pada sebuah prosesor non-pipeline. Anggap kita ingin menjalankan 100 program dengan tipe yang sama pada prosesor pipeline lima tingkat dengan periode clock 20 ns. Berapa rasio speed-up dari pipeline? Berapa speed-up maksimum yang dapat dicapai? Berapa throughput?

Solusi:

Waktu yang digunakan oleh satu program pada prosesor non-pipeline = $m \cdot t_c = 500$ ns. Anggap siklus instruksi menggunakan lima clock masing-masing 20 ns, jadi $m = m \cdot t_c / n \cdot t_c = 500 / 100 = 5$. Karena itu, satu program mempunyai 5 instruksi. Jumlah instruksi dalam 100 program = 500 instruksi.

Sekarang kita kalkulasi speed-up ketika 100 program dijalankan dalam prosesor pipeline.

Untuk $n = 5$ dan $m = 500$, $t_c = 20$ ns.

$$\text{Speed - up} = \frac{nm}{n + m - 1} = \frac{5 \times 500}{(5 + 500 - 1)} = 4,96$$

Speed-up maksimum = $n = 5$

Efisiensi (ratio speed-up) = speed-up aktual / speed-up maks = $4,96 / 5 = 0,992 = 99,2\%$

$$\text{Throughput} = \frac{m}{(n + m - 1)t_c} = \frac{500}{(5 + 500 - 1) \times 20} = 0,0496$$

11.3.2 Frekuensi Clock

Waktu yang digunakan untuk menyelesaikan langkah-langkah instruksi pada berbagai tingkat berbeda-beda. Misalnya, jika pengambilan instruksi memerlukan waktu yang lebih besar daripada pendekodean instruksi, maka periode clock pipeline tidak akan lebih kecil dari waktu yang digunakan oleh tingkat yang lebih tinggi dalam pipeline. Akibatnya, segmen-segmen pipeline yang telah menyelesaikan operasinya lebih cepat dipaksa untuk menunggu. Untuk mengurangi waktu tunggu ini dan untuk mencapai efisiensi yang lebih baik dari pipelining, maka langkah-langkah yang lebih besar dipecah menjadi lebih dari satu tingkat yang kecil agar waktu yang diperlukan untuk setiap tingkat berkurang. Hal ini mengizinkan frekuensi

clock yang lebih tinggi. Metode lain adalah pengoperasian secara parallel unit-unit multi untuk suatu tingkat tunggal. Pada pipeline ideal, waktu yang digunakan oleh setiap tingkat adalah sama. Waktu ini adalah durasi periode clock.

11.3.3 Pipeline Bubble

Untuk menyelesaikan instruksi tertentu, beberapa langkah dalam pipeline ada yang tidak relevan, hal ini disebut '*pipeline bubble*'. Untuk beberapa instruksi, sejumlah langkah dalam pipeline tidak akan dikerjakan. Misalnya, untuk instruksi NOOP, hanya ada dua langkah yang diperlukan yaitu pengambilan instruksi dan dekode instruksi. Karena itu instruksi ini menyebabkan '*bubble*' dalam pipeline setelah dua langkah pertama. Demikian pula, instruksi LOAD tidak dikerjakan pada tingkat penulisan hasil. Namun, instruksi-instruksi tersebut harus melalui semua tingkatan tanpa melompati langkah yang tidak diinginkan.

11.3.4 Pipeline Hazard

Pipeline beroperasi dengan efisiensi penuh asalkan program adalah suatu yang ideal yang tidak mengganggu pada tujuan penyelesaian dari satu instruksi untuk setiap clock. Tetapi kadang-kadang program praktis mempunyai tipe yang berbeda dari hubungan antar-instruksi yang dihasilkan pada tempat pipeline. Dengan kata lain, ketergantungan antara instruksi-instruksi yang berurutan/berdekatan menyebabkan hazard di dalam pipeline dan memengaruhi operasi mulus dari pipeline instruksi. Ada tiga hal utama yang menyebabkan hazard:

1. Konflik sumber daya disebut juga *structural hazard*
2. Ketergantungan data disebut juga *data hazard*
3. Kesulitan pencabangan disebut juga *control hazard*

Konflik Sumber Daya

Bila dua segmen berbeda dalam pipeline memerlukan sumber daya hardware secara bersamaan, maka akan menghasilkan konflik sumber daya (*resource conflict*). Berikut beberapa contoh khas:

1. Anggap pada tingkat akhir TH (tulis hasil) suatu instruksi memerlukan akses memori untuk penyimpanan hasil, dan pada waktu yang

sama FO (pengambilan operand) juga memerlukan akses memori untuk pengambilan operand untuk instruksi berikutnya. Jelas, salah satunya akan mengalami penundaan yang memengaruhi kinerja pipeline. Jika kita menunda akses memori TH, dia akan menunda operasi pipeline untuk instruksi berikutnya hingga akses memori TH selesai. Dengan kata lain jika kita menunda akses memori FO, operasi parsial instruksi antara FO dan TH akan berjalan terus sedangkan bagian yang mendahului FO akan tetap diam. Satu solusi untuk mengatasi masalah ini adalah dengan menggunakan ‘buffer keluaran tulis’. Tingkat TH dengan mudah menyimpan hasil dan ‘informasi kontrol lainnya’ (yaitu alamat memori) pada buffer tulis sebagai pengganti pengaksesan memori utama. Dengan kata lain operasi penyimpanan dalam memori utama dihilangkan dari pipeline. Sirkuit kontrol tambahan melakukan operasi penulisan memori bila memori tersedia dengan mengambil informasi dari buffer tulis. Intel 80486 mengikuti teknik ini. Dia mempunyai empat buffer keluaran yang digunakan bersama oleh operasi penulisan memori dan operasi penulisan I/O. Karena itu jika bus tidak tersedia, diizinkan hingga empat keluar dari pipeline. Jika tetap, bus secara kontinu tidak tersedia, maka pipeline dihentikan hingga buffer tulis menjadi minimal kosong secara parsial.

- Konflik sumber daya lainnya terjadi bila tingkat FI memerlukan akses memori untuk pengambilan instruksi dan tingkat FO memerlukan akses memori untuk pengambilan operand pada saat yang sama. Hal ini dapat diselesaikan dengan mempunyai unit-unit memori cache terpisah untuk instruksi (cache kode) dan operand (cache data) seperti pada kasus mikropresessor Pentium. Secara bersamaan, cache kode dan cache data dapat diakses.

Ketergantungan Data

Jika operand suatu instruksi merupakan hasil dari instruksi sebelumnya yang belum selesai dalam pipeline, maka kasus ini disebut ketergantungan data (*data dependency*).

	1	2	3	4	5	6	7	8	9	10	11	12	→ Siklus clock
ADD R1,R2	FI	DI	KO	FO	EI	TH							
SUB R1,R2		FI	DI	KO	FO	EI	TH				
			FI	DI	KO	FO	EI	TH			

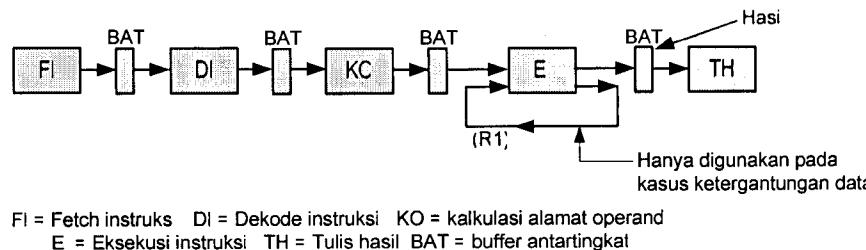
Pipeline diam pada dua siklus clock

Gambar 11.14 Pipeline diam karena ketergantungan data—enam tingkat

Gambar 11.14 menunjukkan pipeline diam karena ketidaktersediaan operand untuk tingkat FO instruksi kedua sebab instruksi pertama belum selesai. Tidak ada yang dilakukan pada siklus clock 5 dan 6 untuk instruksi kedua dan instruksi berikutnya. Karena itu harus dibayar dengan penalti karena ketergantungan data pada dua siklus clock. Gambar 11.15 menunjukkan efek yang sama pada pipeline empat tingkat. Solusi standar untuk ketergantungan data disebut ‘*operand forwarding*’. Gambar 11.16 mengilustrasikan konsep ini. Hasil dari bagian eksekusi (EI) merupakan umpan balik ke input bagian yang sama agar dia tersedia dengan mudah (sebelum diperlukan) untuk digunakan pada langkah berikutnya untuk instruksi berikutnya. Unit kontrol meng-‘enable’ jalur ini pada waktu yang tepat.

	1	2	3	4	5	6	7	8	9	10	11	→ Siklus clock
1. ADD R1,R2	FI	DI	EI	TH								
2. SUB R1,R2		FI	DI		DI	EI	TH					
3.		FI			DI	EI	TH					
4.					FI	DI	EI	TH				
5.						FI	DI	EI	TH			

Gambar 11.15 Pipeline diam karena ketergantungan data—kasus empat tingkat



Gambar 11.16 Ilustrasi operand forwarding

Ketergantungan data diklasifikasikan dalam tiga jenis:

1. *Read-After-Write (RAW) hazard*
2. *Write-After-Read (WAR) hazard*
3. *Write-After-Write (WAW) hazard*

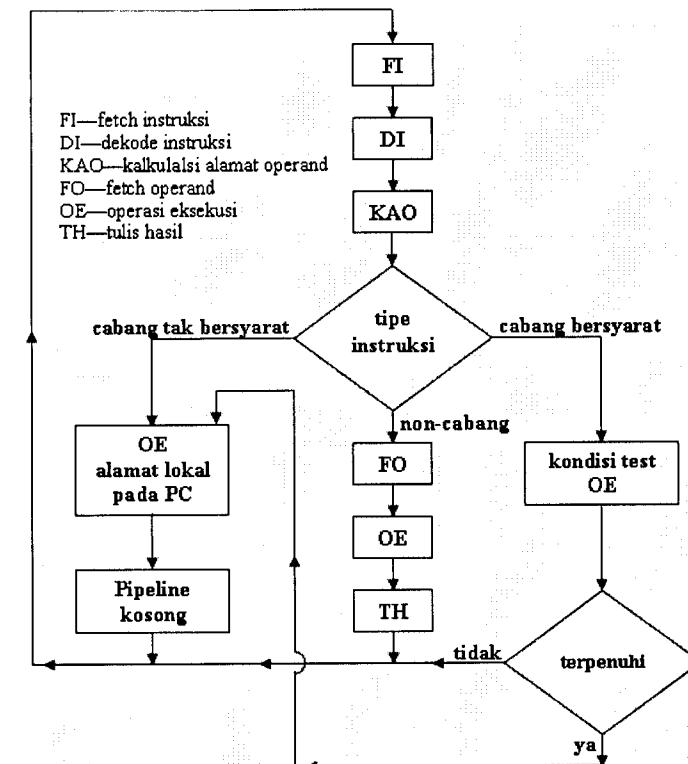
Antara dua instruksi A dan B terjadi '*RAW hazard*' jika B membaca beberapa item data yang dimodifikasi oleh A. '*WAR hazard*' terjadi jika B memodifikasi beberapa item data yang dibaca oleh A. '*WAW hazard*' terjadi jika A dan B keduanya memodifikasi beberapa item data. Deteksi dini situasi hazard dan koreksi awal untuk mencegah hazard umumnya dilakukan oleh *hardware logic* yang tepat.

Solusi Software pada Data Hazard

Di sini, kompiler mengindra hazard dengan melakukan *scanning* instruksi dalam program. Pada keadaan data hazard, dia menyiapkan sebuah instruksi NOOP di antara dua instruksi, tempat di mana ketergantungan data. Akibatnya, terjadi tunda (*delay*) sebanyak dua atau lebih siklus clock dan menyelesaikan ketergantungan data. Keuntungan metode ini adalah penggunaan hardware yang sederhana dan bebas menyiapkan beberapa instruksi berguna dalam slot NOOP di mana pun oleh kompiler.

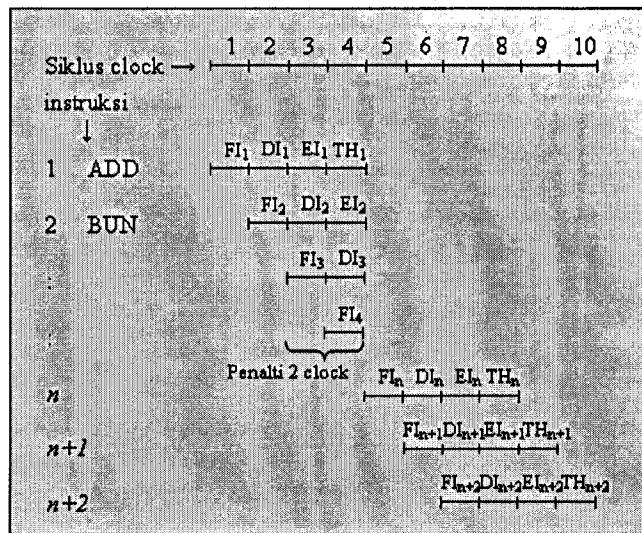
Kesulitan Pencabangan

Bila suatu instruksi cabang memasuki pipeline, dia dikenali ketika fase pendekodean instruksi. Tindakan khusus dilakukan oleh prosesor untuk instruksi cabang tidak bersyarat (*unconditional branch*) dan instruksi cabang bersyarat (*conditional branch*) dapat dilihat Gambar 11.17.



Gambar 11.17 Efek instruksi cabang

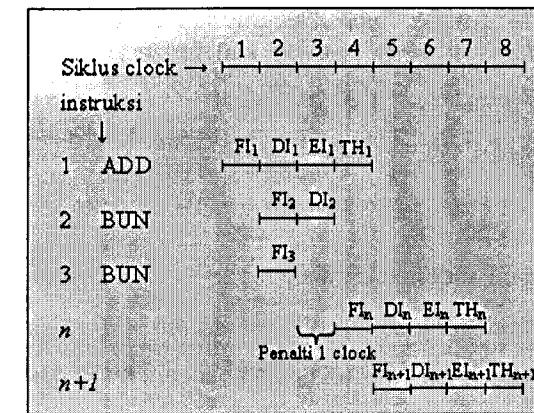
Pada Gambar 11.18 diilustrasikan instruksi cabang dengan pipeline empat tingkat. Kita mempunyai dua clock penalti karena instruksi ini. Pada clock 4, alamat cabang diisikan ke PC (*program counter*). Karena itu, langkah-langkah yang dilakukan pada clock 3 dan 4 untuk instruksi ketiga dan keempat dibilas/dipulihkan (*flush*). Pada clock 5, instruksi baru diambil/dibaca dari alamat cabang. Jika tidak ada instruksi cabang, maka clock 5 melakukan FI5, DI4 dan EI3. Pemborosan disebabkan karena hasil-hasil instruksi cabang tak bersyarat pada aksi FI3, DI3 dan FI4 sudah selesai tetapi tidak digunakan. Dengan kata lain, instruksi cabang menyebabkan tambahan tunda sebesar $(n - 1)$ clock.



Gambar 11.18 Efek penalti cabang tak-bersyarat

Hasil-hasil instruksi cabang dalam pembatalan instruksi yang telah dieksekusi sebagian, dilakukan dengan jalan pembilasan dari pipeline dan pipeline disegarkan kembali. Kehadiran instruksi cabang yang banyak di dalam program mengurangi speed-up yang dicapai pipeline instruksi. Jika p adalah hasil bagi antara jumlah instruksi-cabang terhadap total jumlah instruksi, maka $1/p$ adalah faktor batas speed-up yang disebabkan instruksi cabang.

Tindakan khusus dilakukan oleh hardware untuk mengindra instruksi cabang sebelumnya bila berada pada antrian instruksi. Dalam kasus ini, instruksi pada alamat cabang diambil lebih awal dari semestinya. Gambar 11.19 mengilustrasikan efek pada instruksi cabang tak-bersyarat tersebut. Alamat cabang diisikan ke PC pada clock 2 daripada clock 3, instruksi diambil dari alamat ini. Penalti cabang berkangur hanya satu siklus clock.



Gambar 11.19 Efek penalti cabang tak-bersyarat—menggunakan logic tambahan

Waktu Tunda yang Disebabkan Instruksi Cabang Bersyarat

Instruksi cabang bersyarat mempunyai dua kondisi:

1. Kondisi yang diuji berhasil; di sini isi pipeline dibilas dan instruksi pada alamat cabang masuk ke pipeline.
2. Kondisi yang diuji tidak berhasil dan di sini dilakukan eksekusi sekuen-sial berikutnya; jadi dalam kasus ini, isi pipeline tidak terpengaruh.

Waktu tunda yang disebabkan oleh instruksi cabang bersyarat dijelaskan menggunakan contoh berikut:

Anggap ada m instruksi yang akan dieksekusi. p adalah probabilitas instruksi cabang bersyarat dan q adalah probabilitas instruksi cabang yang sukses. Karena itu jumlah instruksi yang menyebabkan instruksi cabang sukses adalah mpq (lihat Gambar 11.20).

Pada prosesor pipeline, instruksi diselesaikan pada kecepatan satu instruksi per siklus clock. Jika tidak ada instruksi cabang di dalam kancah (stream), rutin program selesai dalam m clock. Semua hasil instruksi cabang pada pembilasan pipeline dan pengisian kembali adalah dari alamat cabang. Instruksi dari alamat cabang (alamat target) memerlukan n siklus clock untuk pengeksekusian.

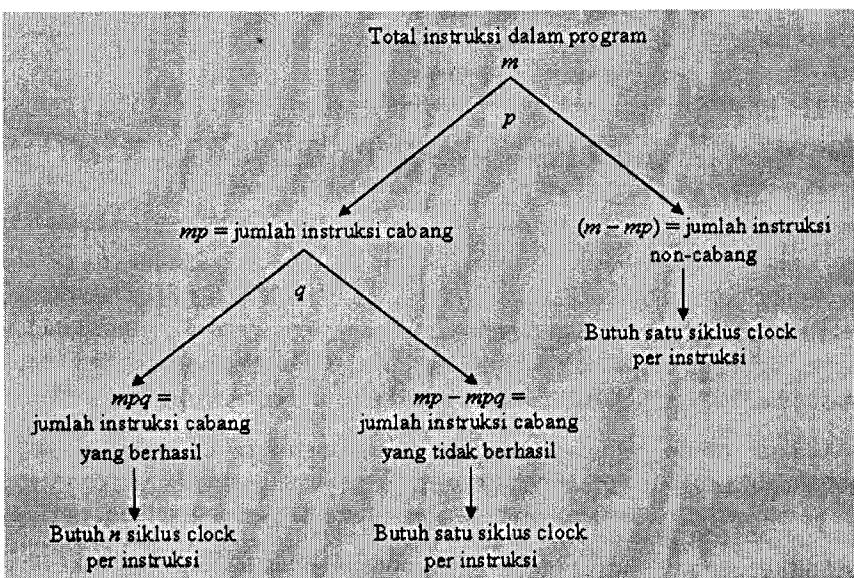
Total siklus clock yang diperlukan untuk mengesekusi m instruksi = (waktu tunda dalam siklus clock yang dimulai dari instruksi cabang) + (siklus clock yang diperlukan untuk instruksi non-cabang):

$$\begin{aligned} &= mpqn + \{(m - mp) + mp - mpq\} \\ &= mpqn + \{(m - mpq)\} \\ &= mpqn + m(1 - pq) \end{aligned}$$

Jadi untuk m instruksi, dieksekusi dalam $mpqn + m(1 - pq)$ siklus clock.

Rata-rata jumlah clock per instruksi (CPI) = $1 + pq(n - 1)$

Jika program menyebabkan tidak ada pencabangan, maka $q = 0$ dan CPI = 1.



Gambar 11.20 Waktu tambahan yang ditimbulkan oleh pencabangan

Contoh 11.2

Sebuah program mempunyai instruksi cabang 20%. Anggap semua pencabangan berhasil, hitung speed-up dalam pipeline dengan enam tingkat.

Solusi:

Jika tidak ada instruksi cabang dalam program, speed-up = $n =$ jumlah tingkat = 6

$$p = 0,2$$

$$q = 1,0$$

$$n = 6$$

$$CPI = 1 + pq(n - 1) = 1 + 0,2 \times 1,0 \times (6-1) = 2,0$$

Prosesor non-pipeline memerlukan 6 clock per instruksi

$$\text{Speed-up} = 6/2 = 3,0$$

Contoh 11.3

Sebuah program mempunyai instruksi cabang 20%. Jika dijalankan pada prosesor pipeline lima tingkat, dari pengamatan menghasilkan pencabangan sukses 60%. Hitung peningkatan kinerjanya pada prosesor pipeline. Berapa speed-up maksimum yang mungkin jika tidak ada pencabangan?

Solusi:

$$p = 0,2$$

$$q = 0,6$$

$$n = 5$$

$$CPI = 1 + pq(n - 1) = 1 + 0,2 \times 0,6 \times (5-1) = 1,48$$

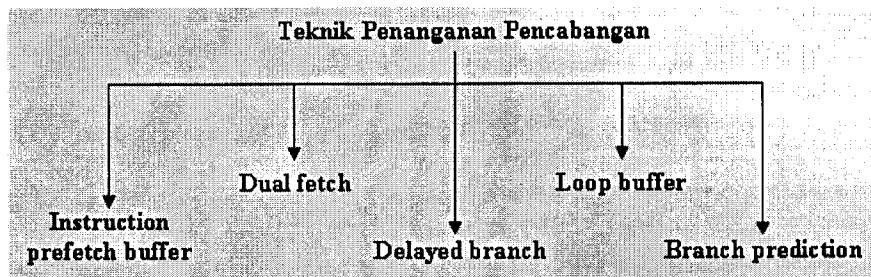
Prosesor non-pipeline memerlukan 5 clock per instruksi

$$\text{Speed-up} = 5/2 = 2,5$$

Speed-up maksimum jika tidak ada pencabangan = $n = 5$

Peminimalan Pengaruh Instruksi-Instruksi Cabang Bersyarat

Sejumlah teknik yang digunakan pada berbagai prosesor yang berbeda untuk mengurangi waktu tunda yang disebabkan oleh instruksi-instruksi cabang bersyarat dapat dilihat pada Gambar 11.21. Di sini hanya akan dijelaskan salah satunya, yaitu *delayed branch* (yang merupakan teknik software).



Gambar 11.21 Teknik-teknik penanganan pencabangan

Delayed Branch

Instruksi yang mengikuti instruksi pencabangan (*branching*) membutuhkan *delay slot*, seperti yang ditunjukkan pada Gambar 11.18 yaitu dua delay slot dan yang ditunjukkan pada Gambar 11.19 dengan satu delay slot. Kompiler mencoba melakukan penyusunan kembali instruksi-instruksi di manapun agar beberapa instruksi yang tidak berhubungan akan dihilangkan setelah instruksi pencabangan yang dapat dilanjutkan dalam pipeline. Jika dalam situasi tersebut tidak ada instruksi yang dapat dibawa setelah instruksi pencabangan, maka kompiler menyisipkan sebuah instruksi NOOP setelah instruksi pencabangan. Ilustrasi ini dapat dilihat pada potongan program berikut:

Load	R1,X
SUB	R1,R2
JZ	TOP
SHL	R1
ADD	R1,R3
TOP	SHR R1
INC	R4

(a) Program awal

Load	R1,X
SUB	R1,R2
JZ	TOP
NOOP	
SHL	R1
ADD	R1,R3
TOP	SHR R1
INC	R4

(b) Delayed branch dengan tundaan

Load	R1,X
SUB	R1,R2
JZ	TOP
INC	R4
SHL	R1
ADD	R1,R3
TOP	SHR R1

(c) Program setelah disusun kembali

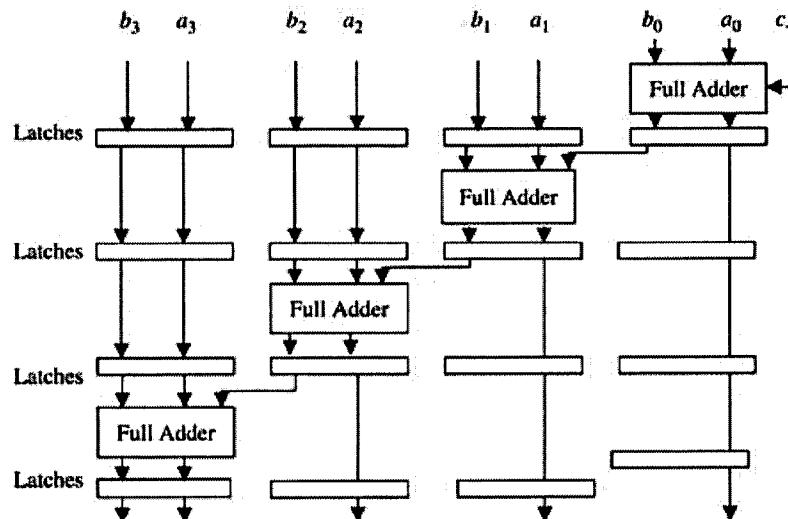
11.4 PIPELINE ARITMETIKA

Prinsip-prinsip yang digunakan dalam pipeline instruksi dapat digunakan untuk meningkatkan kinerja komputer dalam pelaksanaan operasi aritmetika seperti penjumlahan, pengurangan, dan perkalian. Dalam hal ini, prinsip-prinsip tersebut akan digunakan untuk merealisasikan sirkuit aritmetika di dalam ALU. Pada bagian ini, kita akan membahas penggunaan pipeline aritmetika dalam meningkatkan kecepatan aritmetika. Kita akan mulai dengan operasi aritmetika fixed-point dan dilanjutkan dengan pembahasan operasi aritmetika floating-point.

11.4.1 Pipeline Aritmetika Titik-Tetap

Operasi aritmetika titik-tetap (*fixed-point*) dasar yang dilakukan dalam ALU adalah penjumlahan dua operand *n*-bit $A = a_{n-1}a_{n-2} \dots a_2a_1a_0$ dan $B = b_{n-1}b_{n-2} \dots b_2b_1b_0$. Penjumlahan kedua operand ini dapat dilakukan menggunakan beberapa teknik. Teknik-teknik ini pada dasarnya berbeda dalam dua hal, yaitu tingkat kompleksitas dan kecepatan. Kedua hal ini agak kontradiktif, yaitu realisasi yang sederhana menghasilkan sirkuit yang

lambat sedangkan realisasi yang kompleks menghasilkan sirkuit yang cepat. Misalnya, penjumlahan *carry ripple through* (CRT) dan penjumlahan *carry look-ahead* (CLA). Penjumlahan CRT sederhana tetapi lambat, sedangkan CLA kompleks tetapi lebih cepat.

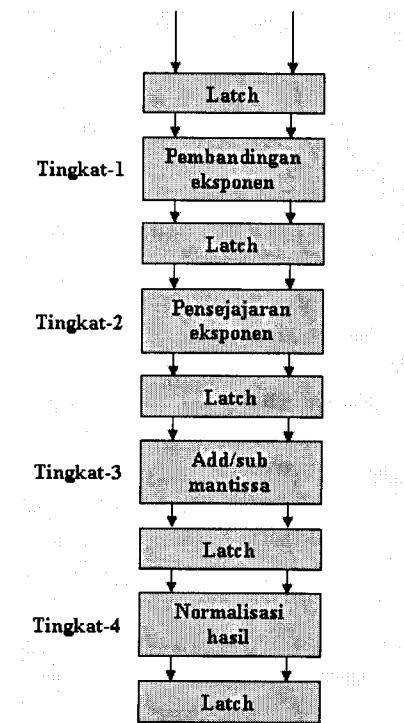


Gambar 11.22 Penjumlahan CRT 4-bit yang disederhanakan

Pada Gambar 11.22 ditunjukkan sebuah penjumlahan CRT 4-bit yang disederhanakan. Pada kasus ini, dua operand A dan B diumpulkan pada penjumlahan CRT melalui penggunaan elemen-elemen sinkronisasi, seperti latch dengan clock. Latch ini akan menjamin perpindahan nilai carry parsial dalam penjumlahan CRT yang disinkronisasikan dengan input penjumlahan tingkat berikutnya dengan bit-bit operand yang lebih tinggi. Misalnya, *carry-out* (c_0) pertama yang tiba dan pasangan bit kedua (a_1 dan b_1) disinkronisasikan pada input penjumlahan penuh kedua menggunakan sebuah latch.

Walaupun operasi penjumlahan CRT yang disederhanakan tetap dalam prinsip yang sama, yaitu *carry-out* merambat secara seri melalui penjumlahan, bagian latch membolehkan beberapa (multi) pasangan operand berada pada penjumlahan dalam waktu yang sama. Misalnya, penjumlahan M pasang

operand, di mana operand setiap pasang adalah n -bit. Waktu yang dibutuhkan untuk melakukan penjumlahan M pasang operand menggunakan penjumlahan CRT adalah $T_{np} = M \times n \times T_a$ di mana T_a adalah waktu yang dibutuhkan untuk melakukan penjumlahan bit tunggal. Ini akan dibandingkan dengan waktu yang dibutuhkan untuk melakukan komputasi yang sama dengan menggunakan pipeline penjumlahan CRT yaitu $T_{pp} = (n + M - 1) \times T_a$. Misalnya, jika $M = 16$ dan $n = 64$ -bit, maka kita mempunyai $T_{np} = 1024 \times T_a$ dan $T_{pp} = 79 \times T_a$. Jadi, menghasilkan speed-up sekitar 13. Pada kasus ekstrem di mana memungkinkan untuk memberikan pasangan operand (M) dalam jumlah tak terbatas ke penjumlahan CRT pada waktu yang sama, speed-up akan mencapai 64, jumlah bit pada setiap operand.



Gambar 11.23 Pipeline penjumlahan/pengurangan floating point

11.4.2 Pipeline Aritmetika Titik-Mengambang

Dengan menggunakan pendekatan yang sama, memungkinkan melakukan pipeline penjumlahan/pengurangan bilangan titik-mengambang (*floating-point*, FP). Pada kasus ini, pipeline harus dibuat berdasarkan operasi-operasi yang diperlukan untuk melakukan penjumlahan FP. Operasi utama yang diperlukan dalam penjumlahan FP adalah pembanding eksponen, pencegahan eksponen, penjumlahan, dan normalisasi. Karena itu, memungkinkan menyusun pipeline empat-tingkat seperti yang diperlihatkan pada Gambar 11.23 dengan penjumlah FP. Memungkinkan mempunyai multiset operand FP dalam penjumlahan secara bersamaan, jadi mengurangi waktu keseluruhan yang dibutuhkan untuk penjumlahan FP. Sinkronisasi latch dibutuhkan, untuk sinkronisasi operand pada input tingkat dalam penjumlahan FP.

11.5 PIPELINE MESIN RISC

Salah satu karakteristik yang dimiliki RISC (*reduced instruction set computer*) adalah kemampuannya yang dapat digunakan untuk mengefisiensikan pipeline instruksi. Kesederhanaan instruction set dapat digunakan untuk mengimplementasikan suatu pipeline instruksi menggunakan jumlah sub-operasi yang sedikit dengan mengeksekusinya masing-masing dalam satu siklus clock. Karena alasan format instruksiya adalah fixed-length, pendekodean (penerjemahan) operasi dapat terjadi pada saat yang bersamaan ketika pemilihan register. Semua instruksi manipulasi data mempunyai operasi register-ke-register. Karena semua operand berada dalam register, maka tidak diperlukan kalkulasi alamat efektif atau pengambilan operand dari memori. Karena itu, pipeline instruksi dapat diimplementasikan dengan dua atau tiga segmen. Satu segmen mengambil instruksi dari memori program, dan segmen lainnya mengeksekusi instruksi dalam ALU. Segmen yang ketiga dapat digunakan untuk menyimpan hasil operasi ALU dalam sebuah register tujuan.

Instruksi transfer data pada RISC terbatas untuk mengambil dan menyimpan instruksi. Instruksi-instruksi ini menggunakan pengalaman tak-langsung register. Mereka biasanya memerlukan tiga atau empat tingkat dalam pipeline. Untuk mencegah konflik antara sebuah akses memori dalam mengambil instruksi dan dalam mengambil atau menyimpan operand, maka mesin RISC umumnya menggunakan dua bus secara terpisah

dengan dua memori: satu untuk penyimpanan instruksi dan yang lainnya untuk penyimpanan data. Dua memori kadang-kadang dapat beroperasi pada kecepatan yang sama dengan clock CPU dan mengarahkan ke memori cache.

Salah satu keuntungan utama RISC adalah kemampuannya meng-eksekusi instruksi pada kecepatan (rate) satu per siklus clock. Tidak mungkin mengharapkan bahwa setiap instruksi dapat diambil dari memori dan dieksekusi dalam satu siklus clock. Keuntungan RISC terhadap CISC (*complex instruction set computer*) adalah bahwa RISC dapat mencapai segmen pipeline dengan hanya memerlukan satu siklus clock, sedangkan CISC menggunakan banyak segmen dalam pipeline, dengan segmen paling panjang memerlukan dua atau lebih siklus clock.

Karakteristik RISC yang lain adalah mendukung kompiler dalam menerjemahkan program bahasa tingkat tinggi menjadi program bahasa mesin. Sebagai pengganti perancangan hardware dalam penanganan kesulitan-kesulitan yang berhubungan dengan konflik data dan penalti pencabangan (*branch penalties*), prosesor RISC mengandalkan efisiensi kompiler untuk mendeteksi dan mengurangi waktu tunda (*delay*) yang menyertai masalah ini.

11.6 PEMROSESAN VEKTOR

Prosesor vektor digunakan untuk komputasi saintifik kinerja tinggi, biasanya aritmetika vektor dan matriks. Komputer dengan kemampuan pemrosesan vektor umumnya untuk aplikasi khusus misalnya:

- Peramalan cuaca
- Eksplorasi minyak
- Analisis data seismik
- Diagnosa medis
- Simulasi penerbangan luar angkasa dan aerodinamik
- Intelejensi tiruan dan sistem pakar
- Pemrosesan citra
- Pengindraan jauh

Untuk memenuhi level kinerja tinggi yang disyaratkan, diperlukan penggunaan hardware tercepat dan andal dan memerlukan prosedur-prosedur yang inovatif dan teknik pemrosesan parallel, karena prosesor

tradisional untuk komputasi semacam itu memerlukan penanganan yang berhari-hari (lama). Prosesor vektor untuk aplikasi tersebut pada saat ini antara lain Cray Y-MP dan Convex C3880.

Prosesor vektor efisien menangani operasi-operasi aritmetika pada elemen-elemen larik yang dikenal dengan sebutan ‘vektor’. Untuk sebuah instruksi vektor tunggal, prosesor vektor melakukan operasi-operasi multi-homogen pada suatu larik elemen-elemen. Prosesor vektor menggunakan paralelisme data dengan struktur-struktur datapath paralel yang kompak dengan unit-unit multi-fungsional yang dioperasikan untuk instruksi vektor yang sama. Karena sebuah instruksi tunggal menetapkan keseluruhan set operasi-operasi, yang menghemat waktu pengambilan instruksi dan operasi-operasi dekode dibandingkan dengan prosesor tradisional.

11.6.1 Operasi-Operasi Vektor

Sebuah vektor v , adalah merupakan daftar elemen-elemen:

$$v = (v_1, v_2, v_3, \dots, v_n)$$

di mana n adalah jumlah elemen-elemen dalam vektor. Dalam suatu program komputer, vektor adalah sebuah larik satu dimensi. Misalnya, dalam bahasa FORTRAN, kita mendeklarasikan v dengan pernyataan berikut:

$$\text{Dimension} = v(N)$$

Di mana N adalah variabel integer yang menunjukkan *panjang vektor*.

Operasi-operasi aritmetika dapat dikerjakan pada vektor. Dua buah vektor dijumlahkan dengan melakukan penjumlahan pada setiap elemen-elemen yang bersesuaian:

$$Z = x + y = x_1 + y_1, x_2 + y_2, \dots, x_n + y_n$$

Pada FORTRAN, penjumlahan vektor ditetapkan dengan rutin berikut:

```
DO 300 I = 1, N
300   Z(I) = X(I) + Y(I)
```

di mana Z adalah vektor untuk sum (hasil penjumlahan) dan Z , x , dan y dikenal dengan larik dimensi N . Operasi ini disebut *elementwise addition*.

Pada prosesor tradisional (skalar), kompiler menerjemahkan program terlebih dahulu menjadi rentetan instruksi-instruksi berikut:

```
INITIALIZE I = 0
300  READ x(I)
      READ y(I)
      READ N
      ADD Z(I) = x(I) + y(I)
      INCREMENT I = I + 1
      IF I ≤ N GO TO 300
      CONTINUE
```

Ketika program dieksekusi, prosesor skalar melakukan instruksi penjumlahan n kali, untuk menjumlahkan n elemen larik x dengan elemen-elemen yang bersesuaian dengan larik y . Loop dilakukan sebanyak n kali. Jika n lebih besar, waktu yang digunakan untuk pengambilan instruksi dan pendekodean sangatlah signifikan. Pada prosesor vektor, sebuah instruksi vektor mengantikan semua perulangan n kali tersebut.

11.6.2 Register Vektor

Umumnya prosesor vektor mempunyai register-register vektor. Dibandingkan dengan register fungsi umum atau register untuk floating-point yang menyimpan nilai tunggal, register-register vektor menyimpan elemen-elemen vektor secara simultan. Isi suatu register vektor ditransfer ke pipeline vektor satu elemen setiap saat.

11.6.3 Register Skalar

Sebuah register skalar, seperti halnya register fungsi umum atau register bilangan floating-point berfungsi untuk menyimpan sebuah nilai tunggal. Tetapi dia dikonfigurasikan untuk digunakan oleh suatu pipeline vektor. Nilai dalam register dibaca satu kali setiap δ unit waktu dan melepaskannya ke dalam pipeline, yang mirip dengan penerimaan suatu elemen vektor dari pipeline vektor. Misalnya, sebuah komputasi:

$$j = 3.4 \times i$$

Konstanta 3.4 disimpan dalam register skalar dan ditransmisikan ke dalam pipeline perkalian vektor setiap 6 unit waktu untuk berpartisipasi dalam perkalian setiap elemen i .

11.6.4 Memory-Memory Vector Processor

Memory-memory vector processor tidak mempunyai register-register vektor. Dia mengambil (*fetch*) vektor-vektor secara langsung dari memori untuk mengisi pipeline dan menyimpan hasil-hasil secara langsung ke memori. Waktu *start-up* untuk pipeline vektor lebih lama sebab dia menghabiskan waktu untuk memulai sebuah operasi vektor karena waktu akses memori meningkat. Satu contoh dari memory-memory vector processor adalah CDC Cyber 205.

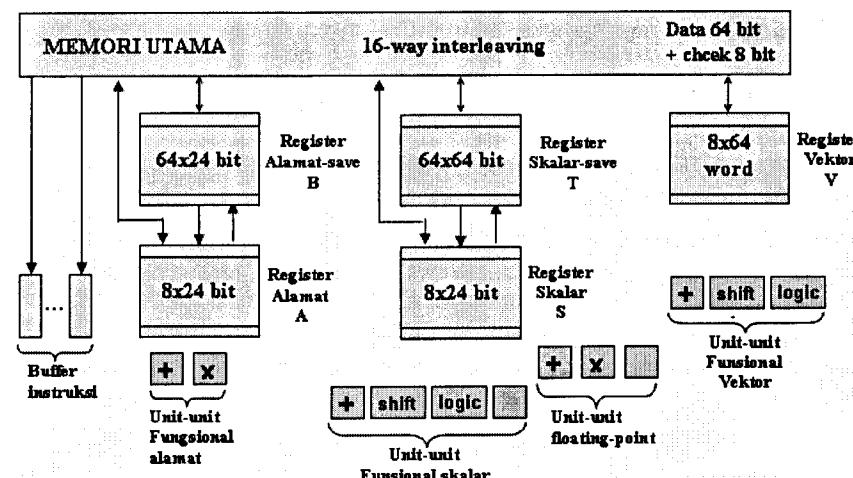
Karena waktu akses vektor berkang dan tumpang tindih antara penyimpanan vektor dan operasi-operasi lainnya, *vector-register vector processor* umumnya lebih efisien daripada memory-memory vector processor. Namun, jika vektor-vektornya panjang, perbedaan efisiensi tidaklah signifikan. Pada sisi lain, memory-memory vector processor menarik jika vektor-vektor sangat panjang.

Contoh Kasus: Cray-1

Cray-1 adalah sebuah superkomputer dengan menggunakan pipeline pada unit aritmetika vektor. Cray-1 adalah seri pertama dari superkomputer Cray. Cray-1 merupakan sebuah mesin vektor-register. Setiap register vektor menyimpan 64 elemen. Setiap elemen atau word mempunyai 64 bit. Cray-1 mempunyai 12 unit fungsional yang di-pipeline. Semua 12 unit dapat beroperasi secara bersamaan. Cray-1 merupakan prosessor vektor pertama yang menggunakan teknik *chaining*. Ada empat kelompok unit-unit fungsional:

1. *Vector pipeline* yang melakukan operasi integer dan logika pada vektor-vektor
2. *Floating-point pipeline* yang mengeksekusi operasi-operasi floating-point menggunakan skalar-skalar atau vektor-vektor.
3. *Scalar pipeline* yang melakukan operasi-operasi integer dan logika pada skalar-skalar.
4. *Address pipeline* yang melakukan kalkulasi alamat.

Gambar 11.24 mengilustrasikan unit-unit fungsional dan register-register. Tidak ada unit pembagi floating-point. Kebalikan floating-point digunakan untuk pembagian, yaitu x/y dikomputasi dengan $x(1/y)$. Ada lima jenis register utama yang dapat diprogram yaitu register A, B, S, T, dan V. Selain itu ada beberapa register pendukung lainnya yaitu Vector Length Register (VL), Vector Mask Register (VM), Program Counter (P), Base Address Register (BA), Limit Address Register (LA), Exchange Address Register (XA), Flag Register (F) dan Mode Register (M). Register 'A' dikenal dengan register alamat. Register ini bekerja seperti register-register alamat untuk referensi memori dan seperti register-register indeks. Di samping itu, ada juga yang digunakan untuk menghitung pergeseran dan menghitung loop. Register 'B' disebut juga register alamat-save dan digunakan sebagai penyimpan pembantu (buffer) untuk register 'A'. Register 'S' dikenal dengan register skalar, digunakan sebagai register sumber dan register tujuan untuk instruksi-instruksi aritmetika skalar dan logika. Register 'T' disebut dengan register skalar-save yang digunakan sebagai penyimpan pembantu untuk register 'S'. Register 'V' yang disebut dengan register vektor, digunakan sebagai register sumber dan register tujuan untuk unit-unit fungsional (pipeline).



Gambar 11.24 Register-register dan unit-unit fungsional Cray-1

Cray-1 membolehkan satu memori membaca dan menulis per siklus clock. Karena itu, dia hanya dapat membaca satu vektor dan menulis satu hasil vektor secara bersamaan. Jika diperlukan membaca lebih dari satu (atau menulis) operasi yang sama, maka salah satu operasi harus ditunda. Misalkan perkalian dua vektor yang panjangnya melebihi 64:

$$z = x \times y$$

Pertama dua register vektor dibaca dari memori: satu 64 elemen pertama dari x , dan yang lainnya 64 elemen pertama dari y . Hasil untuk z masuk ke register vektor yang ketiga. Setelah 64 elemen pertama selesai, register register vektor sumber dibaca kembali dari memori dan isi register vektor hasil disimpan ke memori. Karena hanya satu pembacaan dan satu penulisan yang dieksekusi per siklus, maka elemen-elemen vektor input yang lain tidak dapat dibaca bersamaan, jadi pipeline menunda salah satu operasi pembacaan.

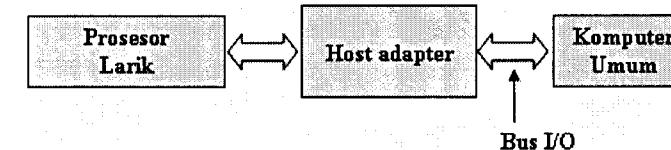
Memori utama yang dimiliki adalah memori yang di-interleave 16-way. Memori ini adalah memori semikonduktor dengan deteksi error dan logic koreksi (SECDED). Terdapat 12 I/O channel. Compiler Fortran Cray-1 merupakan *optimizing compiler*. Pemrogram tidak perlu memodifikasi program-program sumber yang ada karena kompiler menjaga ‘vectorization’ loop DO dan membangkitkan instruksi-instruksi vektor.

11.7 PROSESOR LARIK

Prosesor larik (*array processor*) melakukan komputasi secara simultan pada elemen-elemen dari suatu vektor (larik atau suatu tabel data dalam multi-dimensi). Prosesor larik mengeksekusi suatu rentetan aksi yang dibutuhkan untuk suatu operasi pada keseluruhan larik. Unit-unit multi dalam prosesor larik melakukan operasi yang sama pada data yang berbeda. Penggunaan yang umum dari prosesor larik termasuk analisis dinamika fluida dan rotasi objek 3-dimensi, pemrosesan data cuaca serta *data retrieval*, di mana elemen-elemen dari suatu basis data di-scan secara simultan, termasuk pemrosesan sinyal dan aplikasi-aplikasi dengan persamaan diferensial, manipulasi matriks atau aljabar linier. Prosesor vektor tidak dapat beroperasi pada semua elemen-elemen dari suatu vektor secara bersamaan, dia hanya beroperasi pada elemen-elemen multi secara bersamaan. Prosesor larik dikelompokkan dalam dua jenis:

1. *Dedicated Array Processor*
2. *Attached Array Processor*

Dedicated Array Processor adalah sebuah sistem komputer yang berdiri sendiri, sedangkan *Attached Array Processor* dicantolkan ke sistem komputer umum yang lain sebagai sebuah perluasan. Prosesor ILLIAC IV dan STARAN merupakan dua prosesor larik yang populer. Gambar 11.25 mengilustrasikan pengantarmukaan sebuah *Attached Array Processor* ke suatu komputer umum. FSP-164/MAX adalah sebuah *attached processor* untuk sistem VAX 11. Dia dicantolkan (*interfacing*) ke komputer umum yang menambahkan arsitektur vektor ke komputer umum. *Dedicated Array Processor* termasuk ke dalam arsitektur SIMD, karena itu biasa disebut prosesor larik SIMD.



Gambar 11.25 Pencantolan prosesor larik

Contoh Kasus: ILLIAC IV—Prosesor Larik SIMD

ILLIAC IV adalah sebuah prosesor larik tujuan khusus. Dia didesain dengan mempunyai 256 elemen-elemen pemrosesan (EP) yang diorganisasi dalam empat kuadran 64 EP, masing-masing kuadran 8 x 8 larik EP. Setiap EP dihubungkan dengan tetangganya dalam empat arah. Setiap EP mempunyai memori 2K word. Unit kontrol pusat (CU) menyebarkan instruksi yang dieksekusi oleh EP. Unit kontrol pusat adalah merupakan unit yang sangat kompleks dan canggih dengan sumberdaya yang dimilikinya untuk pengeksekusian instruksi-instruksi skalar tertentu. Dengan kata lain, unit kontrol pusat ekivalen dengan sebuah prosesor. Semua EP melakukan operasi instruksi (katakanlah ADD) secara simultan pada komponen-komponen vektor yang berbeda-beda. ILLIAC IV dihubungkan (link) dengan komputer kontrol B6500 yang bertindak sebagai link antara pengguna dan ILLIAC IV. Secara logika, ILLIAC IV di-link seperti sebuah subsistem peripheral ke B6500.

RINGKASAN

Ada dua strategi dasar yang digunakan untuk meningkatkan kinerja komputer:

1. Tumpang tindih (*overlap*): Memecah tugas menjadi multi-tugas yang dapat dilaksanakan dengan cara tumpang tindih.
2. Paralelisme: Pengeksekusian lebih dari satu tugas secara paralel.

Pada paralelisme level instruksi, paralelisme digunakan pada sebuah prosesor tunggal, sedangkan pada paralelisme level prosesor, multiprosesor melakukan sharing workload (tugas). Hal ini dapat dicapai dengan menggunakan lebih dari satu prosesor dalam sebuah sistem komputer tunggal atau pendistribusian tugas-tugas di antara sejumlah sistem komputer yang di-link seperti cluster atau jaringan.

Organisasi komputer berdasarkan Klasifikasi Flynn dibagi menjadi empat macam:

1. *Single instruction stream, single data stream* (SISD)
2. *Single instruction stream, multiple data stream* (SIMD)
3. *Multiple instruction stream, single data stream* (MISD)
4. *Multiple instruction stream, multiple data stream* (MIMD)

Sebuah prosesor skalar sederhana hanya melakukan satu operasi aritmetika dalam satu waktu. Dia mengeksekusi satu instruksi pada waktu yang diberikan dan setiap instruksi mempunyai satu set operand. Pada prosesor vektor, setiap instruksi mempunyai multi set operand seperti elemen-elemen dua larik dan operasi yang sama dilakukan secara simultan pada set operand yang berbeda. Pada prosesor skalar non-pipeline (sekuensial), tidak ada tumpang tindih siklus-siklus instruksi berurutan. Pada prosesor pipeline, multi-instruksi beroperasi pada berbagai tingkat siklus instruksi secara simultan dalam bagian-bagian prosesor berbeda. Prosesor larik mempunyai unit-unit eksekusi multi yang beroperasi secara simultan pada elemen-elemen sebuah vektor. Prosesor superskalar adalah prosesor skalar yang melakukan siklus instruksi multi secara simultan untuk instruksi-instruksi berurutan.

Pipeline adalah sebuah teknik pemecahan satu tugas menjadi multi sub-tugas dan mengeksekusi sub-tugas. Pipeline instruksi memecah sebuah siklus instruksi menjadi multilangkah yang dieksekusi satu per satu dalam bagian-bagian prosesor yang berbeda. Pipeline aritmetika memecah suatu operasi aritmetika menjadi beberapa langkah aritmetika yang masing-masing dieksekusi satu per satu dalam bagian-bagian ALU

yang berbeda. Instruksi cabang dalam program mengurangi speed-up disebabkan oleh pipeline dan meningkatkan jumlah siklus clock yang dibutuhkan untuk m instruksi. Ketergantungan antara instruksi yang berurutan menyebabkan masalah *hazard* di dalam pipeline dan memengaruhi operasi normal pipeline instruksi. Berikut ada tiga hal utama yang menyebabkan hazard:

1. Konflik sumber daya disebut juga *structural hazard*
2. Ketergantungan data disebut juga *data hazard*
3. Kesulitan pencabangan disebut juga *control hazard*

Prosesor vektor efisien menangani operasi-operasi aritmetika pada elemen-elemen larik (*array*) yang dikenal dengan sebutan ‘vektor’. Untuk sebuah instruksi vektor tunggal, prosesor vektor melakukan operasi-operasi multi-homogen pada suatu larik elemen-elemen. Prosesor vektor menggunakan paralelisme data dengan struktur-struktur datapath paralel yang kompak dengan unit-unit multi-fungsional yang dioperasikan untuk instruksi vektor yang sama. Prosesor larik melakukan komputasi secara simultan pada elemen-elemen dari suatu vektor (larik atau suatu tabel data dalam multi-dimensi). Prosesor larik mengeksekusi suatu rentetan aksi yang dibutuhkan untuk suatu operasi pada keseluruhan larik. Unit-unit multi dalam prosesor larik melakukan operasi yang sama pada data yang berbeda. *Dedicated Array Processor* adalah sebuah sistem komputer yang berdiri sendiri (*stand alone*), sedangkan *Attached Array Processor* dicantolkan ke sistem komputer umum yang lain sebagai sebuah perluasan.

SOAL-SOAL ULANGAN

1. Ada dua strategi dasar yang digunakan untuk meningkatkan kinerja komputer: (1) tumpang tindih (*overlap*): Memecah tugas menjadi multi-tugas yang dapat dilaksanakan dengan cara tumpang tindih, dan (2) _____.
2. Strategi tumpang tindih menggunakan operasi-operasi konkuren melalui (1) unit-unit fungsional, sedangkan paralelisme menggunakan operasi-operasi konkuren melalui (2) _____.
3. Secara garis besar disebutkan ada dua level paralelisme: (1) paralelisme level instruksi, dan (2) _____.
4. Paralelisme yang hanya menggunakan sebuah prosesor tunggal disebut paralelisme level _____.
5. Paralelisme yang menggunakan lebih dari satu prosesor disebut paralelisme level _____.
6. Sistem multiprosesor dan sistem jaringan komputer merupakan contoh paralelisme level _____.
7. Arsitektur superskalar dan arsitektur VLIW (*very long instruction word*) merupakan contoh paralelisme level _____.
8. Klasifikasi Flynn membagi komputer menjadi empat kelompok utama: (1) *single instruction stream, single data stream* (SISD), (2) *single instruction stream, multiple data stream* (SIMD), (3) _____, dan (4) _____.
9. _____ adalah suatu teknik pemecahan satu pekerjaan/tugas menjadi beberapa sub-tugas, dan mengeksekusi sub-tugas tersebut secara bersamaan/paralel dalam unit-unit multi hardware atau segmen-segmen.
10. Tujuan yang ingin dicapai dalam pipeline adalah untuk meningkatkan _____.
11. Pipeline ada dua jenis yaitu: (1) pipeline instruksi, dan (2) _____.
12. Pipeline _____ adalah pipeline yang memecah operasi-operasi siklus instruksi menjadi multilangkah yang dieksekusi satu per satu pada segmen-segmen yang berbeda dari prosesor.
13. Pipeline _____ adalah pipeline yang membagi suatu operasi aritmetika seperti perkalian menjadi langkah-langkah multi-aritmetika yang masing-masing dieksekusi satu per satu pada segmen-segmen aritmetika di dalam ALU.

14. _____ adalah waktu yang digunakan dalam mode non-pipeline dibagi waktu yang digunakan dalam mode pipeline.
15. Ada tiga hal utama yang menyebabkan hazard yaitu: (1) konflik sumberdaya disebut juga *structural hazard*, (2) _____, dan (3) _____.
16. Ketergantungan data diklasifikasikan dalam tiga jenis yaitu: (1) *Read-After-Write* (RAW) hazard, (2) *Write-After-Read* (WAR) hazard, dan (3) _____.
17. _____ digunakan untuk komputasi saintifik kinerja tinggi, biasanya aritmetika vektor dan matriks.

SOAL-SOAL LATIHAN

1. Sebuah pipeline enam-tingkat memproses 200 instruksi, di mana setiap tingkat dalam pipeline membutuhkan waktu yang sama yaitu 10 ns (*cycle time*).
 - a. Berapa jumlah siklus yang digunakan dalam pipeline?
 - b. Berapa total waktu yang dibutuhkan untuk memproses 200 instruksi dalam pipeline?
 - c. Berapa speed up aktual?
 - d. Berapa speed up maksimumnya?
 - e. Berapa throughput?
2. Sebuah program mempunyai 20% instruksi pencabangan. Anggap semua pencabangan berhasil. Hitung speed-up di dalam prosesor pipeline lima tingkat.
3. Sebuah program mempunyai 20% instruksi pencabangan. Bila program ini dijalankan pada prosesor pipeline empat tingkat, diperoleh 60% pencabangan adalah berhasil. Hitung peningkatan kinerja dalam prosesor pipeline. Berapa speed-up maksimum yang memungkinkan jika tidak ada pencabangan?
4. Perhatikan potongan program berikut:

I1:	Load 7,R1	; R1 \leftarrow 7
I2:	Load 5,R2	; R2 \leftarrow 5
I3:	Sub R2,R1,R2	; R2 \leftarrow R2 - R1
I4:	Add R1,R2,R3	; R3 \leftarrow R1 + R2

Gambarkan *space-time diagram* (*Gantt's chart*) menggunakan pipeline lima-tingkat. Tunjukkan dan berikan penjelasan jika terdapat *structural hazard*, *data hazard*, atau *control hazard*.

Catatan: Pipeline 5 tingkat yang dimaksud di atas berturut-turut adalah:
IF, ID, OF, IE, WB
di mana:

IF = pembacaan instruksi dari memori
ID = pendekodean/penerjemahan instruksi
OF = pembacaan operand
IE = pengeksekusian instruksi
WB = penulisan hasil

BAB 12

PENGANTAR SISTEM MULTIPROSESOR

Sasaran bab ini:

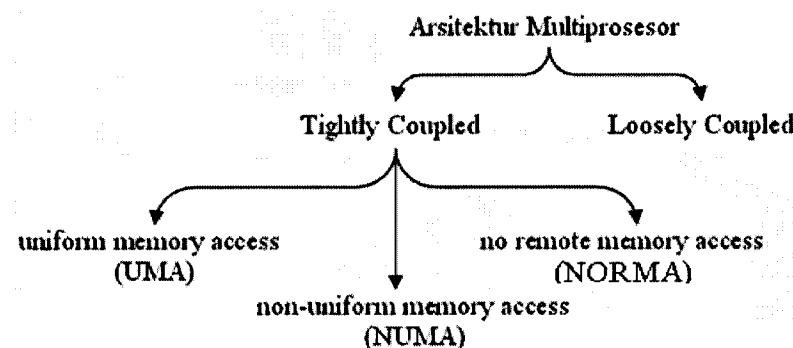
1. Klasifikasi Multiprosesor
2. Multiprosesor Simetris
3. Struktur Interkoneksi Multiprosesor
4. Clustering
5. Fault Tolerance

Pada bagian ini dibahas pengantar multiprosesor, yang merupakan sistem yang terdiri dari dua atau lebih prosesor yang dikoneksikan satu sama lain yang dapat melakukan eksekusi pekerjaan secara bersamaan/paralel. Alasan utama penggunaan multiprosesor adalah untuk membangun komputer yang tangguh dengan cara mengkoneksi-kannya dalam satu area yang lebih kecil. Multiprosesor diharapkan dapat mencapai kecepatan yang lebih baik dibandingkan dengan sebuah uniprosesor yang tercepat. Selain itu sebuah multiprosesor terdiri dari beberapa uniprosesor tunggal yang diharapkan harganya lebih efisien daripada membangun suatu uniprosesor performa tinggi. Keuntungan lain dari sistem multiprosesor adalah karena terdiri dari n prosesor, di mana jika terjadi suatu kegagalan pada satu prosesor, maka masih tersisa $n-1$ prosesor yang akan dapat melanjutkan pelayanan walaupun kinerjanya menjadi berkurang.

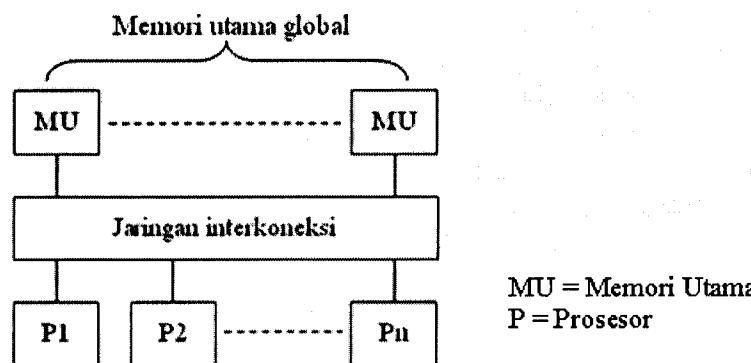
12.1 KLASIFIKASI MULTIPROSESOR

Sistem multiprosesor dapat dibedakan menjadi dua kelompok yaitu *tightly coupled* dan *loosely coupled* seperti yang ditunjukkan pada Gambar 12.1. Pada multiprosesor *tightly coupled*, multiprosesor melakukan sharing informasi melalui sebuah memori bersama (memori global). Pada Gambar 12.2 ditunjukkan tipe ini yang juga dikenal dengan sistem multiprosesor memori sharing. Selain penggunaan memori global bersama, setiap

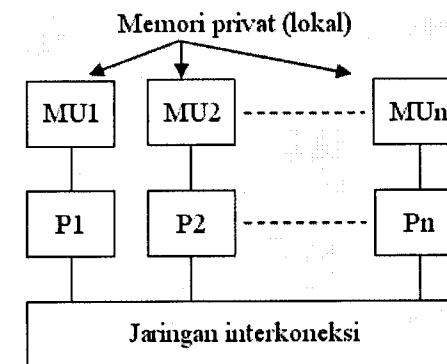
prosesor juga dapat mempunyai memori yang didedikasikan secara lokal yang tidak dapat diakses oleh prosesor lain di dalam sistem tersebut. Pada sistem multiprosesor loosely coupled, memori tidak di-sharing dan setiap prosesor mempunyai memori sendiri (lihat Gambar 12.3).



Gambar 12.1 Arsitektur sistem multiprosesor



Gambar 12.2. Sistem multiprosesor tightly coupled



Gambar 12.3. Sistem multiprosesor loosely coupled

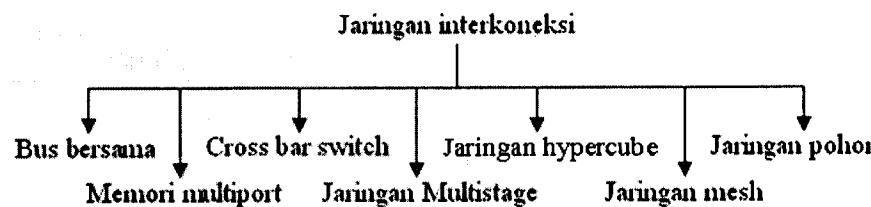
12.2 MULTIPROSESOR SIMETRIS

Pada sistem UMA (*uniform memory access*), waktu akses memori sama untuk semua prosesor. Multiprosesor simetris (*symmetric multiprocessor*, SMP) adalah sebuah sistem multiprosesor UMA dengan prosesor-prosesor yang identik, semuanya mampu melakukan fungsi-fungsi yang sama dengan cara yang identik. Semua prosesor mempunyai waktu akses yang sama untuk sumber daya memori dan I/O. Untuk sistem operasi, semua prosesor adalah sama dan prosesor mana saja dapat mengeksekusinya. Istilah UMA dan SMP dapat digunakan secara bergantian.

Sistem multiprosesor loosely coupled mempunyai memori terdistribusi secara fisik sesuai dengan prosesor. Ini ada dua jenis: Multiprosesor DSM (*distributed shared memory*) dan cluster. Pada multiprosesor DSM, prosesor mempunyai ruang alamat bersama atau *shared* untuk semua memori. Multiprosesor DSM disebut juga dengan mesin NUMA (*non-uniform memory access*). Pada sistem ini, waktu akses memori berbeda untuk prosesor yang berbeda. Sedangkan pada *cluster*, tidak ada sharing ruang alamat oleh prosesor.

12.3 STRUKTUR INTERKONEKSI

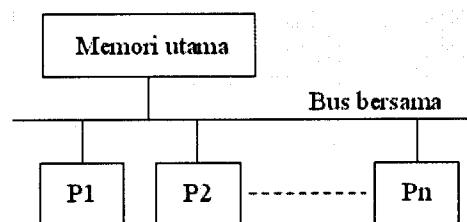
Struktur interkoneksi yang digunakan dalam sistem multiprosesor dapat bermacam-macam seperti yang ditunjukkan pada Gambar 12.4.



Gambar 12.4. Struktur interkoneksi multiprosesor

12.3.1 Struktur Bus Bersama

Struktur bus bersama disebut juga struktur bus time-shared , ilustrasinya ditunjukkan pada Gambar 12.5. Semua prosesor, memori dan subsistem I/O dihubungkan ke bus. Di samping bus bersama, setiap prosessor terdiri atas bus lokal yang digunakan untuk melakukan komunikasi dengan memori lokal dan I/O lokal.



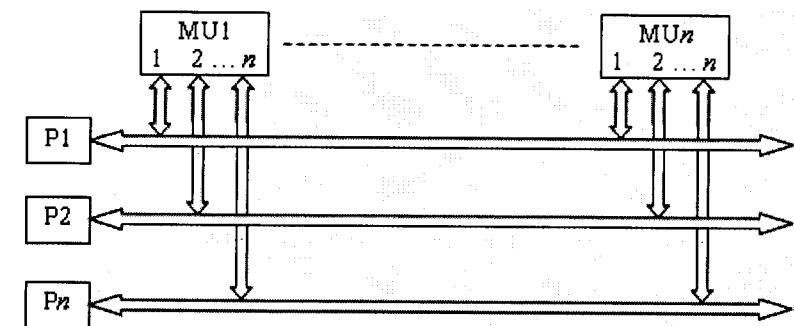
Gambar 12.5. Interkoneksi bus bersama time-shared

Keuntungannya: strukturnya sederhana dan relatif murah. Kelemahannya: sangat lambat karena dalam satu waktu hanya ada satu transfer/komunikasi yang terjadi pada bus, misalnya jika satu prosesor mengakses memori maka prosesor lainnya tidak dapat melakukan suatu operasi pada bus.

12.3.2 Struktur Memori Multiport

Pada sistem memori multiport, terdapat bus yang terpisah antara masing-masing modul memori dan prosesor. Gambar 12.6 menunjukkan sebuah sistem dengan n prosesor dan n modul memori, setiap modul memori mempunyai n port. Setiap modul memori mempunyai logika

prioritas yang mengatasi konflik yang terjadi ketika ada permintaan yang bersamaan dari prosesor-prosesor lainnya.

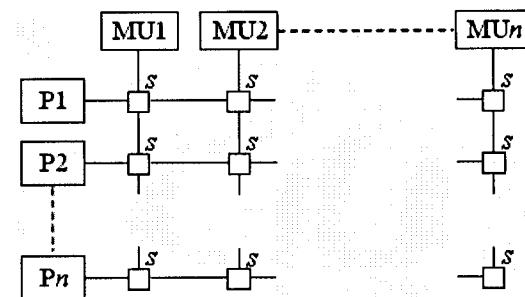


Gambar 12.6. Interkoneksi memori multiport

Keuntungannya: karena dengan adanya multipath maka prosesor-prosesor dapat secara simultan/bersamaan mengakses memori sehingga kecepatan transfernya menjadi besar. Kelemahannya: dibutuhkan hardware yang lebih besar di dalam modul memori dan dibutuhkan banyak kabel interkoneksi untuk prosesor yang banyak. Cara ini cocok untuk sistem yang kecil.

12.3.3 Struktur Crossbar Switch

Struktur crossbar switch mirip dengan crossbar telephone exchange. Gambar 12.7 menunjukkan interkoneksi crossbar switch untuk sebuah sistem dengan n prosesor dan n modul memori. Setiap s (switch) di dalam Gambar 12.7 sebenarnya adalah sebuah sirkuit elektronika yang menyediakan jalur yang diinginkan. Sistem ini mendukung logika prioritas untuk mengatasi konflik.



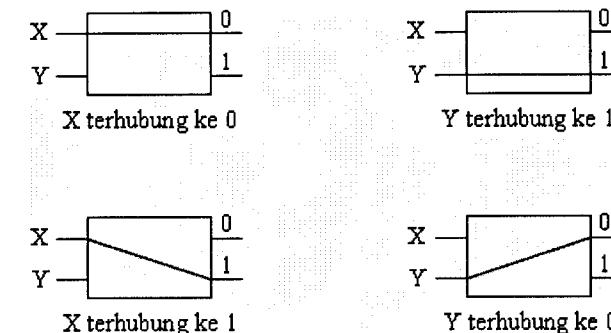
Gambar 12.7. Interkoneksi crossbar switch

Keuntungannya: karena setiap modul memori mempunyai jalur terpisah, maka semua modul memori dapat melakukan komunikasi secara simultan. Kelemahannya: diperlukan hardware yang kompleks bila jumlah prosesor menjadi banyak.

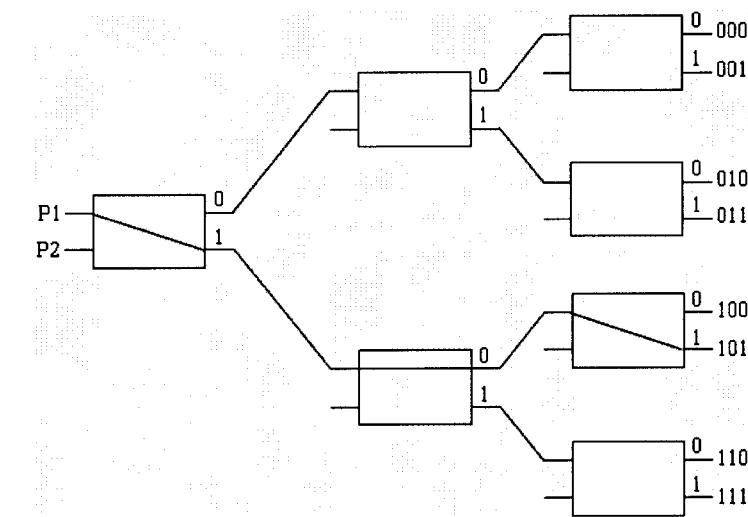
12.3.4 Struktur Jaringan Multistage

Pada struktur jaringan *multistage*, terdapat lebih dari satu tingkat switch elektronika (bandingkan pada struktur crossbar switch yang hanya mempunyai satu tingkat) yang digunakan untuk set-up jalur antara prosesor dan modul memori. Setiap switch mempunyai dua input dan dua output. Sebuah input dapat dihubungkan ke sembarang output.

Komponen dasar jaringan multistage adalah *interchange switch* dua-input dan dua-output. Pada Gambar 12.8 ditunjukkan switch 2x2 yang mempunyai dua input dengan label X dan Y dan dua output dengan label 0 dan 1. Ada sinyal-sinyal kontrol (tidak nampak) yang bersama dengan switch yang membangun interkoneksi antara terminal-terminal input dan output. Switch mempunyai kemampuan menghubungkan input X ke salah satu output. Demikian halnya dengan terminal Y mempunyai cara yang sama. Switch juga mempunyai kemampuan melakukan arbitrasi antara permintaan-permintaan yang konflik. Jika input X dan Y keduanya melakukan permintaan terminal output yang sama, maka hanya salah satu saja yang akan dihubungkan, yang lainnya diblok.



Gambar 12.8. Operasi interchange switch 2 x 2

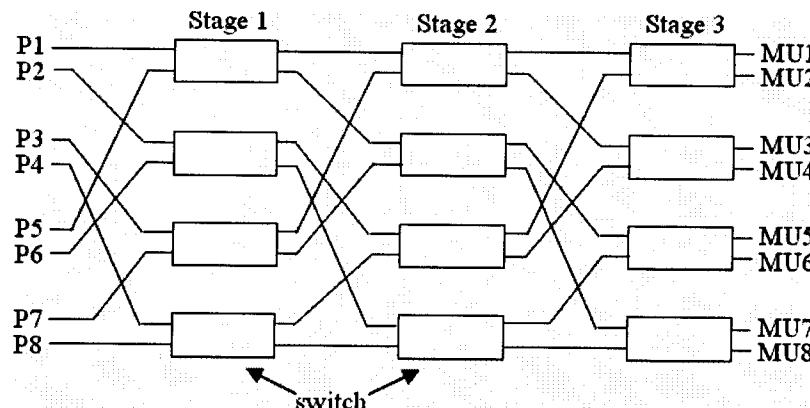


Gambar 12.9. Pohon biner dengan switch 2 x 2

Penggunaan switch 2x2 sebagai blok pembangun, memungkinkan untuk membangun sebuah jaringan multistage untuk mengontrol komunikasi antara sejumlah sumber dan target/tujuan. Untuk melihat bagaimana hal ini dilakukan, pikirkan pohon biner yang ditunjukkan pada Gambar 12.9. Dua prosesor P1 dan P2 dihubungkan melalui switch ke delapan modul memori yang diberi tanda biner dari 000 sampai 111. Lintasan dari suatu sumber ke suatu target ditetapkan dari bit-bit biner bilangan target. Bit

pertama bilangan target menetapkan keluaran switch pada level pertama. Bit kedua menetapkan keluaran switch pada level kedua, dan bit ketiga menetapkan keluaran switch pada level ketiga. Misalnya, untuk menghubungkan P1 ke memori 101, perlu dibentuk sebuah lintasan dari P1 ke output 1 pada switch level pertama, output 0 pada switch level kedua dan output 1 pada switch level ketiga. Jadi, jelaslah bahwa P1 atau P2 dapat dihubungkan ke salah satu dari delapan memori. Misalnya, jika P1 dihubungkan ke salah satu tujuan 000 sampai 011, maka P2 hanya dapat dihubungkan ke salah satu tujuan 100 sampai 111.

Beberapa skema memungkinkan untuk interkoneksi melalui switch. Misalnya, jaringan omega switching yang merupakan jaringan yang populer (Gambar 12.10).



Gambar 12.10. Jaringan 8x8 omega multistage switching

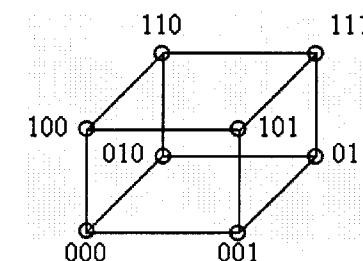
Keuntungannya: mendukung struktur pembiayaan efektif. Kelemahannya: hanya memungkinkan beberapa komunikasi simultan yang terbatas.

12.3.5 Struktur Jaringan Hypercube

Hypercube adalah sebuah kubus n -dimensi, digunakan untuk menghubungkan 2^n prosesor dengan cara loosely coupled. Setiap simpul pada hypercube merepresentasikan sebuah prosesor. Gambar 12.11 menunjukkan sebuah hypercube 3-dimensi yang terhubung dengan delapan prosesor yang diberi nomor 000 sampai 111. Setiap simpul tidak hanya bertindak sebagai prosesor, tetapi juga mendukung sejumlah memori lokal dan I/O

untuk prosesor. Pinggiran kubus terhubung ke jalur komunikasi. Setiap prosesor mempunyai sebuah jalur komunikasi sendiri ke prosesor tetangganya melalui pinggiran/tепи.

Setiap simpul ditetapkan sebagai sebuah alamat biner di mana alamat dari dua simpul yang bertetangga hanya berbeda satu posisi bit. Misalnya, tiga simpul yang bertetangga dengan alamat 000 dalam struktur tiga-kubus adalah 001, 010 dan 100. Setiap bilangan biner ini berbeda dari alamat 000 dengan nilai satu bit



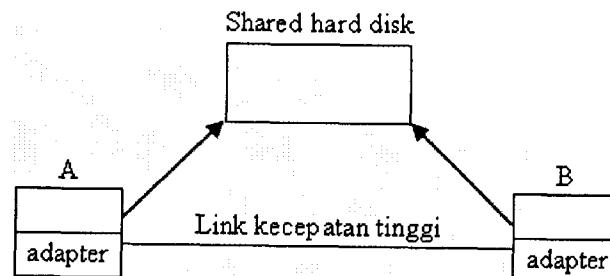
Gambar 12.11. Interkoneksi hypercube 3-dimensi

Keuntungannya: protokol cerdas komunikasi dapat diimplementasikan lebih mudah. Kelemahannya: karena terdapat multi rute antar prosesor, maka kompleksitas routing meningkat.

12.4 CLUSTER

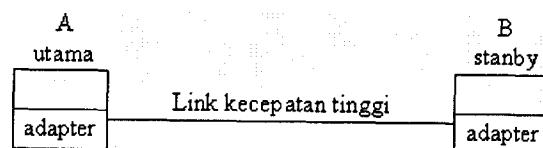
Clustering adalah suatu interkoneksi sistem komputer multi yang independen yang dioperasikan seperti sebuah sistem tunggal dalam suatu kerjasama/kolaborasi. Cara ini mengantikan sistem multi prosesor di mana prosesor-prosesor jamak berada dalam suatu sistem komputer tunggal. Setiap simpul (sistem) dalam suatu cluster dapat juga bekerja secara independen. Cluster berbeda dengan sebuah jaringan komputer yang mempunyai tujuan utama yaitu "resource sharing". Sebuah cluster menyediakan semua keuntungan dari sebuah sistem multiprosesor. Ada dua tujuan utama di belakang formasi suatu cluster:

1. *Load sharing*: Di sini, dua sistem A dan B membentuk sebuah cluster dan sharing beban pemrosesan. Lebih dari dua sistem juga dapat membentuk sebuah cluster (lihat Gambar 12.12).



Gambar 12.12. clustering Clustering untuk load sharing

2. *Fault tolerance*: Pada kasus ini, sistem B adalah sebuah *hot standby* untuk sistem A. Ketika terjadi kegagalan dalam sistem A, sistem B mengambil alih fungsi tersebut. Bila sistem A bekerja dengan benar, sistem B diam memonitor status normal dari sistem A dengan melakukan analisis "detak jantung" yang diterimanya. Jika detak jantung tidak ada atau korup, maka sistem B mulai bekerja (lihat Gambar 12.13).



Gambar 12.13. clustering Clustering untuk fault tolerance

Pada kedua jenis clustering, software sistem berfungsi mengatur sistem secara tepat.

12.5 MASALAH CACHE COHERENCE

Memori cache pada dasarnya merupakan fitur sistem komputer yang berfungsi mengurangi waktu akses memori utama pada waktu siklus instruksi. Beberapa prosesor menganut "*write-through policy*" sementara yang lainnya mengikuti "*write-back policy*". Ketika lebih dari satu prosesor menggunakan secara sharing sebuah memori utama, salinannya berada dalam memori-memori cache semua prosesor. Ini penting ketika satu prosesor menulis dalam memori cache- nya, tindakan yang tepat diambil

agar informasi yang lama, dan yang tidak valid pada memori cache lainnya tidak digunakan oleh yang lain. Pikirkan dua prosesor A dan B, masing-masing dengan memori cache yang terpisah. Anggap sebuah lokasi memori 1000 dipetakan ke memori-memori cache kedua prosesor tersebut dan nilainya adalah s. Sekarang, jika A mengubahnya menjadi N di dalam memori cache- nya dan B tidak menyadarinya/mengetahuinya, maka ketika membaca lokasi memori 1000 dia menerima s dari memori cache. Jelaslah hal ini tidak dikehendaki. *Cache coherence* didefinisikan sebagai suatu kondisi di mana semua baris cache dari sebuah blok memori yang di-share berisi informasi yang sama pada suatu waktu. Masalah cache coherence terjadi jika sebuah prosesor mengubah cache- nya tanpa menjaga keseragaman pada semua cache. Ini merupakan suatu masalah yang dihadapi jika suatu sistem mengizinkan prosesor-prosesor untuk mengakses salinan-salinan data. Metoda Metode hardware dan software digunakan sebagai suatu solusi untuk masalah cache coherence. Metoda Metode software adalah metoda metoda yang murah tetapi lambat, sedangkan metoda metode hardware cepat tetapi mahal.

12.6 FAULT TOLERANCE

Fault tolerance adalah kemampuan suatu komputer untuk mengeksekusi operasi-operasinya secara benar tanpa terpengaruh oleh kegagalan hardware atau software. Untuk membuat suatu sistem fault tolerance, maka hardware dan/atau modul-modul software ditambahkan pada arsitektur tersebut. Saat ini teknik-teknik fault tolerance sederhana tersedia pada komputer-komputer murah. Beberapa teknik tersebut antara lain:

1. *Error detection/correction* menggunakan *parity check*, *humming code*, *cycle redundancy check* dan sebagainya sehingga kegagalan ketika transfer data dan penyimpanan data terdeteksi.
2. Replikasi modul hardware (seperti ALU) untuk melakukan suatu fungsi dengan paralel oleh dua (atau lebih) modul dan membandingkan hasil-hasilnya.
3. Menggunakan tiga modul yang identik (seperti prosesor) untuk melakukan suatu fungsi dan memilih hasil-hasil "utama".

Komputer khusus fault tolerance didesain dengan sejumlah modul hardware yang banyak dan jalur-jalur alternatif agar mereka dapat berfungsi tanpa menyebabkan kerusakan. Namun, level kinerja nya menurun

setelah sebuah kegagalan, dibandingkan dengan level normal. Keadaan ini disebut dengan "*graceful degradation*" karena komputer fault tolerance bekerja secara benar (tetapi dengan efisiensi yang berkurang) sekalipun telah mengalami kegagalan. Komputer tandem merupakan contoh komputer fault tolerance yang unggul/baik.

RINGKASAN

Sistem multiprosesor terdiri dari dua atau lebih prosesor yang dikoneksikan satu sama lain yang dapat melakukan eksekusi pekerjaan secara bersamaan/paralel. Alasan utama penggunaan sistem multiprosesor adalah membangun komputer yang tangguh untuk meningkatkan kinerja dengan menggunakan pemrosesan paralel. Juga menawarkan fault tolerance. Pada sistem multiprosesor tightly coupled, multiprosesor menggunakan memori bersama secara sharing. Prosesor melakukan share informasi melalui memori bersama. Pada sistem multiprosesor loosely coupled, tidak ada memori bersama/shared dan setiap prosesor mempunyai memori privat sendiri-sendiri. Informasi dipertukarkan dengan prosesor lain melalui jaringan interkoneksi dengan sebuah protokol message passing. Multiprosesor simetris adalah suatu sistem multiprosesor dengan prosesor yang identik dengan kemampuan yang sama. Semua prosesor mempunyai waktu akses yang sama ke sumber daya memori dan I/O.

Sistem multiprosesor loosely coupled mempunyai memori terdistribusi secara fisik. Ada dua jenis: multiprosesor memori shared terdistribusi dan cluster. Pada memori shared terdistribusi, prosesor mempunyai memori bersama, ruang alamat di-share untuk semua memori. Pada cluster, prosesor tidak melakukan sharing terhadap ruang alamat. Clustering merupakan interkoneksi sistem komputer jamak yang independen yang fungsinya sebagai sistem tunggal dengan cara kolaborasi/kerjasama. Dua tujuan utama pembentukan suatu cluster adalah: load sharing dan fault tolerance.

Bila lebih dari satu prosesor menggunakan (*sharing*) memori utama, salinan dari blok memori utama berada di dalam memori cache semua prosesor. Masalah cache coherence terjadi jika setiap prosesor diizinkan mengubah cache-nya tanpa tindakan pencegahan untuk menjaga keseragaman di dalam semua cache. Metoda Metode hardware dan software dapat digunakan sebagai suatu solusi dalam masalah cache coherence.

Fault tolerance membolehkan sebuah komputer mengeksekusi operasi-operasinya dengan baik tanpa terpengaruh oleh kegagalan hardware atau software. Untuk membuat suatu sistem fault tolerance, maka hardware dan/atau modul-modul software ditambahkan pada arsitektur tersebut.

Saat ini teknik-teknik fault tolerance sederhana tersedia pada komputer-komputer murah.

SOAL-SOAL ULANGAN

1. Sistem multiprosesor dapat dibedakan menjadi dua kelompok yaitu: (1) *tightly coupled*, dan (2) _____.
2. Pada sistem multiprosesor _____, memori tidak di-sharing dan setiap prosesor mempunyai memori sendiri.
3. Pada multiprosesor _____, multiprosesor melakukan sharing informasi melalui sebuah memori bersama (memori global).
4. Sistem multiprosesor *tightly coupled* dibagi menjadi tiga yaitu: (1) *uniform memory access*, (2) *non-uniform memory access*, dan (3) _____.
5. Struktur interkoneksi multiprosesor terdiri dariterdiri atas (1) bus bersama, (2) memori multiport, (3) cross bar switch , (4) jaringan multistage, (5) jaringan mesh, (6) jaringan pohon, dan (7) _____.
6. Keuntungan struktur interkoneksi _____ karena dengan adanya multipath maka prosesor-prosesor dapat secara simultan/bersamaan mengakses memori sehingga kecepatan transfernya menjadi besar. Kelemahannya membutuhkan hardware yang besar di dalam modul memori.
7. Keuntungan struktur interkoneksi _____ karena setiap modul memori mempunyai jalur terpisah, maka semua modul memori dapat melakukan komunikasi secara simultan. Kelemahannya: diperlukan hardware yang kompleks bila jumlah prosesor menjadi banyak.
8. _____ adalah suatu interkoneksi sistem komputer multi yang independen yang dioperasikan seperti sebuah sistem tunggal dalam suatu kerjasama/kolaborasi.
9. _____ didefinisikan didefinisikan sebagai suatu kondisi di mana semua baris cache dari sebuah blok memori yang di-share berisi informasi yang sama pada suatu waktu.
10. _____ adalah kemampuan suatu komputer untuk mengeksekusi operasi-operasinya secara benar tanpa terpengaruh oleh kegagalan hardware atau software.

SOAL-SOAL LATIHAN

1. Berapa banyak titik switch yang berada dalam sebuah jaringan crossbar switch yang menghubungkan p prosesor ke m modul memori?
2. Jaringan omega switching 8×8 pada Gambar 12.10 mempunyai tiga stage dengan empat switch pada setiap stage, dengan total 12 switch . Berapa banyaknya stage dan switch per stage yang diperlukan pada sebuah jaringan omega switch ing $n \times n$?
3. Anggap kawat terputus antara switch pada baris pertama, kolom kedua dan switch pada baris kedua, kolom ketiga dalam jaringan omega switch ing Gambar 12.10. Lintasan manakah yang akan terputus?
4. Bangunlah sebuah diagram untuk jaringan omega swirching switching 4×4 . Tunjukkan switch setting yang diperlukan untuk menghubungkan masukan 3 ke keluaran 1.
5. Bangunlah (gambaran) sebuah diagram yang menunjukkan struktur sebuah hipercube jaringan empat-dimensi. Lengapi semua lintasan (*path*) nomor alamat antar simpul yang bertetangga (dengan perbedaan satu bit satu sama lain).

DAFTAR PUSTAKA

- Abd-El-Barr, Mostafa, El-Rewini, Hesham. 2005. *Fundamentals of Computer Organization and Architecture*. New Jersey: John Wiley & Sons.
- Govindarajalu, B. 2004. *Computer Architecture and Organization, Design Principles and Applications*. New Delhi: Tata McGraw-Hill.
- Hamacher, V. Carl; Vranesic, Zvonko G.; Zaky, Safwat G. 1996. *Computer Organization*. Fourth edition. Singapore: McGraw-Hill.
- Leach, Donald P., Malvino, Albert Paul. 1995. *Digital Principles and Applications*. Fifth edition, Glencoe, USA: McGraw-Hill.
- Mano, M. Morris. 1993. *Computer System Architecture*, third edition. USA: Prentice-Hall.
- Murdocka, Miles J., Heuring, Vincent P. 1999. *Principles of Computer Architecture*, Class test edition. USA: Prentice-Hall.
- Null, Linda, Lobur, Julia. 2003. *The Essentials of Computer Organization and Architecture*. USA: Jones and Bartlett Publishers Inc.
- Stallings William. 2003. *Computer Organization and Architecture, Designing For Performance*, sixth edition. USA: Prentice-Hall by Pearson Education Inc
- Tocci, Ronald J., Widmer, Neal S. 2001. *Digital Systems, Principles and Applications*. Eight edition. New Jersey: Prentice-Hall.

LAMPIRAN A

SIRKUIT TERINTEGRASI DIGITAL

Sirkuit terintegrasi (*integrated circuit, IC*) adalah sebuah sirkuit yang terdiri atas beberapa komponen yang disusun di dalam sebuah chip. Seperti yang telah dibahas sebelumnya pada Bab 4, sebuah IC berisi beberapa gerbang (*gate*) dengan jenis yang sama. Gerbang adalah sebuah sirkuit logika dengan satu atau lebih saluran masuk dan satu saluran keluar. Pada lampiran ini akan dibahas secara singkat topik-topik yang mendasar dalam elektronika digital.

A.1 KLASIFIKASI IC

IC diklasifikasikan dalam level-level integrasi seperti yang ditunjukkan pada Tabel A.1 bergantung pada jumlah gerbang yang dapat ditempatkan dalam sebuah IC.

TABEL A.1 Klasifikasi IC

Level Integrasi	Jumlah Gerbang (Kompleksitas IC)	Ekivalensi Memori (bit)	Produk Khas
SSI (<i>small scale integration</i>)	< 10	—	Quad 2 input NAND gate
MSI (<i>medium scale integration</i>)	10 – 100	< 1K	Flip-flop JK
LSI (<i>large scale integration</i>)	100 – 500	1 – 16K	Programmable controller
VLSI (<i>very large scale integration</i>)	5000 – 50,000	> 16K	Mikroprosesor 80286
ULSI (<i>ultra large scale integration</i>)	> 50,000	> 256K	RAM 256K

Beberapa keluarga IC yang telah dikembangkan adalah TTL (*transistor transistor logic*), ECL (*emitter coupled logic*) dan keluarga MOS (*metal oxide semiconductor*) yang populer saat ini. ECL adalah teknologi IC yang mahal, yang dipilih ketika diperlukan kecepatan yang ekstra tinggi. Teknologi TTL

lebih cepat daripada teknologi MOS tetapi konsumsi dayanya lebih besar dan menempati ruang yang lebih besar untuk sirkuitnya.

Sirkuit digital ada dua macam yaitu sirkuit kombinasional dan sirkuit sekuensial. Output sirkuit kombinasional pada seketika, hanya bergantung pada input. Gerbang (gate), multiplexer, demultiplexer, dan decoder merupakan beberapa contoh sirkuit-sirkuit kombinasional. Output sirkuit sekuensial bergantung pada keadaan input sebelumnya dan pada saat itu. Flip-flop, counter, dan register merupakan beberapa contoh sirkuit-sirkuit sekuensial.

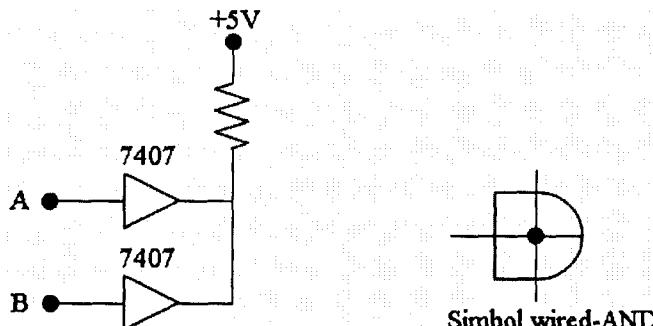
Sinyal digital disebut sinyal periodik jika sinyal tersebut berulang dengan kecepatan konstan. Sinyal clock adalah sebuah bentuk gelombang periodik yang mempunyai sifat yang telah dibahas pada Bab 1.

A.1.1 IC Kolektor Terbuka

Sirkuit TTL kolektor terbuka merupakan sirkuit orisinal yang dikembangkan untuk aplikasi bus. Keuntungan sirkuit kolektor terbuka karena merupakan fungsi wired-AND yang dibentuk dengan menghubungkan multi output secara bersama. Gambar A.1 menunjukkan dua gerbang 7407 (buffer dengan output kolektor terbuka) yang mempunyai output yang diikat bersama. Nilai resistor pull up bergantung pada dua faktor:

1. Jumlah output gerbang yang dihubungkan bersama.
2. Jumlah beban yang dikemudikan oleh output gerbang.

Formula untuk menghitung nilai resistor pull up berdasarkan pada faktor-faktor di atas diberikan dalam *data sheet/data book* IC.



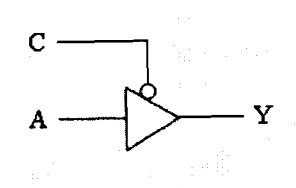
Gambar A.1 Keluaran kolektor terbuka pada bus

A.1.2 Gerbang Tri-State

Gerbang logika tri-state mempunyai tiga keadaan output:

1. **Low**: seperti keadaan 0 TTL biasa
2. **High**: seperti keadaan 1 TTL biasa
3. **High impedance**: bukan *high* dan bukan juga *low*

Gerbang *tri-state* mempunyai sebuah input kontrol yang terpisah. Input ini memutuskan apakah output akan di-enable (0 atau 1) atau tri-state (*disable*). Pada keadaan output mengambang, gerbang ini akan tampak sebagai sirkuit terbuka terhadap beban. Gambar A.2 menunjukkan gerbang penyanga *tri-state* yang diberikan oleh sebuah IC 74125.



(a) simbol

C	A	Y
0	0	0
0	1	1
1	0	Tri-state
1	1	Tri-state

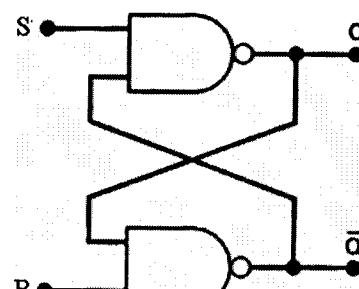
(b) tabel kebenaran

Gambar A.2 Penyangga (gerbang) *tri-state*

Gerbang *tri-state* berguna dalam implementasi struktur bus. Pada sebuah saluran bersama, banyak gerbang keluaran *tri-state* yang dihubungkan. Pada satu waktu, hanya salah satu gerbang yang di-enable dan keluaran gerbang yang di-enable adalah 1 atau 0. Semua gerbang lain di-disable, output gerbang-gerbang ini berada dalam keadaan impedansi tinggi (*high impedance*). Gerbang-gerbang ini secara logika diputus dari saluran bus dan mereka tidak menarik arus dari beban.

A.2 LATCH

Sebuah latch mengunci beberapa informasi sinyal. Bila latch dalam keadaan terbuka, maka sinyal input lewat melalui latch dan mencapai output. Bila latch dalam keadaan tertutup, maka sinyal tidak dapat melewati sirkuit latch. Gambar A.3 menunjukkan sirkuit untuk latch SR menggunakan dua gerbang NAND dan tabel kebenarannya.



(a) Sirkuit Latch NAND

S	R	Output
1	1	No change
0	1	$Q = 1$
1	0	$Q = 0$
0	0	Invalid*

*produces $Q = \bar{Q} = 1$

(b) Tabel Kebenaran Latch NAND

Gambar A.3 Latch NAND

Operasi yang digambarkan di atas dapat diringkaskan sebagai berikut:

1. $S = R = 1$. Kondisi ini merupakan keadaan tetap, tidak berpengaruh pada keadaan output.
2. $S = 0, R = 1$. Keadaan ini akan selalu menyebabkan output dalam keadaan $Q = 1$. Keadaan ini disebut "set".
3. $S = 1, R = 0$. Keadaan ini akan selalu menyebabkan output dalam keadaan $Q = 0$. Keadaan ini disebut "reset" atau "clear"
4. $S = R = 0$. Keadaan ini mencoba men-set dan men-clear latch pada waktu bersamaan dan dapat menghasilkan keadaan yang ambigu. Keadaan ini tidak digunakan.

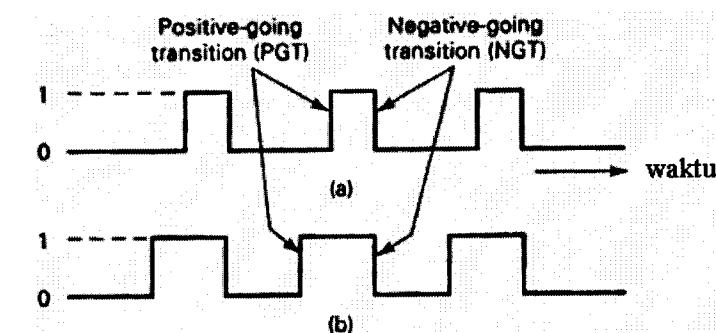
A.3 SINYAL CLOCK DAN FLIP-FLOP

Sistem digital dapat beroperasi dengan cara asinkron atau sinkron. Pada sistem asinkron, output sirkuit logika dapat berubah keadaannya kapan saja pada satu atau lebih perubahan input. Sistem asinkron umumnya lebih sulit dilakukan desain dan troubleshoot daripada sistem sinkron.

Pada sistem sinkron, waktu yang tepat di mana output dapat berubah keadaan ditentukan oleh sebuah sinyal yang disebut **clock**. Transisi (disebut juga tebing) ditunjukkan pada Gambar A.4. Bila perubahan clock dari 0 ke 1, ini disebut **positive-going transition (PGT)**; bila perubahan clock dari 1 ke 0, ini disebut **negative-going transition (NGT)**. Kita akan gunakan singkatan

PGT dan NGT karena akan sering muncul di dalam pembahasan kita ke depan.

Kebanyakan sistem digital pada prinsipnya adalah sinkron (walaupun selalu ada beberapa bagian-bagian asinkron), karena sirkuit sinkron lebih mudah untuk melakukan desain dan troubleshoot.



Gambar A.4 Sinyal clock

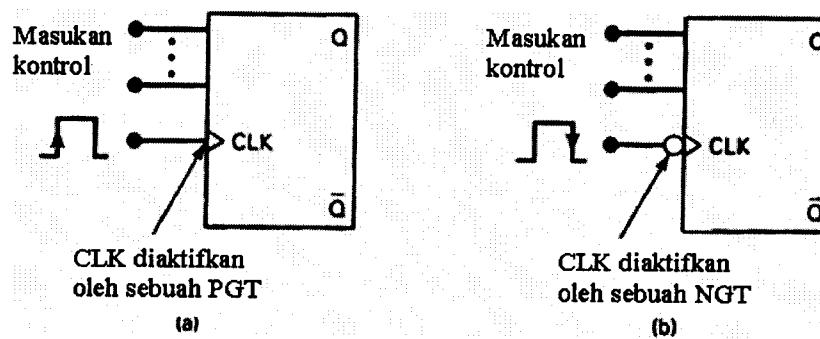
A.4 FLIP-FLOP YANG MEMPUNYAI CLOCK

Beberapa jenis flip-flop (FF) digunakan dalam aplikasi yang luas. Sebelum kita memulai membahas berbagai FF, kita akan membahas gagasan-gagasan prinsip yang umum untuk semuanya:

1. Flip-Flop mempunyai sebuah masukan clock yang secara khas ditandai dengan CLK, CK, atau CP. Kita akan menggunakan CLK, seperti yang ditunjukkan pada Gambar A.5. Kebanyakan masukan CLK dari FF adalah **edged-triggered**, yang berarti diaktifkan oleh sebuah transisi sinyal, hal ini ditunjukkan oleh adanya sebuah segitiga pada masukan CLK. Hal ini berbeda dengan latch, yang merupakan level-triggered.

Gambar A.5(a) adalah sebuah FF dengan segitiga kecil pada masukan CLK nya menunjukkan masukan ini diaktifkan hanya jika terjadi sebuah PGT, tidak ada bagian lain dari pulsa masukan akan mempunyai efek pada masukan CLK. Pada Gambar A.5(b) simbol FF mempunyai bulatan dan segitiga pada masukan CLK. Bulatan ini mengindikasikan bahwa masukan CLK hanya akan aktif jika terjadi

- sebuah NGT. Tidak ada bagian lain dari pulsa masukan yang akan mempunyai efek pada masukan CLK.
- Flip-flop juga mempunyai satu atau lebih masukan kontrol yang dapat mempunyai nama yang bervariasi, bergantung pada operasinya. Masukan-masukan kontrol tidak akan mempunyai efek pada Q hingga terjadi transisi clock aktif. Dengan kata lain, efek ini disinkronisasi dengan sinyal yang diberikan ke CLK. Dengan alasan inilah mereka disebut masukan kontrol sinkron. Misalnya, masukan kontrol FF pada Gambar A.5(a) tidak akan mempunyai efek pada Q hingga terjadi sinyal clock PGT. Sebaliknya, masukan kontrol pada Gambar A.5(b) tidak akan mempunyai efek pada Q hingga terjadi sinyal clock NGT.



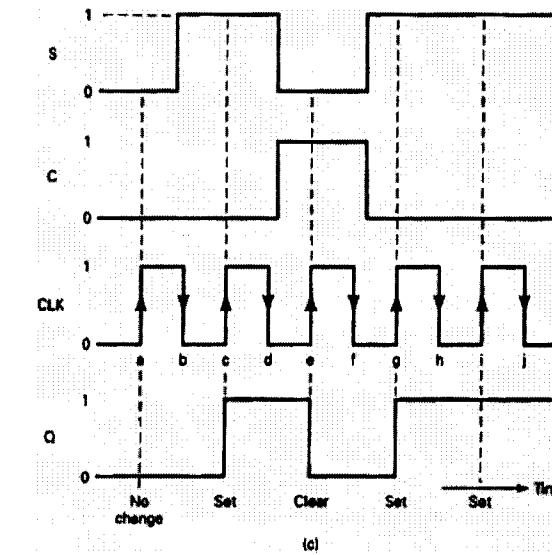
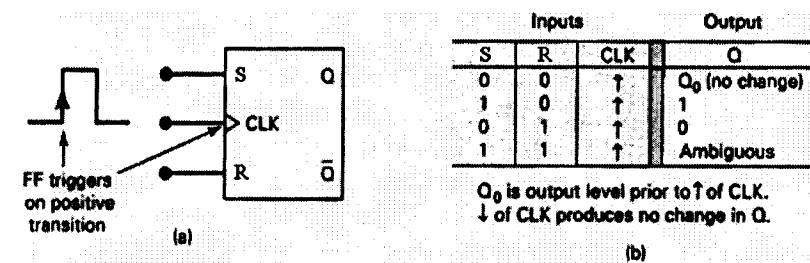
Gambar A.5 flip-flop mempunyai sebuah masukan clock (CLK) yang aktif (a) PGT atau (b) NGT. Masukan-masukan kontrol menentukan efek dari transisi clock aktif.

A.4.1 SR Flip-Flop

Gambar A.6 menunjukkan simbol logika untuk SR flip flop yang dipicu oleh sinyal clock PGT. Hal ini berarti FF dapat berubah keadaan hanya bila sebuah sinyal diberikan pada masukan clock yang membuat transisi dari 0 ke 1.

Tabel kebenaran pada Gambar A.6(b) menunjukkan bagaimana keluaran FF akan merespons PGT pada masukan CLK untuk berbagai kombinasi masukan S dan R. Tanda panah atas (\uparrow) mengindikasikan bahwa sebuah

PGT diperlukan pada clock, label Q_0 mengindikasikan level pada Q sebelum PGT.



Gambar A.6 (a) SR FF yang hanya respons ketika sinyal PGT pada clock, (b) tabel kebenaran, (c) bentuk gelombang khas

Pada Gambar A.6(c) diilustrasikan operasi SR flip-flop. Jika kita menganggap bahwa semua kasus memerlukan persyaratan *setup time* dan *hold time*, maka analisis dapat diberikan sebagai berikut:

- Mula-mula semua masukan adalah 0 dan keluaran Q dianggap 0; jadi $Q_0 = 0$.

2. Bila PGT terjadi pada pulsa clock pertama (titik a), maka masukan S dan R keduanya 0, jadi FF tidak terpengaruh dan tetap dalam keadaan $Q = 0$ (yaitu $Q = Q_0$).
3. Pada kejadian PGT pulsa clock kedua (titik c), masukan S sekarang adalah high, dengan R tetap low. Jadi, FF set ke keadaan 1 pada tebing naik dari pulsa clock ini.
4. Ketika pulsa clock ketiga membuat transisi positif (titik e), akan diperoleh $S = 0$ dan $R = 1$, yang menyebabkan FF menjadi clear ke keadaan 0.
5. Pulsa yang keempat men-set FF sekali lagi ke keadaan $Q = 1$ (titik g) karena $S = 1$ dan $R = 0$ ketika terjadi tebing positif.
6. Pulsa kelima juga diperoleh $S = 1$ dan $R = 0$ ketika terjadi transisi menuju positif. Namun demikian, Q sudah high, jadi dia tetap dalam keadaan high.
7. Kondisi $S = R = 1$ tidak dapat digunakan karena hasilnya ambigu.

A.4.2 JK FLIP-FLOP

Gambar A.7(a) menunjukkan sebuah JK flip-flop yang dipicu oleh sinyal clock PGT. Masukan JK mengontrol keadaan FF sama seperti masukan S dan R kecuali ketika $J = K = 1$ maka kondisi ini tidak menyebabkan hasil output yang ambigu. Untuk kondisi ini, FF akan selalu menuju ke keadaan sebaliknya (komplementernya) ketika sinyal transisi positif tiba. Hal ini disebut operasi **toggle mode**. Pada mode ini, jika kedua J dan K tetap HIGH, maka FF akan berubah keadaan (toggle) untuk setiap clock PGT.

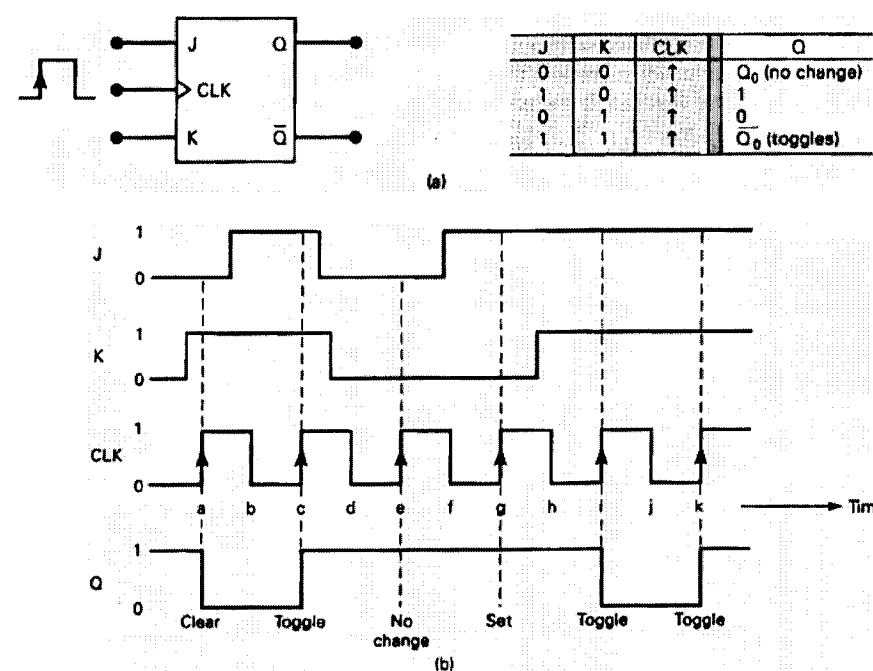
Tabel kebenaran pada Gambar A.7(a) merangkum bagaimana JK flip-flop merespons PGT untuk setiap kombinasi J dan K. Catatan bahwa tabel kebenaran sama seperti SR FF (Gambar A.6) kecuali untuk kondisi $J = K = 1$.

Kondisi ini menghasilkan $Q = \bar{Q}_0$.

Operasi FF ini diilustrasikan oleh bentuk gelombang pada Gambar A.7(b). Sekali lagi kita menganggap bahwa diperlukan persyaratan setup time dan hold time.

1. Mula-mula semua masukan adalah 0, dan keluaran Q dianggap 1; jadi $Q_0 = 1$.

2. Ketika terjadi pulsa clock tebing positif (PGT) (titik a), ada kondisi $J = 0, K = 1$. Hal ini menyebabkan FF akan menjadi clear ke keadaan $Q = 0$.
3. Pulsa clock yang kedua mendapat $J = K = 1$ ketika transisi positif tiba (titik c). Hal ini menyebabkan FF beralih ke toggle ke keadaan sebaliknya (komplementer), $Q = 1$.
4. Pada titik e pada bentuk gelombang clock, J dan K keduanya 0, sehingga FF tidak berubah keluarannya.
5. Pada titik g, $J = 1$ dan $K = 0$. Kondisi ini men-set Q ke keadaan 1. Namun dia sudah berada di posisi 1, dan berati akan tetap di situ.
6. Pada titik i, $J = K = 1$, sehingga FF toggle ke keadaan sebaliknya (komplementer). Hal yang sama juga terjadi pada titik k.

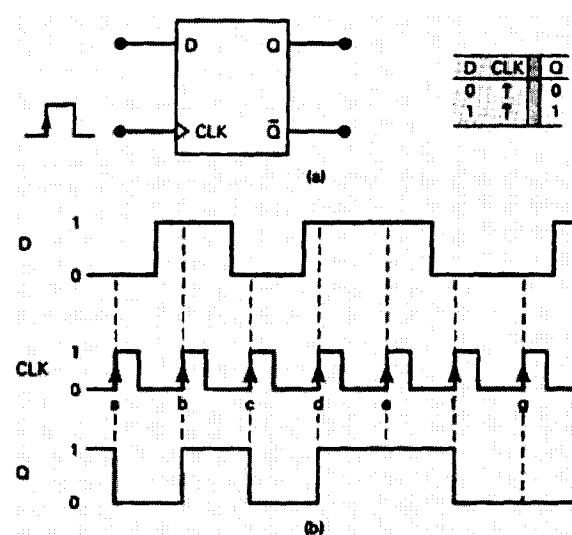


Gambar A.7 (a) JK FF yang hanya merespons pada saat PGT, (b) bentuk gelombang

A.4.3 D FLIP-FLOP

Gambar A.8(a) menunjukkan simbol dan tabel kebenaran untuk D flip-flop yang dipicu oleh clock PGT. Tidak seperti halnya SR FF dan JK FF, flip-flop ini hanya mempunyai satu masukan kontrol sinkron, D, yang merupakan singkatan dari **data**. Operasi FF D sangat sederhana, di mana Q akan menuju ke keadaan yang sama dengan masukan D ketika terjadi sinyal clock PGT. Dengan kata lain bahwa level yang ada pada D akan disimpan di dalam FF pada saat terjadinya PGT. Pada Gambar A.8(b) diberikan ilustrasi operasi ini.

Anggap bahwa mula-mula Q adalah HIGH. Ketika terjadi PGT yang pertama pada titik a, masukan D adalah LOW, jadi Q akan menuju ke keadaan 0. Walaupun level masukan D berubah di antara titik a dan b, dia tidak mempengaruhi Q; Q menyimpan LOW tersebut yang terjadi pada D di titik a. Ketika terjadi PGT pada titik b, maka Q menuju HIGH karena D adalah HIGH pada saat itu. Q menyimpan keadaan HIGH ini sampai PGT pada titik c menyebabkan Q menuju LOW, karena D adalah LOW pada saat itu. Dengan cara yang sama, keluaran Q akan mengambil level D yang ada ketika terjadi PGT pada titik d, e, f dan g. Catatan bahwa Q tetap HIGH pada titik e karena D tetap HIGH.



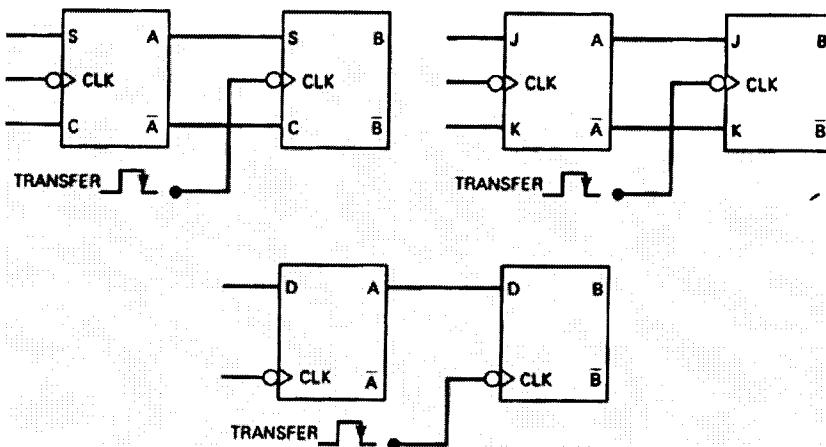
Gambar A.8 (a) D flip-flop yang dipicu oleh PGT, (b) bentuk gelombang

A.5 REGISTER

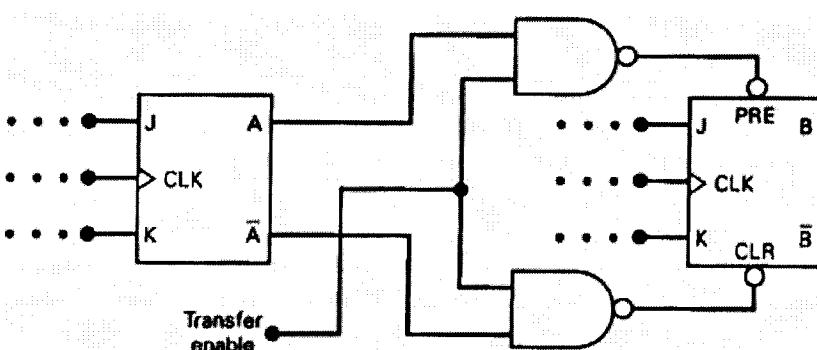
Flip-flop yang merupakan penyimpan satu bit dapat dicaskadekan untuk membentuk penyimpan yang kapasitasnya lebih besar. Untuk membangun sebuah register digunakanlah teknik cascade flip-flop yang menghasilkan penyimpanan yang jumlah bit nya ditentukan oleh perancangan.

Operasi yang paling sering dilakukan pada data adalah menyimpannya di dalam sebuah flip-flop atau register yang disebut operasi transfer data. Hal ini melibatkan transfer data dari satu FF atau register ke yang lain. Gambar A.9 mengilustrasikan bagaimana transfer data dapat dilakukan antara dua FF menggunakan SR FF, JK FF, dan D FF. Pada setiap kasus, nilai logika yang sekarang tersimpan dalam FF A ditransfer ke FF B menggunakan NGT dari pulsa TRANSFER. Jadi, setelah NGT ini, keluaran B akan sama dengan keluaran A.

Operasi transfer pada Gambar A.9 merupakan sebuah contoh **transfer sinkron**, karena kontrol sinkron dan masukan CLK digunakan untuk melakukan transfer. Sebuah operasi transfer dapat juga dilakukan dengan menggunakan masukan asinkron dan sebuah flip-flop. Gambar A.10 menunjukkan bagaimana sebuah **transfer asinkron** dapat dilakukan dengan menggunakan masukan PRESET dan CLEAR dari sebuah jenis flip-flop. Di sini masukan asinkron respons ke level LOW. Bila saluran TRANSFER ENABLE dalam keadaan LOW, maka dua keluaran NAND dijaga HIGH, tanpa mempengaruhi keluaran flip-flop. Bila saluran TRANSFER ENABLE dibuat HIGH, maka salah satu keluaran NAND akan LOW, bergantung pada keadaan keluaran A dan A'. Logika LOW ini akan men-set atau men-clear FF B untuk menyamakan keadaannya seperti FF A. Transfer asinkron ini dilakukan secara terpisah dari masukan sinkron dan masukan CLK.



Gambar A.9 Operasi transfer data sinkron dilakukan dengan bermacam jenis flip-flop

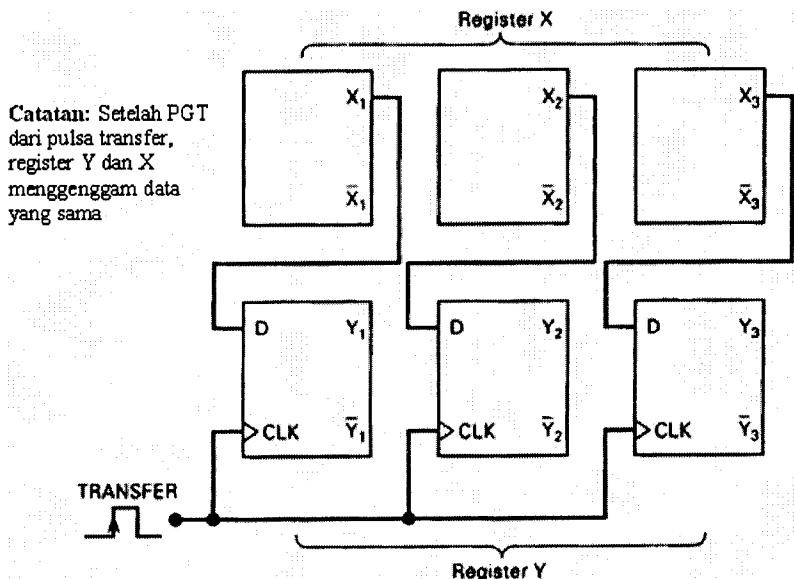


Gambar A.10 Operasi transfer data asinkron

A.5.1 Register Paralel (Transfer Data Paralel)

Gambar A.11 mengilustrasikan transfer data dari satu register ke register lain menggunakan flip-flop D. Register X terdiri atas FF X_1 , X_2 , dan X_3 ; register Y terdiri atas FF Y_1 , Y_2 , dan Y_3 . Pada aplikasi PGT dari pulsa TRANSFER, level yang tersimpan dalam X_1 ditransfer ke Y_1 , X_2 ditransfer ke Y_2 , dan X_3 ditransfer ke Y_3 . Transfer isi dari register X ke dalam register Y adalah operasi transfer sinkron dan juga dikenal dengan transfer paralel

karena isi X_1 , X_2 , dan X_3 ditransfer secara simultan/bersamaan ke dalam Y_1 , Y_2 , dan Y_3 . Jika dilakukan *serial transfer*, isi register X akan ditransfer ke register Y satu bit dalam satu waktu.

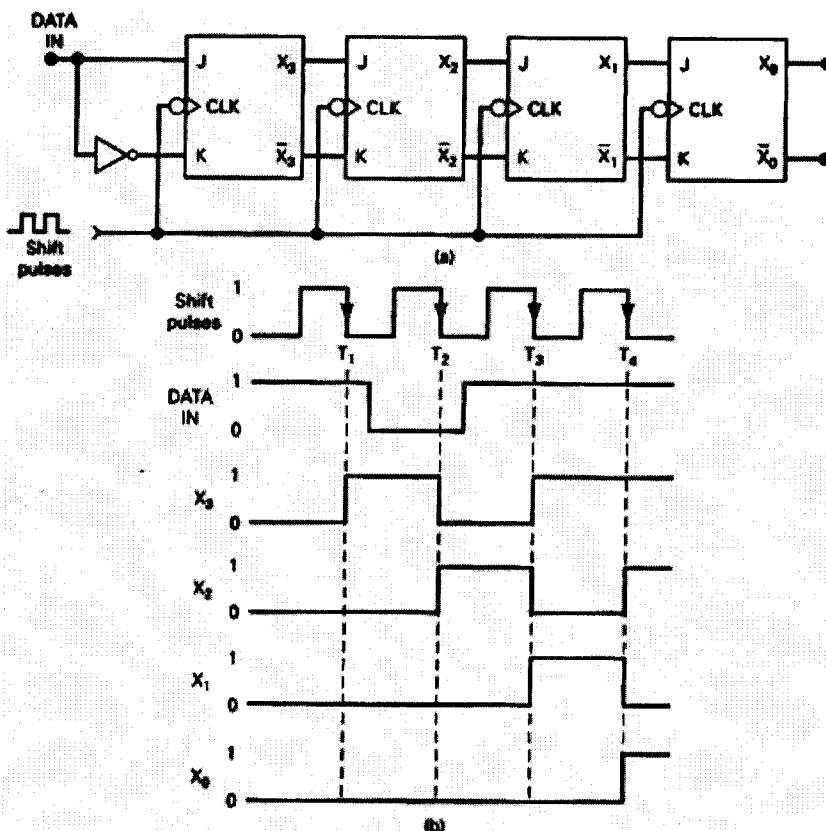


Gambar A.11 Transfer paralel isi register X ke register Y

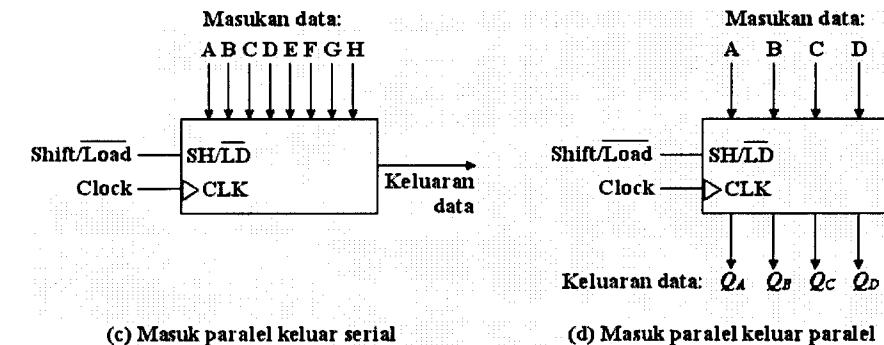
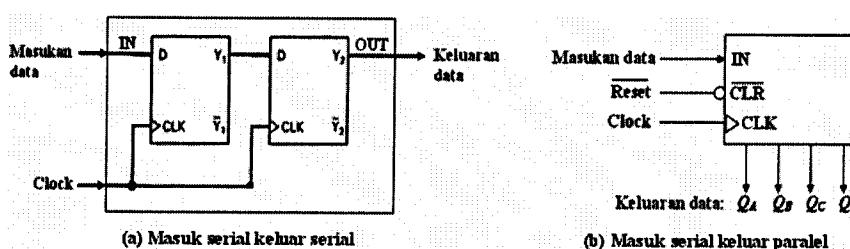
A.5.2 Register Geser (Transfer Data Serial)

Sebuah register geser dapat memindahkan bit-bit yang tersimpan ke kiri atau ke kanan. Register geser dibangun dari sejumlah FF sedemikian rupa sehingga dapat menyimpan bilangan biner dan menggesernya di dalam FF ke FF berikutnya dalam setiap pulsa clock.

Gambar A.12 menunjukkan satu macam JK flip-flop untuk membangun sebuah register geser yang mengoperasikan empat bit pergeseran. Flip-flop dihubungkan demikian agar keluaran X_3 ditransfer ke dalam X_2 ; X_2 ditransfer ke X_1 ; dan X_1 ditransfer ke X_0 .



Gambar A.12 Register geser empat-bit



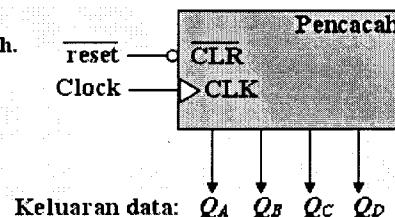
Gambar A.13 Jenis-jenis register

A.6 PENCACAH (COUNTER)

Pencacah adalah sirkuit yang mencacah jumlah pulsa yang diberikan melalui masukan clock. Pola keluaran pada flip-flop mengindikasikan jumlah pulsa clock yang diterima. Gambar A.14 menunjukkan sebuah pencacah biner 4 bit yang mencacah dari 0000 sampai 1111. Jenis pencacah dibedakan atas dua (1) pencacah asinkron atau disebut juga pencacah riak (*ripple counter*) dan (2) pencacah sinkron.

Pada pencacah asinkron (*ripple counter*), sinyal clock eksternal memicu flip-flop tingkat pertama (LSB) dan yang lainnya dipicu oleh keluaran tingkat yang sebelumnya. Pada pencacah sinkron, semua flip-flop dipicu oleh sinyal clock secara bersama.

Q_A, Q_B, Q_C, Q_D adalah bit-bit keluaran pencacah.
Keluaran *Q_A* adalah LSB dan *Q_D* adalah MSB.
Masukan reset ketika LOW maka pencacah menjadi clear.



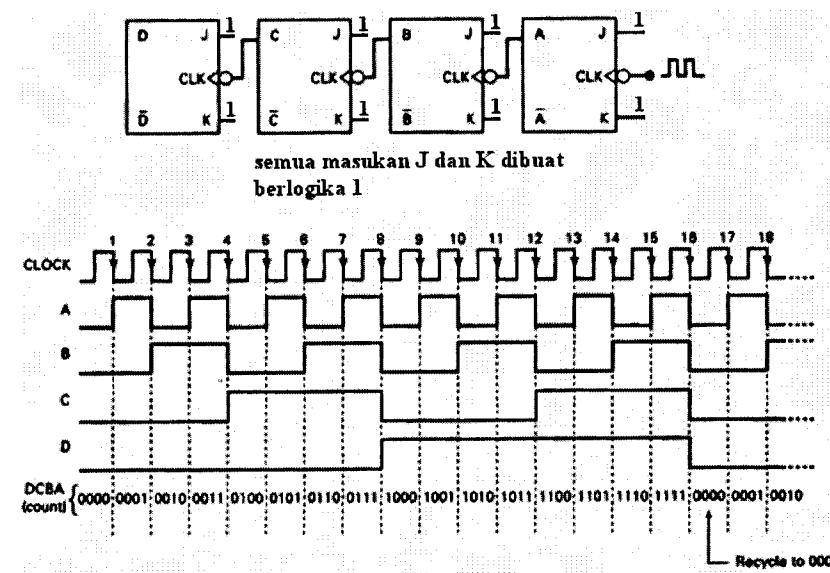
Gambar A.14 Pencacah biner

A.6.1 Pencacah Asinkron

Gambar A.15 menunjukkan sebuah sirkuit pencacah asinkron 4 bit. Operasi pencacah ini dijelaskan sebagai berikut:

1. Pulsa-pulsa clock hanya diberikan pada masukan CLK flip-flop A. Jadi flip-flop A akan toggle (berubah ke keadaan sebaliknya / komplemen) setiap saat pulsa clock memasuki transisi negatif (high ke low). Di sini semua masukan J dan K pada flip-flop dikondisikan dalam keadaan HIGH.
2. Keluaran normal flip-flop A bertindak sebagai masukan CLK untuk flip-flop B, dan juga flip-flop B akan toggle setiap saat keluaran A berubah dari 1 ke 0. Dengan cara yang sama, flip-flop C akan toggle bila keluaran B berubah dari 1 ke 0, dan flip-flop D akan toggle bila keluaran C berubah dari 1 ke 0.
3. Keluaran flip-flop D, C, B, dan A merepresentasikan sebuah bilangan biner empat bit dengan D sebagai MSB. Mari kita menganggap bahwa semua flip-flop telah dalam keadaan clear atau 0 (masukan CLEAR tidak diperlukan). Bentuk gelombang pada Gambar A.15 menunjukkan bahwa sebuah urutan cacah biner dari 0000 ke 1111 diberikan oleh pulsa-pulsa clock yang secara kontinu masuk ke flip-flop A.

4. Setelah NGT pulsa clock yang ke-15 terjadi, maka pencacah FF berada dalam kondisi 1111. Pada NGT yang ke-16, flip-flop A beralih dari 1 ke 0, yang menyebabkan flip-flop B beralih dari 1 ke 0, demikian seterusnya hingga pencacah dalam keadaan 0000. Dengan kata lain, pencacah telah melewati satu siklus lengkap (0000 hingga 1111) dan mempunyai siklus balik ke 0000, di mana pencacah akan mulai siklus pencacahan yang baru seperti pemberian pulsa-pulsa clock sebelumnya.



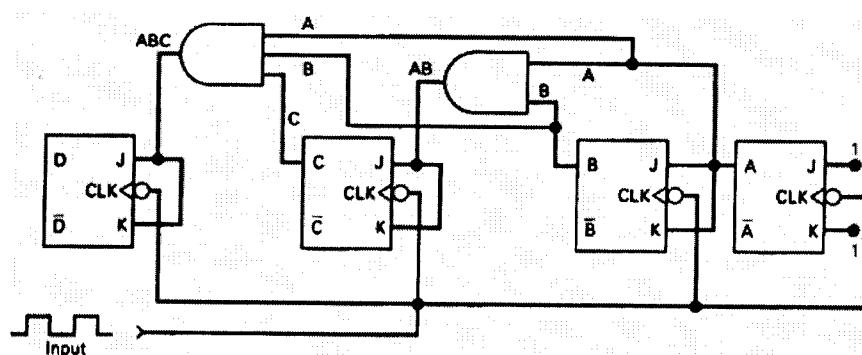
Gambar A.15 (a) sirkuit pencacah asinkron 4 bit, (b) bentuk gelombang

A.6.2 Pencacah Sinkron

Masalah yang dihadapi pada pencacah asinkron adalah akumulasi waktu tunda propagasi flip-flop, di mana setiap flip-flop berikutnya akan menunggu hingga keluaran flip-flop sebelumnya memberikan toggle. Jadi akan diperoleh waktu tunda yang paling lama pada flip-flop MSB untuk dapat memberikan keluarannya yang valid. Atau dengan kata lain, flip-flop tidak semuanya mengubah keadaannya secara simultan. Hal ini dapat diatasi dengan pencacah sinkron atau pencacah paralel di mana semua flip-

flop dipicu secara simultan (dengan paralel) oleh pulsa-pulsa masukan clock. Oleh karena itu pulsa-pulsa diberikan pada semua flip-flop. Pada Gambar A.16 ditunjukkan sebuah sirkuit pencacah sinkron 4 bit di mana perbedaan utama yang dapat dibandingkan dengan pencacah asinkron (Gambar A.15) adalah:

1. Masukan CLK semua flip-flop dihubungkan secara bersama sehingga sinyal clock masukan diberikan ke setiap flip-flop secara simultan.
2. Hanya flip-flop A (LSB) yang pada masukan J dan K nya diberikan level HIGH. Masukan J dan K flip-flop lainnya dikemudian oleh sejumlah kombinasi keluaran flip-flop.
3. Pencacah sinkron memerlukan sirkuit yang lebih daripada pencacah asinkron.



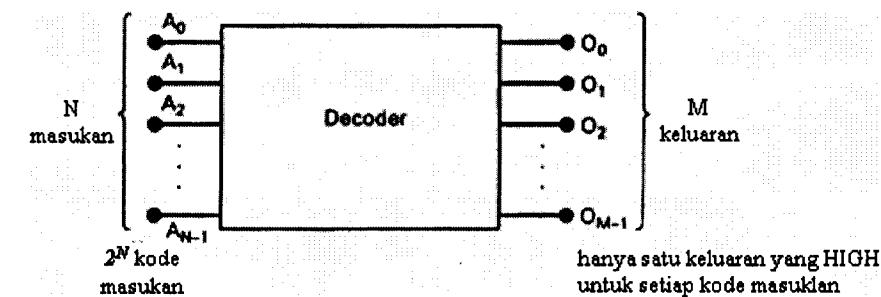
Gambar A.16 Sirkut pencacah sinkron

A.7 DECODER

Decoder adalah sirkuit logika yang menerima sekumpulan masukan yang merepresentasikan sebuah bilangan biner dan hanya mengaktifkan keluaran yang bersesuaian dengan bilangan masukan. Dengan kata lain, sebuah sirkuit decoder melihat pada masukannya, menentukan bilangan biner yang mana yang ada di sana, dan mengaktifkan satu keluaran yang bersesuaian ke bilangan tersebut; keluaran yang lain semuanya tetap tidak aktif. Diagram untuk decoder secara umum ditunjukkan pada Gambar A.17 dengan N masukan dan M keluaran. Karena setiap N masukan dapat 0 atau 1, maka ada 2^N kombinasi masukan yang mungkin atau kode-kode. Untuk

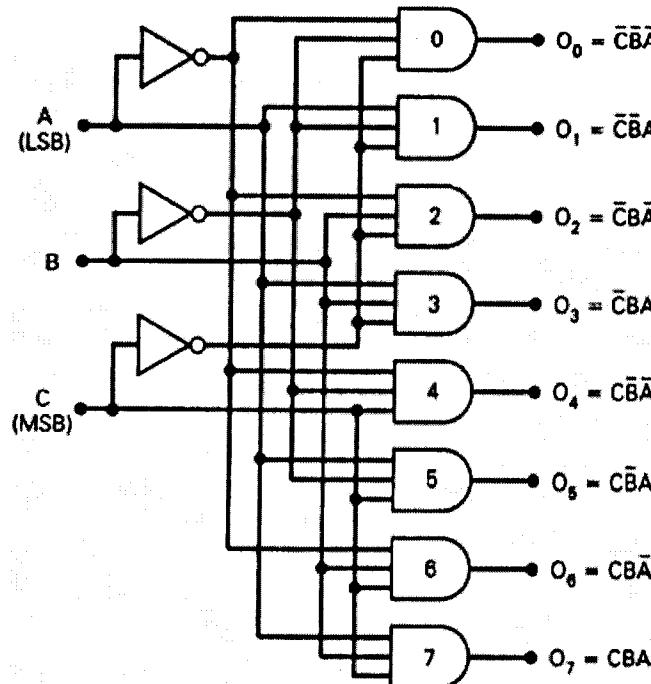
setiap kombinasi masukan ini hanya salah satu dari M keluaran yang akan aktif (HIGH); semua keluaran yang lain adalah LOW. Banyak decoder yang dirancang untuk menghasilkan keluaran aktif LOW, di mana hanya keluaran yang terpilih yang LOW sedangkan yang lainnya adalah HIGH. Hal ini akan diindikasikan oleh adanya lingkaran kecil pada saluran keluaran pada diagram decoder.

Beberapa decoder tidak menggunakan semua dari 2^N kode masukan yang mungkin tetapi hanya yang tertentu. Misalnya, sebuah decoder BCD-ke-desimal mempunyai kode masukan empat-bit dan sepuluh saluran yang bersesuaian ke sepuluh grup kode BCD 0000 sampai 1001. Jenis decoder ini sering dirancang sehingga jika ada kode yang tidak digunakan pada masukan yang ada, maka tidak ada keluaran yang akan diaktifkan.



Gambar A.17 Diagram decoder yang umum

Gambar A.18 menunjukkan sirkuit sebuah decoder dengan tiga masukan dan $2^3 = 8$ keluaran. Decoder ini semuanya menggunakan gerbang AND, dan juga keluarannya aktif-HIGH. Catatan bahwa untuk kode masukan yang diberikan, hanya keluaran yang aktif HIGH yang bersesuaian dengan ekivalen desimal dari kode masukan biner (misalnya, keluaran O_6 menjadi HIGH hanya jika $CBA = 110_2 = 6_{10}$).



C	B	A	O ₇	O ₆	O ₅	O ₄	O ₃	O ₂	O ₁	O ₀
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Gambar A.18 Decoder 3-ke-8

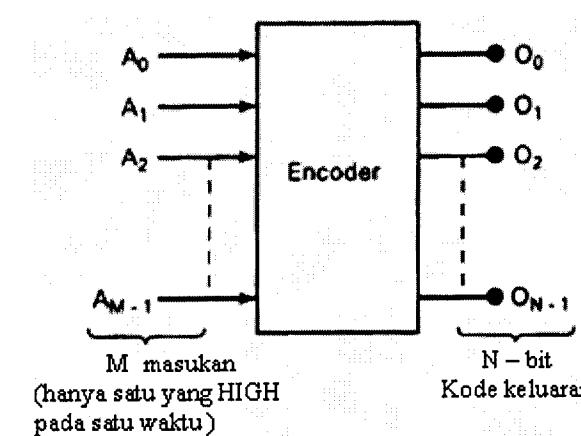
Decoder ini dapat dijelaskan dalam beberapa cara. Decoder ini dapat disebut decoder 3-baris-ke-8-baris, karena mempunyai tiga saluran masuk dan delapan saluran keluar. Juga dapat disebut decoder biner-ke-oktal atau converter karena dia mengambil kode masukan biner tiga-bit dan mengaktifkan salah satu dari delapan (oktal) keluaran yang bersesuaian

dengan kode tersebut. Juga dapat disebut sebagai decoder 1-8 karena hanya 1 dari 8 keluaran yang aktif pada satu waktu.

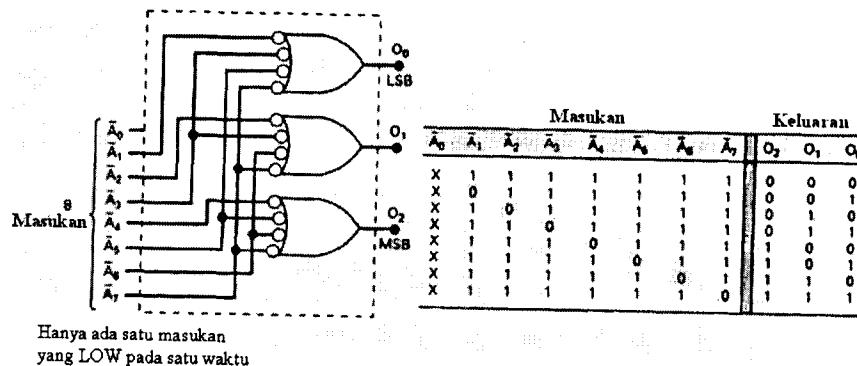
A.8 ENCODER

Sebuah encoder mempunyai banyak saluran masuk, hanya salah satu yang diaktifkan dalam satu waktu, dan menghasilkan kode keluaran N -bit, bergantung pada masukan mana yang diaktifkan. Gambar A.19 adalah sebuah diagram umum untuk sebuah encoder dengan M masukan dan N keluaran. Di sini masukan adalah aktif-HIGH, yang berarti bahwa mereka secara normal adalah LOW.

Kita melihat bahwa sebuah decoder biner-ke-oktal (3-baris-ke-8-baris) menerima kode masukan tiga-bit dan mengaktifkan salah satu dari delapan saluran keluaran yang bersesuaian dengan kode tersebut. Sebuah encoder oktal-ke-biner (encoder 8-baris-ke-3-bit) melakukan fungsi yang sebaliknya: dia menerima delapan saluran masukan dan menghasilkan kode keluaran tiga-bit yang bersesuaian dengan masukan yang diaktifkan. Gambar A.20 menunjukkan sirkuit logika dan tabel kebenaran untuk sebuah encoder oktal-ke-biner dengan masukan aktif-LOW.



Gambar A.19 Diagram encoder yang umum

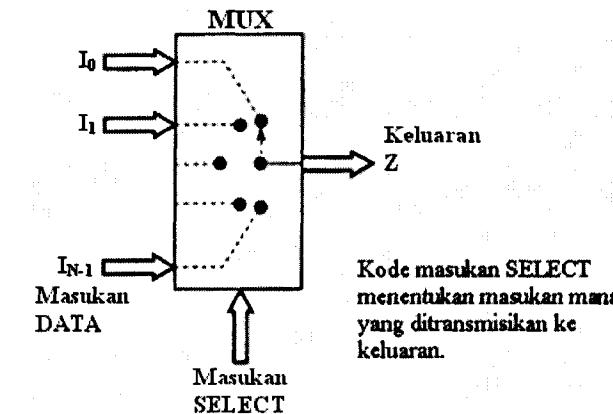


Gambar A.20 Sirkuit logika encoder oktal-ke-biner (8-baris-ke-3-baris)

A.9 MULTIPLEXER

Multiplexer adalah sebuah sirkuit logika yang melakukan pemilihan salah satu dari beberapa sinyal masukan dan melewatkannya pada keluaran. Multiplexer digital disebut juga **data selector** adalah sebuah sirkuit logika yang menerima beberapa masukan data digital dan memilih salah satu dari masukan tersebut pada satu waktu untuk dilewatkannya pada keluaran. Rute dari masukan data yang diinginkan ke saluran keluaran dikontrol oleh masukan **SELECT** (biasa disebut dengan masukan **ADDRESS**). Gambar A.21 menunjukkan diagram fungsional dari sebuah multiplexer digital yang umum. Masukan dan keluaran digambarkan sebagai anak panah yang lebih lebar daripada saluran; hal ini mengindikasikan bahwa mereka sebenarnya dapat lebih dari satu saluran sinyal.

Multiplexer berprilaku seperti sebuah saklar multi-posisi yang dikontrol secara digital di mana kode digital dikenakan pada masukan **SELECT** yang mengontrol masukan data mana yang akan di-switch ke keluaran. Misalnya, keluaran Z akan sama dengan masukan data I_0 untuk beberapa kode masukan khusus; Z akan sama dengan I_1 untuk kode masukan **SELECT** khusus lainnya; dan seterusnya. Dengan kata lain, sebuah multiplexer memilih 1 dari N masukan sumber data dan mentransmisikan data yang terpilih ke sebuah kanal keluaran tunggal. Hal ini disebut **multiplexing**.



Gambar A.21 Diagram fungsional multiplexer digital (MUX)

A.9.1 Multiplexer Dasar Dua-Masukan

Gambar A.22 menunjukkan sirkuit logika untuk multiplexer dua-masukan dengan masukan data I_0 , I_1 dan masukan **SELECT** S . Level logika dikenakan pada masukan S menentukan gerbang AND yang mana yang di-enable sehingga masukan datanya dilewatkannya melalui gerbang OR ke keluaran Z . Lihat dengan cara lain, yaitu dengan menggunakan ekspresi Boolean untuk keluaran Z :

$$Z = I_0 \bar{S} + I_1 S$$

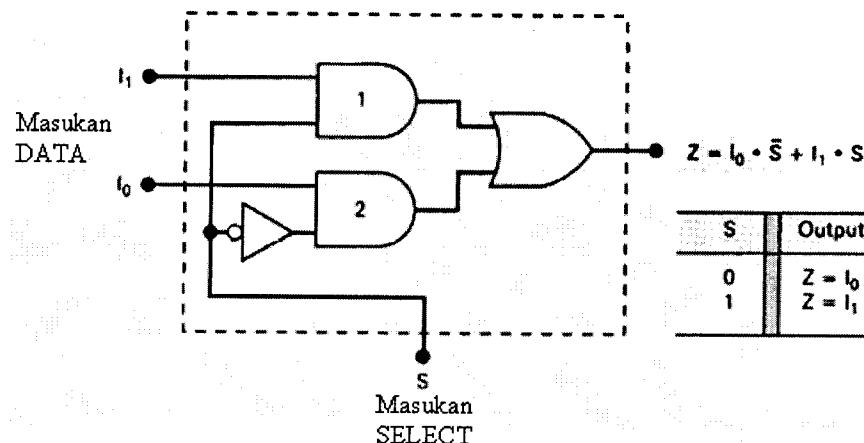
Dengan $S = 0$, ekspresi ini menjadi:

$$\begin{aligned} Z &= I_0 \cdot 1 + I_1 \cdot 0 \\ &= I_0 \end{aligned} \quad (\text{Gerbang 2 di-enable})$$

yang mengindikasikan bahwa Z akan identik dengan sinyal masukan I_0 , yang pada gilirannya dapat menjadi level logika tetap atau sinyal logika yang berubah terhadap waktu. Dengan $S = 1$, ekspresi menjadi:

$$Z = I_0 \cdot 0 + I_1 \cdot 1 = I_1 \quad (\text{Gerbang 1 di-enable})$$

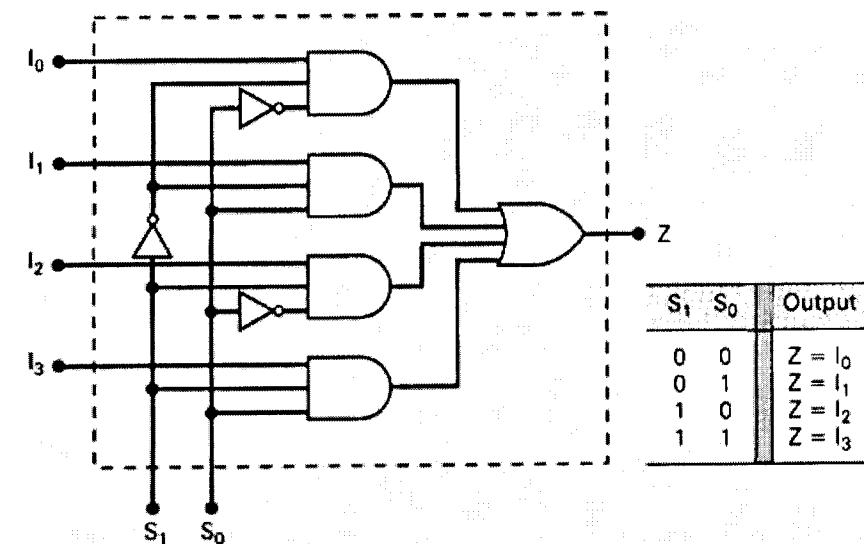
menunjukkan bahwa keluaran Z akan identik dengan sinyal masukan I_1 .



Gambar A.22 Multiplexer dua-masukan

A.9.2 Multiplexer Empat-Masukan

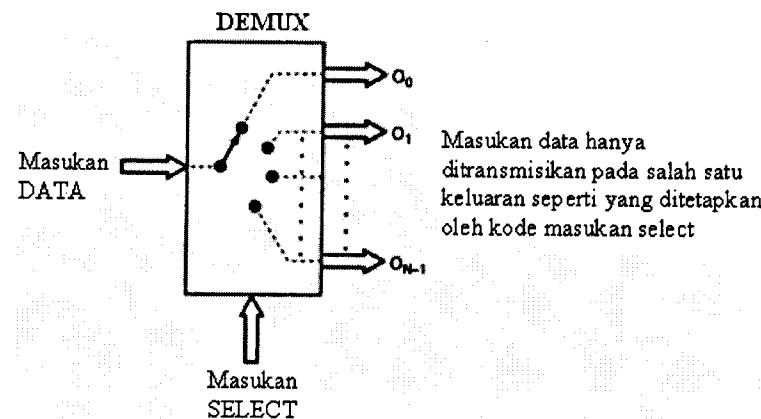
Gagasan yang sama dapat digunakan untuk membentuk multiplexer empat-masukan seperti yang ditunjukkan pada Gambar A.23. Di sini ada empat masukan, yang secara selektif mentransmisikannya ke keluaran sesuai dengan empat kombinasi masukan select S_1S_0 yang mungkin. Setiap masukan data digerbangkan dengan sebuah kombinasi level masukan select.



Gambar A.23 Multiplexer empat-masukan

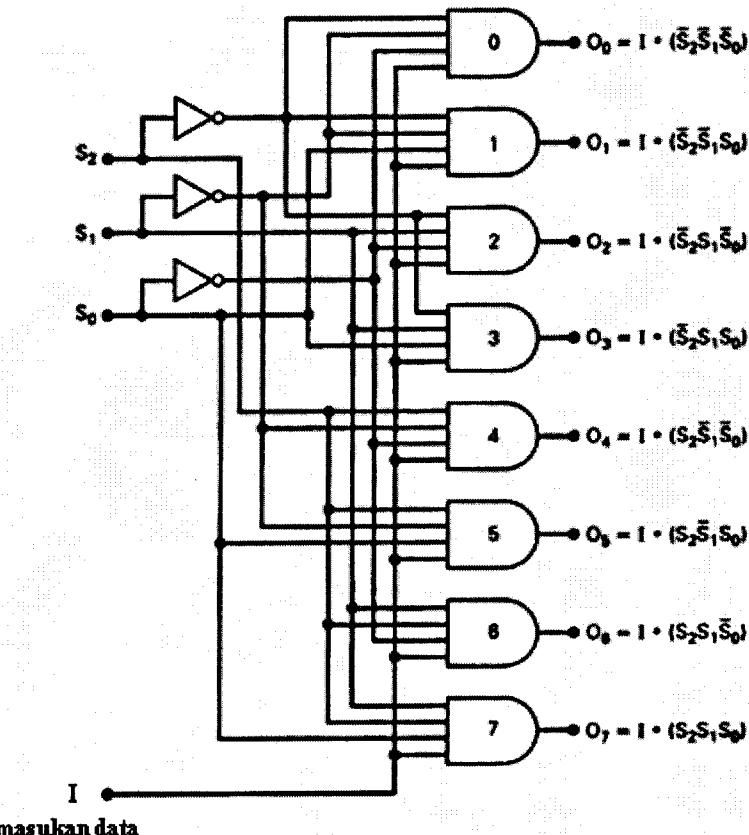
A.10 DEMULTIPLEXER

Sebuah multiplexer mengambil sejumlah masukan dan mentransmisikan salah satu nya ke keluaran. **Demultiplexer (DEMUX)** disebut juga **data distributor** melakukan operasi sebaliknya, dia mengambil sebuah masukan tunggal dan mendistribusikannya ke salah satu dari beberapa keluaran. Gambar A.24 menunjukkan diagram fungsional untuk sebuah demultiplexer digital. Tanda panah besar untuk masukan dan keluaran dapat merepresentasikan satu atau lebih saluran. Kode masukan select menentukan masukan DATA akan ditransmisi ke keluaran yang mana.



Gambar A.24 Diagram fungsional demultiplexer digital (DEMUX)

Gambar A.25 menunjukkan diagram logika sebuah demultiplexer yang mendistribusikan satu saluran masuk ke salah satu dari delapan saluran keluar. Saluran masukan data tunggal I dihubungkan ke semua delapan gerbang AND, tetapi hanya salah satu dari gerbang ini yang akan di-enable oleh saluran masuk SELECT.

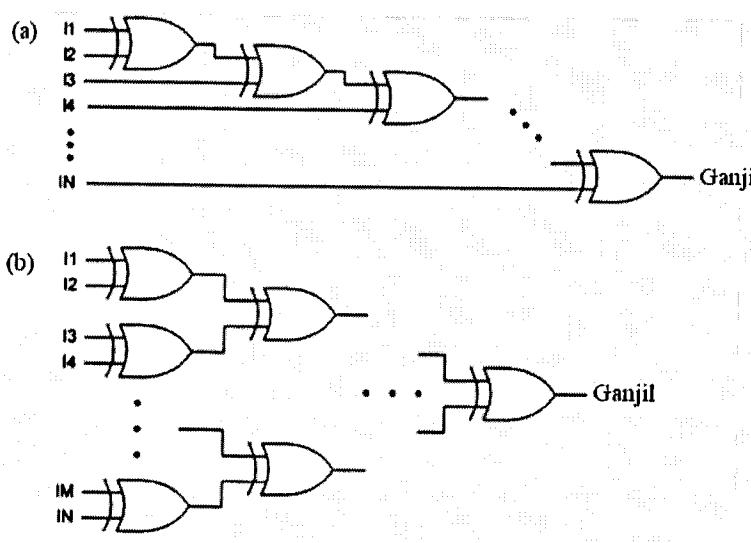


SELECT S_2 S_1 S_0	KELUARAN							
	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0 0 0	0	0	0	0	0	0	0	1
0 0 1	0	0	0	0	0	0	1	0
0 1 0	0	0	0	0	0	1	0	0
0 1 1	0	0	0	0	0	1	0	0
1 0 0	0	0	0	1	0	0	0	0
1 0 1	0	0	1	0	0	0	0	0
1 1 0	0	1	0	0	0	0	0	0
1 1 1	1	0	0	0	0	0	0	0

Gambar A.25 Demultiplexer 1-saluran-ke-8-saluran

A.11 PARITY GENERATOR/CHECKER

Gambar A.26(a) menunjukkan sejumlah gerbang n XOR dapat dikaskadekan untuk membentuk sebuah sirkuit dengan masukan $n + 1$ dan sebuah keluaran tunggal. Sirkuit ini disebut sirkuit parity-ganjil, karena keluarannya adalah 1 jika total jumlah masukan 1 berjumlah ganjil. Sirkuit pada Gambar A.26(b) juga merupakan sirkuit parity-ganjil, tetapi lebih cepat karena gerbangnya disusun dalam struktur-pohon. Jika keluaran sirkuit dibalik, maka kita peroleh sebuah sirkuit parity-genap yang mempunyai keluaran 1 jika masukan 1 berjumlah genap.

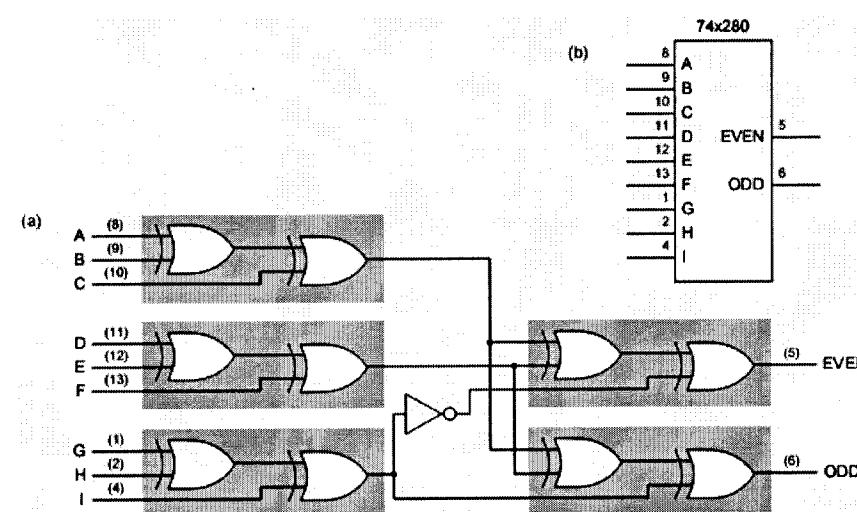


Gambar A.26 Caskade gerbang XOR: (a) hubungan daisy-chain dan (b) struktur-pohon

Pembangkit-parity (*parity-generator*) dalam bentuk IC dapat berupa kemasan MSI tunggal yang diberikan pada Gambar A.27. Sirkuit ini mempunyai sembilan masukan dan dua keluaran yang mengindikasikan apakah masukan 1 jumlahnya genap atau ganjil.

Aplikasi penguji-parity (*parity-checker*) adalah untuk mendeteksi error dalam transmisi dan penyimpanan data. IC 74x280 merupakan chip yang selain berfungsi sebagai pembangkit-parity untuk membangkitkan atau

menyimpan sebuah kode juga berfungsi untuk memeriksa/menguji bit parity ketika sebuah kode diambil kembali atau diterima.



Gambar A.27 Pembangkit-parity ganjil/genap 9 bit 74x280

A.12 CUSTOM CHIP

Sebuah *custom chip* berisi beberapa gerbang dengan interkoneksi khusus antara gerbang-gerbang. Keuntungan *custom chip* yaitu karena perancang sirkuit terbebas dari pembuatan interkoneksi dalam suatu sirkuit yang menggunakan beberapa gerbang. Hal ini meningkatkan sirkuit atau keandalan PCB. Jadi, sebuah *custom IC* menawarkan desain logika lengkap dalam sebuah chip tunggal.

A.12.1 Semicustom Chip

Pada sebuah *semicustom chip*, laris gerbang disediakan tanpa interkoneksi sirkuit yang selesai, karena itu memberikan pilihan koneksi kepada perancang sirkuit. Ada dua jenis *semicustom chip* yaitu (1) *Programmable Logic Array* (PLA), dan (2) *Programmable Array Logic* (PAL).

Pada sebuah PLA, terdapat sekumpulan gerbang AND yang merupakan gerbang-gerbang masukan. Masukan-masukan pada setiap gerbang AND dapat dipilih oleh pemakai dari pin-pin masukan bersama. Jadi, di dalam chip koneksi dari masukan ke gerbang AND tidak lengkap. Dan juga terdapat sekumpulan gerbang OR di mana masukan ke gerbang OR tidak lengkap. Pemakai mempunyai pilihan menghubungkan keluaran khusus gerbang AND ke masukan gerbang OR yang dikehendaki. Jadi, dalam sebuah PLA, koneksi masukan ke gerbang AND dan koneksi keluaran dari gerbang AND ke gerbang OR dibuat berdasarkan ketentuan perancang sirkuit. Keluaran PLA adalah keluaran gerbang OR. Jadi, sebuah chip PLA yang diberikan dapat digunakan pada berbagai sirkuit-sirkuit berbeda dalam mode interkoneksi yang berbeda. Ada dua jenis PLA yaitu: PLA dan FPLA (*Field Programmable Logic Array*).

Pada FPLA, pabrik IC menyediakan sebuah koneksi sekring (fuse) pada setiap interkoneksi yang mungkin baik untuk masukan gerbang AND maupun keluaran gerbang AND ke masukan gerbang OR. Jadi, FPLA memberikan semua kemungkinan interkoneksi. Perancang sirkuit membuka (menghilangkan) koneksi dengan memutus sekring.

Pada PLA, interkoneksi secara permanen dibuat oleh pabrik IC, sesuai instruksi pelanggan (perancang sirkuit). Perancang sirkuit memberikan perusahaan IC detail koneksi yang akan dibuat untuk sebuah desain khusus. Pabrik IC menyiapkan sebuah mask sesuai pesanan pelanggan. Ada juga PLA yang berisi gerbang-gerbang (AND dan OR) serta flip-flop.

Sebuah PLA atau FPLA dapat digunakan untuk mengganti sebuah sirkuit dengan beberapa gerbang logika atau sebagai sebuah IC pengubah kode mengantikan sebuah ROM *look-up table*.

Seperti halnya *custom IC*, *semicustom IC* juga menyediakan desain keamanan (atau rahasia) karena fungsi yang dilakukan oleh sebuah *semicustom IC* tidak dapat diketahui dengan mencari diagram sirkuit. Semicustom IC dapat berisi beberapa gerbang logika lengkap, flip-flop, register-register dan fungsi-fungsi yang serupa. Tabel A.2 membandingkan *custom* dan *semicustom IC*. IC 82S100 adalah sebuah FPLA $16 \times 48 \times 8$. Nomor tipe FPLA $16 \times 48 \times 8$ menunjukkan hal berikut:

- Terdapat 16 variabel masukan
- Terdapat 48 gerbang AND (gerbang-gerbang masukan)
- Terdapat 8 gerbang OR (gerbang-gerbang keluaran)

TABEL A.2 Perbandingan custom IC dan semicustom IC

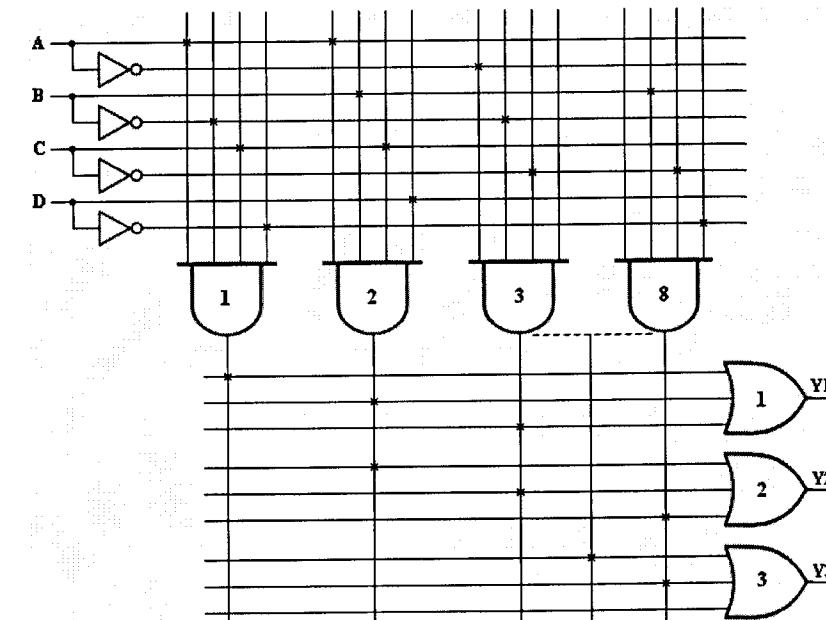
Fitur	Custom IC	Semicustom IC	Programmable Semicustom IC
Biaya pengembangan IC	tinggi	sedang	rendah
Waktu pengembangan IC	lama	sedang	singkat
Testability	cukup kuat	cukup kuat	sedang

Gambar A.28 mengilustrasikan sebuah PLA dengan 4 masukan dan 3 keluaran. PLA ini mempunyai 8 gerbang AND dan 3 gerbang OR. Setiap gerbang AND mempunyai 4 masukan, sedangkan setiap gerbang OR mempunyai 3 masukan. Koneksi mask programmable ditunjukkan oleh sebuah simbol X.

$$Y_1 = (AB'CD') + (ABCD) + (A'B'C')$$

$$Y_2 = (ABCD) + (A'B'C') + (BC'D')$$

$$Y_3 = \dots + (BC'D')$$



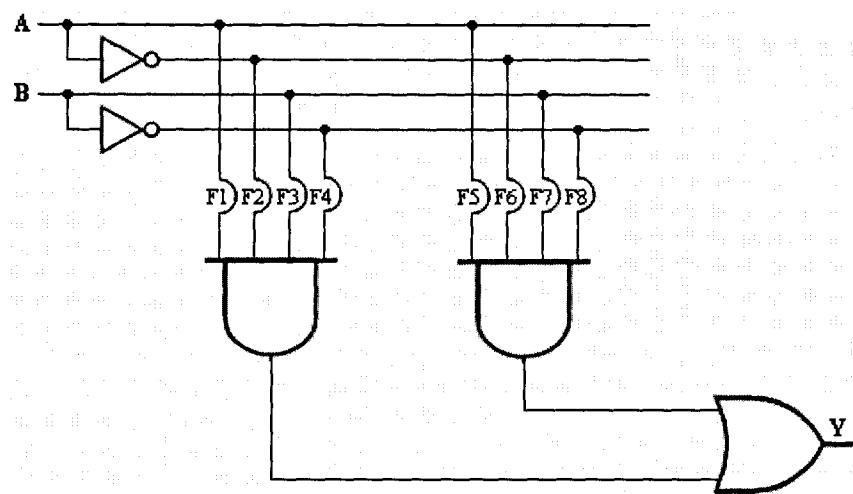
Gambar A.28 Sirkuit PLA dengan 4 masukan dan 3 keluaran

PAL

Sistem PAL mirip dengan PLA kecuali pada PAL variasinya sedikit dan hubungan keluaran gerbang AND ke masukan gerbang OR bersifat permanen. Jadi pada PAL, hanya koneksi/hubungan ke masukan gerbang AND yang dipilih oleh perancang.

Karena kesederhanaannya, maka PAL dipabrikasi/dibuat dengan lebih mudah. Menggunakan PAL pada sirkuit lebih sederhana daripada PLA.

Gambar A.29 mengilustrasikan sebuah PAL dengan 2 masukan, 1 keluaran dan 8 fuse. Untuk mengimplementasikan $Y = (AB' + A'B)$, maka fuse yang harus diputus/dibuka adalah F2, F3, F5 dan F8. Untuk mengimplementasikan $Y = (AB + A'B')$, maka fuse yang harus diputus/dibuka adalah F2, F4, F5, dan F7.



Gambar A.29 Sirkuit PAL dengan 2 masukan, 1 keluaran dan 8 fuse

A.12.2 Field Programmable Gate Array (FPGA)

FPGA mirip dengan PLA atau FPLA. FPGA mempunyai sekumpulan gerbang AND (gerbang-gerbang masukan) seperti pada PLA tetapi FPGA tidak mempunyai gerbang OR (gerbang-gerbang keluaran). FPGA berisi beberapa gerbang, register dan fungsi-fungsi logika sederhana lainnya yang

semuanya dihubungkan dengan metalisasi. Perancang sirkuit (pengguna IC) harus menghilangkan/memutus beberapa metal/logam setiap fungsi yang diinginkan dari FPGA. Hal ini dilakukan dengan memberikan arus listrik melalui pin-pin tertentu menggunakan peralatan khusus.

LAMPIRAN B
JAWABAN SOAL-SOAL ULANGAN

1. Jawaban Soal-soal Ulangan Bab 1

1. (1) disimpan; (2) diprogram
2. digital
3. biner
4. transistor
5. (1) logika; (2) ALU
6. mesin
7. bahasa tingkat tinggi
8. compiler
9. hardware
10. software
11. program aplikasi user
12. BIOS (*basic input output system*)
13. program aplikasi
14. program sistem
15. arsitektur komputer
16. organisasi komputer
17. unit input/output
18. unit kontrol
19. unit input
20. (1) program, (2) data
21. unit kontrol
22. ALU
23. unit output
24. *stored program concept*
25. (2) memori sekunder/memori pembantu (*auxiliary memory*)
26. pengontrol divais
27. (3) terintegrasi dengan CPU.
28. (3) sinyal data
29. sinyal kontrol
30. sinyal status
31. *word length*

32. jumlah lokasi
 33. waktu akses
 34. memori akses acak (memori utama)
 35. memori maknetik
 36. memori RAM
 37. (1) MAR (*memory address register*), dan (2) MBR (*memory buffer register*)
 38. MAR
 39. MBR
 40. (1) antarmuka sinkron, dan (2) antarmuka asinkron
 41. (2) fase eksekusi
 42. (2) operand
 43. *program counter* (PC)
 44. register instruksi
 45. MAR
 46. dekoder instruksi
 47. operasi makro
 48. operasi mikro
 49. interupsi
 50. *Interrupt Service Routine* (ISR, rutin pelayanan interupsi)
 51. *Interrupt Service Routine* (ISR, rutin pelayanan interupsi)
 52. *interrupt nesting*
 53. *interrupt priority* (prioritas interupsi)
 54. *interrupt masking*
 55. *non-maskable interrupt*
 56. I/O driver
 57. (2) *interrupt mode*; dan (3) DMA (*direct memory access*) mode
 58. *programmed mode (polling)*
 59. DMA (*direct memory access*) mode
 60. "Direct I/O" atau "I/O mapped I/O"
 61. *memory mapped I/O*
 62. hubungan bus/jalur
 63. (1) bus internal (menghubungkan (*link*) komponen-komponen internal dalam mikroprosesor) dan (2) bus eksternal (menghubungkan mikroprosesor dengan unit-unit eksternal seperti memori, pengontrol I/O, dan serpih (chip) pendukung lainnya)
 64. siklus bus (*bus cycle*)

65. (3) *I/O read bus cycle*: CPU membaca (menerima) data dari port input; dan (4) *I/O write bus cycle*: CPU menulis (mengirim) data ke port output.

66. jumlah waktu
 67. Program Benchmark

2. Jawaban Soal-soal Ulangan Bab 2

1. (4) Fitur-fitur dasar yang diinginkan user dan aspek-aspek teknik
2. (1) struktur fungsional, dan (2) struktur fisik
3. struktur fungsional
4. struktur fisik
5. (1) kinerja, (2) kapasitas,
6. (2) arsitektur CPU berbasis register, dan (3) arsitektur CPU berbasis stack
7. (2) *microprogrammed control unit*, dan (3) *hybrid control unit*
8. (2) *interrupt*, dan (3) *Direct memory access (DMA)*
9. (3) tumpangtindih, dan (4) realokasi
10. (7) *embedded system*
11. (2) teknologi transistor, dan (3) teknologi IC (rangkaian terintegrasi)
12. teknologi tabung hampa (*vacuum tube*)
13. teknologi transistor
14. teknologi Sirkut terpadu (SSI dan MSI)
15. teknologi LSI
16. teknologi VLSI

3. Jawaban Soal-soal Ulangan Bab 3

1. (1) *complex instruction set computing* (CISC), dan (2) *reduce instruction set computing* (RISC)
2. biayanya tinggi (mahal)
3. (1) mengurangi harga sistem (penggunaan memori kecil), dan (2) mengurangi waktu eksekusi program
4. *simple instruction* (instruksi sederhana)
5. unit kontrol
6. (6) satu clok per instruksi
7. (5) kinerja sistem
8. (1) CPU berbasis akumulator, (2) CPU berbasis register, dan (3) CPU berbasis stack
9. mesin satu alamat

10. mesin nol alamat
11. (1) kode operasi, dan (2) operand
12. (1) memori utama, (2) register CPU, (3) I/O port, dan (4) pada instruksi itu sendiri.
13. (1) *little endian*, dan (2) *big endian*
14. *little endian*
15. *big endian*
16. mode pengalamatan immediate
17. mode pengalamatan langsung (absolut)
18. (1) pengalamatan tak-langsung memori, dan (2) pengalamatan tak-langsung register.

4. Jawaban Soal-soal Ulangan Bab 4

1. (1) aritmetika biner, dan (2) aritmetika desimal
2. (1) aritmetika biner titik-tetap (*fixed-point*), dan (2) aritmetika biner titik-mengambang (*floating-point*)
3. bilangan BCD (*binary coded decimal*)
4. aritmetika biner titik-mengambang
5. aritmetika biner titik-mengambang
6. register
7. multiplexer
8. decoder
9. ASCII (*American Standard Code for Information Interchange*)
10. EBCDIC (*Extended Binary Coded Decimal Interchange Code*)
11. (1) representasi *signed magnitude*, (2) representasi komplemen-1, dan (3) representasi komplemen-2
12. representasi komplemen-2
13. (1) mantissa, (2) basis, dan (3) eksponen
14. (1) presisi tunggal, dan (2) presisi ganda
15. (1) *half adder*, dan (2) *full adder*
16. (1) *ripple carry adder*, dan (2) *carry look-ahead adder*
17. *carry look-ahead adder*
18. $n \times t_d$ (n dikalikan t_d)
19. *carry look-ahead adder*
20. bilangan biner bertanda

5. Jawaban Soal-soal Ulangan Bab 5

1. (3) Level transfer register, dan (4) level gerbang (*gate level*)
2. level arsitektur
3. opcode
4. level gerbang
5. (2) unit kontrol
6. bahasa transfer register (*Register transfer language – RTL*)
7. (3) mikro-operasi logika, dan (4) mikro-operasi pergeseran.
8. mikro-operasi transfer register
9. mikro-operasi logika
10. adder

6. Jawaban Soal-soal Ulangan Bab 6

1. unit kontrol
2. (1) *memory read*, dan (2) *memory write*
3. (1) pengambilan instruksi (*instruction fetch*), dan (2) eksekusi instruksi
4. (5) *execute operation* (EO), dan (6) *store result* (SR)
5. (3) *execute operation* (EO),
6. (3) *operand fetch* (OF)
7. (3) *operand address calculation* (OAC), (4) *operand fetch*
8. panjang instruksi; panjang word
9. datapath
10. opcode
11. (1) *hardwired control unit*, dan (2) *microprogrammed control unit*
12. *microprogrammed control unit*
13. *hardwired control unit*
14. (1) Horizontal micropogramming, an (2) Vertical micropogramming
15. (1) Desainnya tidak kompleks, dan (2) mikroprogram mudah dimodifikasi (fleksibel)
16. Lambat
17. Firmware
18. (1) Mikroprogram horisontal, dan (2) mikroprogram vertikal

7. Jawaban Soal-soal Ulangan Bab 7

1. (4) bandwidth
2. panjang word

3. waktu akses dan waktu pemulihan (*recovery time*)
4. waktu akses
5. waktu pemulihan (*recovery time*)
6. bandwidth
7. latency
8. lima bit
9. 2^N word
10. sinyal pemilih chip (*chip select*) atau sinyal enable
11. (4) metode akses
12. akses acak
13. memori utama
14. memori cache
15. memori virtual
16. *charge coupled device*
17. pabrik
18. PROM (*programmable read only memory*)
19. EPROM (*erasable programmable read only memory*), dan EEPROM (*erasable electrically read only memory*)
20. EEPROM (*erasable programmable read only memory*),
21. EEPROM (*electrically erasable read only memory*)
22. memori flash
23. memori RWM (*read/write memory*) yang lebih populer dengan istilah RAM
24. (1) memori statik (SRAM, *static random access memory*), dan (2) memori dinamik (DRAM, *dynamic random access memory*)
25. memori SRAM
26. memori SRAM
27. memori DRAM
28. memori DRAM
29. memori SRAM
30. memori DRAM
31. enam bit
32. 26 desimal
33. (3) BiCMOS ataupun CMOS
34. kecepatan lebih tinggi
35. (1) kapasitas lebih besar, dan (2) konsumsi daya lebih rendah

8. Jawaban Soal-soal Ulangan Bab 8

1. memori semikonduktor
2. (4) register
3. keandalan (*reliability*)
4. level L1
5. cache on-chip (cache L1)
6. (2) kapasitas
7. (2) waktu akses memori utama
8. (4) memori cache
9. memori virtual
10. prefetch instruksi
11. Interleaving memori
12. 40 MB/sec
13. n -kali
14. buffer tulis
15. memori cache
16. (2) *spatial locality*
17. *temporal locality*
18. *spatial locality*
19. (3) pemetaan asosiatif set (*set associative mapping*)
20. *cache hit*
21. *cache miss*
22. *hit rate atau hit ratio*
23. pemetaan langsung (*direct mapping*)
24. pemetaan asosiatif penuh (*fully associative mapping*)
25. penggantian cache (*cache replacement*)
26. *miss penalty*
27. *least frequently used*
28. *first-in first-out (FIFO)*
29. (2) *write-back policy*
30. *write-through policy*.
31. (2) *split cache*
32. (1) instruksi, dan (2) data
33. (2) alamat fisik
34. *offset*
35. *segmentation*

9. Jawaban Soal-soal Ulangan Bab 9

1. (2) *I/O driver*
2. pengontrol I/O
3. *I/O driver*
4. antarmuka
5. antarmuka paralel
6. bit demi bit (satu per satu secara serial).
7. pemilihan antarmuka
8. status/kondisi internal
9. *Custom built IC* atau *gate array*.
10. port I/O
11. *memory mapped I/O*
12. *direct I/O* atau *I/O mapped I/O*
13. (2) asinkron
14. sinyal clock yang sama
15. pengakuan (*acknowledgement*)
16. interupsi
17. rutin layanan interupsi (*interrupt service routine, ISR*)
18. (1) interupsi internal, dan (2) interupsi eksternal
19. bercabang ke ISR
20. *interrupt enable (IE)*
21. interupsi tersarang (*interrupt nesting*)
22. (2) transfer dari memori ke divais output
23. (2) melalui CPU (metode softdware)
24. pengontrol DMA (*DMA controller*)
25. *interrupt mode*
26. arbitrasi bus
27. (1) arbitrasi bus terpusat, dan (2) arbitrasi bus terdistribusi
28. (3) metode *fixed priority* atau *independent request*
29. bus loteng tengah (*mezzanine bus*)
30. bus I/O

10. Jawaban Soal-soal Ulangan Bab 10

1. (3) mengirimkan kode ke komputer
2. *scanning*
3. (3) *capacitive keyswitch*
4. ECMA-23 (edisi kedua)
5. *American Standard Code for Information Interchange*

6. tampilan CRT (*cathode ray tube*)
7. tampilan LCD (*liquid crystal display*)
8. (2) *non-impact printer*
9. *impact printer*
10. (2) *printer daisy wheel*
11. *non-impact printer*
12. *head*
13. *removable hard disk drive*
14. (2) *movable head*
15. *track*
16. *track*
17. *silinder*
18. *sektor*
19. *intertrack gap* (atau gap saja)
20. semakin kecil
21. lebih lambat
22. (2) *zone bit recording*
23. (4) *transfer time*
24. *seek time*
25. *rotational delay* atau disebut juga *rotational latency*
26. *access time*
27. *transfer time*
28. RAID (*Redundant Array of Inexpensive Disks*) yang kemudian diganti dengan kepanjangan "*Redundant Array of Independent Disks*"
29. RAID level 0
30. RAID level 1
31. secara serial
32. cahaya laser
33. *pit* (cekungan).
34. MODEM (*modulator demodulator*)
35. *hand-held scanner*

11. Jawaban Soal-soal Ulangan Bab 11

1. (2) *Paralelisme*: Pengeksekusian lebih dari satu tugas secara paralel.
2. (2) unit-unit fungsional homogen
3. (2) paralelisme level prosesor
4. paralelisme level instruksi
5. paralelisme level prosesor

6. paralelisme level prosesor
7. paralelisme level instruksi
8. (3) *multiple instruction stream, single data stream* (MISD), (4) *multiple instruction stream, multiple data stream* (MIMD)
9. *pipeline*
10. *throughput*
11. (2) *pipeline* aritmetika
12. *pipeline* instruksi
13. *pipeline* aritmetika
14. Gain kinerja (*speed-up*) *pipeline*
15. (2) ketergantungan data disebut juga *data hazard*, (3) kesulitan pencabangan disebut juga *control hazard*
16. *Write-After-Write* (WAW) *hazard*
17. prosesor vektor

12. Jawaban Soal-soal Ulangan Bab 12

1. (2) *loosely coupled*
2. *loosely coupled*
3. *tightly coupled*
4. (3) *no remote memory access*
5. (7) jarigan hypercube
6. interkoneksi memori multiport
7. interkoneksi crossbar switch
8. *clustering*
9. *cache coherence*
10. *fault tolerance*

LAMPIRAN C

JAWABAN SOAL-SOAL LATIHAN

JAWABAN SOAL-SOAL LATIHAN BAB 1

1. Karena komputer *notebook* lebih mudah dibawa (*portability*) dibandingkan komputer *desktop*, maka dibutuhkan teknologi tinggi untuk membuat chip yang ukurannya kecil tetapi tetap dengan kinerja yang sama atau bahkan lebih baik.
2. (i) Yang membutuhkan perubahan arsitektur adalah jika memilih point (b) yaitu memperkenalkan sebuah memori cache kecil; (ii) Yang membutuhkan teknologi yang lebih baik adalah jika memilih point (a) mengganti memori utama dengan memori yang lebih cepat dan (c) meningkatkan frekuensi clock hingga 200 MHz.
3. Memori magnetik lebih superior dalam aspek "**harga per byte**" daripada memori semikonduktor.
4. Memori fisik maksimum yang dapat dibangun adalah: 2^{32} byte atau sama dengan 4GB (GB = giga byte)
5. Diketahui memori dengan *word length* = 4B (32-bit); dan saluran alamat CPU 24 bit.
Jadi kapasitas memori = *word length* x jumlah lokasi = $4B \times 2^{24} = 4B \times 16M = 64\text{ MB}$
6. Diketahui komputer dengan frekuensi sinyal clock 4.77MHz.
Jadi periode sinyal clock (T) = $1/f = 1/4.77\text{ MHz} = 1/(4.77 \times 10^6)$ detik $\approx 0.21\text{ }\mu\text{detik.}$
7. MIPS = frekuensi clock / (CPI $\times 10^6$)
Jadi CPI = frekuensi clock / (MIPS $\times 10^6$)
 $CPI = 100\text{ MHz} / (100 \times 10^6) = (100 \times 10^6) / (100 \times 10^6) = 1$
8. T = 100 ns = 0.1 $\mu\text{s.}$

Jumlah siklus clock per instruksi = waktu siklus instruksi / periode clock (T)

Kelompok Instruksi	Persentase kemunculan	Jumlah siklus clock per instruksi
ALU	30	1/0.1 = 10
Load & store	50	0.6/0.1 = 6
Branch	20	0.8/0.1 = 8

$$CPI_a = \frac{\sum_{i=1}^n CPI_i \times I_i}{jumlah\ instruksi} = \frac{30 \times 10 + 50 \times 6 + 20 \times 8}{100} = 7.6$$

9. jumlah siklus clock = waktu CPU x frekuensi clock

jadi:

$$\text{jumlah siklus clock CPU A} = 50 \text{ detik} \times 500 \text{ MHz} = 25 \times 10^9$$

$$\text{jumlah siklus clock CPU B} = 2.5 \times \text{jumlah siklus clock CPU A}$$

$$\text{jumlah siklus clock CPU B} = 2.5 \times 25 \times 10^9 = 62.5 \times 10^9$$

$$\text{frekuensi clock CPU B} = \text{jumlah siklus clock CPU B} / \text{waktu CPU B}$$

$$\text{frekuensi clock CPU B} = 62.5 \times 10^9 / 20 \text{ detik} = 3.125 \times 10^9 \text{ Hz} = 3125 \text{ MHz}$$

10. Waktu CPU = (jumlah instruksi x CPI) / f = (jumlah instruksi x CPI) x T ; JADI:

$$\text{Waktu CPU A} = \text{jumlah instruksi} \times 4.0 \times 50 \text{ ns} = 200 \times \text{jumlah instruksi} \times 1 \text{ ns}$$

$$\text{Waktu CPU B} = \text{jumlah instruksi} \times 2.5 \times 65 \text{ ns} = 162.5 \times \text{jumlah instruksi} \times 1 \text{ ns}$$

- Karena jumlah instruksi yang diproses untuk CPU A dan CPU B sama, berarti CPU B lebih cepat daripada CPU A.

11. Untuk code sequence 1:

$$CPI_{cs1} = (1 \times 2 + 3 \times 1 + 4 \times 2) / (2+1+2) = 2.6$$

Untuk code sequence 2:

$$CPI_{cs2} = (1 \times 4 + 3 \times 3 + 4 \times 1) / (4+3+1) = 2.125$$

- Karena $CPI_{cs2} < CPI_{cs1}$ maka yang paling cepat adalah code sequence 2 dengan nilai CPI = 2.125.

12. Untuk compiler 1:

$$CPI_{com1} = (2 \times 15 + 5 \times 5 + 7 \times 3) / (15+5+3) = 3.304$$

$$\begin{aligned} MIPS_{com1} &= \text{frekuensi clock} / (CPI_{com1} \times 10^6) = 500 \times 10^6 / (3.304 \times 10^6) \\ &= 151.331 \end{aligned}$$

$$\text{Waktu CPU}_{com1} = (\text{jumlah instruksi} \times CPI_{com1}) / \text{frekuensi clock}$$

$$\text{Waktu CPU}_{com1} = (23 \times 10^6 \times 3.304) / (500 \times 10^6) = 0.152 \text{ detik}$$

Untuk compiler 2:

$$CPI_{com2} = (2 \times 25 + 5 \times 2 + 7 \times 2) / (25+2+2) = 2.551$$

$$\begin{aligned} MIPS_{com2} &= \text{frekuensi clock} / (CPI_{com2} \times 10^6) = 500 \times 10^6 / (2.551 \times 10^6) \\ &= 196.001 \end{aligned}$$

$$\text{Waktu CPU}_{com2} = (\text{jumlah instruksi} \times CPI_{com2}) / \text{frekuensi clock}$$

$$\text{Waktu CPU}_{com2} = (29 \times 10^6 \times 2.551) / (500 \times 10^6) = 0.148 \text{ detik}$$

- Karena $MIPS_{com2} > MIPS_{com1}$ maka berdasarkan MIPS: code sequence 2 lebih cepat daripada code sequence 1;
- Karena $\text{Waktu CPU}_{com2} < \text{Waktu CPU}_{com1}$ maka berdasarkan waktu eksekusi: code sequence 2 lebih cepat daripada code sequence 1.

JAWABAN SOAL-SOAL LATIHAN BAB 2

1. Ringkasan fitur-fitur utama generasi komputer:

GENERASI	TEKNOLOGI	PENEMUAN BARU YG UTAMA
1	Tabung hampa	<i>Stored Program Concept</i> , memori magnetik sebagai memori utama, aritmetika biner <i>fixed point</i>
2	Transistor	Sistem operasi, <i>multiprogramming</i> , <i>compiler</i> , hard disk magnetik, aritmetika biner <i>floating point</i> , <i>minicomputer</i>
3	Sirkut terpadu (SSI dan MSI)	<i>Multiprocessing</i> , memori semikonduktor, memori virtual, memori cache, <i>supercomputer</i>

GENERASI	TEKNOLOGI	PENEMUAN BARU YG UTAMA
4	LSI	Konsep RISC, microcomputer, kontrol proses, workstation
5	VLSI	Networking, sistem server, multimedia, embedded system

2. Yang dimaksud *stored program concept* adalah sebuah konsep sistem komputer yang dikemukakan oleh John Von Neumann yang menyatakan *program bahasa mesin disimpan di dalam komputer serta data relevan lainnya, dan secara intrinsik komputer mampu memanipulasi program dan data tersebut, misalnya mengambil (load) data/program dari disk ke memori, memindahkannya dari satu lokasi memori ke lokasi memori lainnya, dan menyimpannya kembali ke disk*
3. Dapat digunakan ekspansi memori konvensional dengan menambahkan langsung chip memori kapasitas 64 KB.

JAWABAN SOAL-SOAL LATIHAN BAB 3

1. Tipe CPU berbasis akumulator lebih kecil dan murah daripada CPU berbasis register karena pada CPU berbasis akumulator penggunaan register lebih sedikit.
2. CPU berbasis register dapat dipandang sebagai CPU berbasis akumulator jamak (*multiple accumulator*) karena penggunaan register yang berfungsi sebagai akumulator jumlahnya lebih banyak.
3. Ketika *program counter* (PC) mulai start pada 0111, maka program akan bercabang ke alamat 1001. Ketika alamat 1001 dibaca dan dieksekusi maka program akan bercabang ke alamat 0111 kembali demikian seterusnya terjadi *looping abadi* sehingga alamat 1000 tidak akan pernah dibaca dan dieksekusi.
4. $X = (A * B) + (C * D)$

CPU berbasis akumulator	CPU berbasis register	CPU berbasis Stack
LD A	LD R1,A	PUSH A
MUL B	MUL R1,B	PUSH B
ST X	LD R2,C	MUL
LD C	MUL D	PUSH C
MUL D	ADD R1,R2	PUSH D
ADD X	ST R1,X	MUL
ST X		ADD
		POP X

5. Dikethau i panjang instruksi 12 bit, *operand field* 4 bit, jumlah instruksi nol-alamat = m , jumlah instruksi dua-alamat = n . *Operand tertinggi* adalah *operand* dua-alamat sehingga *opcode* nya = $12 - 4 - 4 = 4$ bit, format instruksi nya:

Opcode field	Operand1 field	Operand2 field
4 bit	4 bit	4 bit

- jumlah instruksi *operand* satu-alamat yang memungkinkan = $2^4 - m$
 $- n = 16 - m - n$

JAWABAN SOAL-SOAL LATIHAN BAB 4

1. ALU 8-bit melakukan operasi penjumlahan dua operand yaitu $5 + 9$:

$$\begin{array}{r}
 00000101 \\
 00001001 \\
 \hline
 00001110
 \end{array}$$
 - a) *carry flag* = 0 karena hasil operasi *unsigned arithmetic* tidak terdapat *carry out*
 - b) *zero flag* = 0 karena hasil operasinya tidak sama dengan nol
 - c) *overflow flag* = 0 karena hasil operasi *signed arithmetic* tidak keluar dari rentang bilangan bertanda (*signed number*)
 - d) *sign flag* = 0 karena bit MSB hasil operasi aritmetika adalah 0

2. Diketahui sebuah word = 1100001011110010.

- a) 1100001011110010 = -17138 dalam *sign magnitude integer*
- b) 1100001011110010 = +15629 dalam komplement-1
- c) 1100001011110010 = 49906 dalam integer tak bertanda
- d) 1100001011110010 = +15630 dalam komplement-2

3.

a) $+47 = 00101111$	b) $-47 = 11010001$
$\begin{array}{r} +25 = 00011001 \\ \hline +72 = 01001000 \end{array}$	$\begin{array}{r} -25 = 11100111 \\ \hline -72 = 10111000 \end{array}$
c) $+47 = 00101111$	d) $-47 = 11010001$
$\begin{array}{r} -25 = 11100111 \\ \hline +22 = 00010110 \end{array}$	$\begin{array}{r} +25 = 00011001 \\ \hline -22 = 11101010 \end{array}$

4.

a) $-58 = 11000110$

Dalam format saintifik yang sudah ternormalisasi = 1.1000110×2^7

➢ Jadi S = 1 (negatif), M = 1000110, E = 7; E' = E + 127 = 134 = 10000110

Format standar IEEE 754 = 1 10000110
10001100000000000000000000000000

b) Cara konversi 0.1823:

$$\begin{aligned} 0.1823 \times 2 &= 0.3646 \rightarrow 0 \\ 0.3646 \times 2 &= 0.7292 \rightarrow 0 \\ 0.7292 \times 2 &= 1.4584 \rightarrow 1 \\ 0.4584 \times 2 &= 0.9168 \rightarrow 0 \\ 0.9168 \times 2 &= 1.8336 \rightarrow 1 \\ 0.8336 \times 2 &= 1.6672 \rightarrow 1 \\ 0.6672 \times 2 &= 1.3344 \rightarrow 1 \\ 0.3344 \times 2 &= 0.6688 \rightarrow 0 \\ 0.6688 \times 2 &= 1.3376 \rightarrow 1 \\ 0.3376 \times 2 &= 0.6752 \rightarrow 0 \\ 0.6752 \times 2 &= 1.3504 \rightarrow 1 \\ 0.3504 \times 2 &= 0.7008 \rightarrow 0 \\ 0.7008 \times 2 &= 1.4016 \rightarrow 1 \\ 0.4016 \times 2 &= 0.8032 \rightarrow 0 \end{aligned}$$

$$0.8032 \times 2 = 1.6064 \rightarrow 1$$

$$0.6064 \times 2 = 1.2128 \rightarrow 1$$

$$0.2128 \times 2 = 0.4256 \rightarrow 0$$

$$0.4256 \times 2 = 0.8512 \rightarrow 0$$

$$0.8512 \times 2 = 1.7024 \rightarrow 1$$

$$0.7024 \times 2 = 1.4048 \rightarrow 1$$

$$0.4048 \times 2 = 0.8096 \rightarrow 0$$

$$0.8096 \times 2 = 1.6192 \rightarrow 1$$

Sehingga $3.1823 = 11.0010111010101011001101$

Dalam format saintifik ternormalisasi =

$$1.10010111010101011001101 \times 2^1$$

➢ Jadi S = 0 (positif), M = 1000110, E = 1; E' = E + 127 = 128 = 10000000

Format standar IEEE 754 = 0 10000000
10010111010101011001101

c) $0.6 = 0.100110011001100110011001$

Dalam format saintifik ternormalisasi =

$$1.00110011001100110011001 \times 2^{-1}$$

➢ Jadi S = 0 (positif), M = 001100110011001100110011001, E = -1; E' = E + 127 = 126 = 01111110

Format standar IEEE 754 = 0 01111110
00110011001100110011001

d) $38.24 = 100110.001111010111000010$

Dalam format saintifik ternormalisasi =

$$1.00110001111010111000010 \times 2^5$$

➢ Jadi S = 0 (positif), M = 00110001111010111000010, E = 5; E' = E + 127 = 132 = 100000100

Format standar IEEE 754 = 0 100000100
00110001111010111000010

5. a) $00111111111000000000000000000000 \Rightarrow S = 0$ (positif), E' =

$$01111111 (= 127),$$

$$M = 11100000000000000000000000000000$$

Tambahkan 1 hidden bit pada M sehingga M = 1.

$$11100000000000000000000000000000$$

$$E = E' - 127 = 127 - 127 = 0$$

Bilangan biner notasi saintifik lengkap adalah = 1.

$$11100000000000000000000_2 \times 2^0$$

Jadi ekivalen desimalnya = +1.875

b) $01000010100110000000000000000000 \Rightarrow S = 0$ (positif), $E' =$

$$10000101 (= 133),$$

$$M = 01100000000000000000000000$$

Tambahkan 1 **hidden bit** pada M sehingga M =

$$1.0011000000000000000000000000000$$

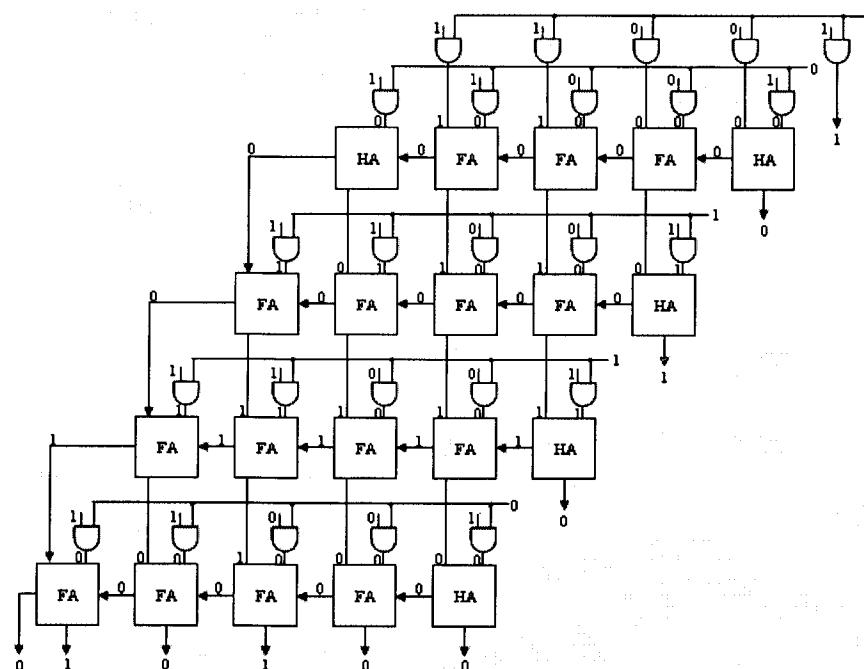
$$E = E' - 127 = 133 - 127 = 6$$

Bilangan biner notasi saintifik lengkap adalah =

$$1.0011000000000000000000000000000 \times 2^6$$

Jadi ekivalen desimalnya = +76.0

6.



7. Multiplier (Q) = $-9 = 10111$; Multiplicand (M) = $+12 = 01100$

A	Q	Q_1	M	
00000	10111	0	01100	Nilai awal
10100	10111	0	01100	$A \leftarrow A - M$
11010	01011	1	01100	SAR
11101	00101	1	01100	Langkah 2 SAR
11110	10010	1	01100	SAR
01010	10010	1	01100	Langkah 3 $A \leftarrow A + M$
00101	01001	0	01100	SAR
11001	01001	0	01100	Langkah 4 $A \leftarrow A - M$
11100	10100	1	01100	SAR

Nilai akhir/hasil

8. Dividend (Q) = $25 = 11001$; Divisor (M) = $7 = 00111$

A	Q	M	
00000	11001	00111	Nilai awal
00001	10010	00111	Geser kiri
11010	10010	00111	$A \leftarrow A - M$
00001	10010	00111	Restoring
00011	00100	00111	Geser kiri
11100	00100	00111	$A \leftarrow A - M$
00011	00100	00111	Restoring
00110	01000	00111	Geser kiri
11111	01000	00111	$A \leftarrow A - M$
00110	01000	00111	Restoring
01100	10000	00111	Geser kiri
00101	10000	00111	$A \leftarrow A - M$
00101	10001	00111	Set $Q_0 = 1$
01011	00010	00111	Geser kiri
00100	00010	00111	$A \leftarrow A - M$
00100	00011	00111	Set $Q_0 = 1$

Sisa Hasil bagi

JAWABAN SOAL-SOAL LATIHAN BAB 5

- Operasi pengambilan instruksi (*fetch instruction*) melibatkan pembacaan memori, dan transfer register (PC, MAR, MDR dan IR) yaitu berturut-turut:
 - $\text{MAR} \leftarrow \text{PC}$; 10 ns (operasi register)
 - $\text{MDR} \leftarrow \text{Mem}[\text{MAR}]$; 80 ns (operasi memori)
 - $\text{PC} \leftarrow \text{PC} + 1$; (operasi ini paralel dengan pembacaan memori)
 - $\text{IR} \leftarrow \text{MDR}$; 10 ns (operasi register)
 Jadi total waktu yang dibutuhkan untuk pengambilan (*fetch*) = $10 + 80 + 10 = 100$ ns
- Siklus instruksi LDA: *fetch, decode, fetch operand* (dari memori)
 - Siklus instruksi STA: *fetch, decode, store operand* (ke memori)
 - Siklus instruksi NOOP: *fetch, decode*
 - Siklus instruksi HALT: *fetch, decode, reset flip-flop RUN*
 - Siklus instruksi BUN (*branch unconditionally*): *fetch, decode, salin alamat branch ke PC, lompat ke alamat branch.*
 Dari rincian siklus instruksi di atas terlihat instruksi BUN secara kuantitatif paling lama dibandingkan dengan yang lain.
- Dari rincian siklus instruksi pada no.2 di atas terlihat instruksi NOOP secara kuantitatif paling singkat dibandingkan dengan yang lain.
- (a) MOV AL,BL ; $\text{AL} \leftarrow \text{BL}$
 (b) ADC AL,45h ; $\text{AL} \leftarrow \text{AL} + 45h + \text{CF}$
 (c) SUB BL,CL ; $\text{BL} \leftarrow \text{BL} - \text{CL}$
 (d) INC CX ; $\text{CX} \leftarrow \text{CX} + 1$

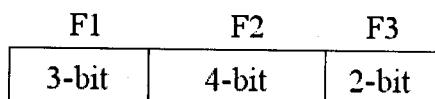
JAWABAN SOAL-SOAL LATIHAN BAB 6

- (a) Mikrooperasi adalah suatu operasi yang mendasar (operasi elementer) dari sebuah komputer digital
 (b) Mikroinstruksi adalah suatu instruksi yang tersimpan di dalam memori kontrol
 (c) Mikroprogram adalah urut-urutan dari sejumlah mikroinstruksi
 (d) Mikrokode sama dengan mikroprogram

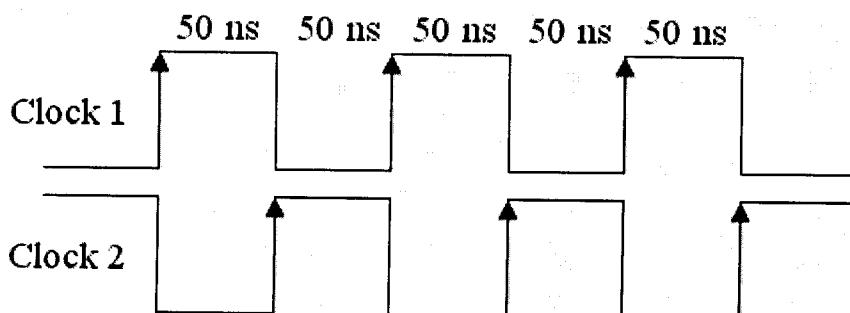
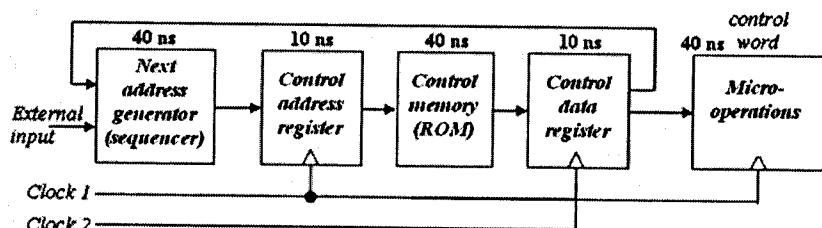
- Terdapat lebih dari satu solusi. Satu di antaranya adalah sebagai berikut:

isi ROM	alamat
A B C D	0 0 0 0
1 0 0 0	0 0 0 0
0 0 1 1	0 0 0 1
0 1 0 0	0 0 1 0
0 0 1 1	0 0 1 1
0 1 0 0	0 1 0 0
0 0 1 1	0 1 0 1
0 1 0 0	0 1 1 0
0 0 1 1	0 1 1 1
0 0 1 1	1 0 0 0
0 0 1 1	1 0 0 1
0 0 1 1	1 0 1 0
0 0 1 1	1 0 1 1
0 0 1 1	1 1 0 0
0 0 1 1	1 1 0 1
0 0 1 1	1 1 1 0
0 0 1 1	1 1 1 1

- 5 bit menetapkan $2^5 - 1 = 31$ mikrooperasi, 4 bit menetapkan $2^4 - 1 = 15$ mikrooperasi. Jadi jumlah mikrooperasi yang mungkin adalah = 46 mikrooperasi
- a) untuk F1 dibutuhkan sebesar $\log_2 4 = 2$ bit; F1 dibutuhkan sebesar $\log_2 4 = 2$ bit; F3 dibutuhkan sebesar $\log_2 3 \approx 2$ bit; F4 dibutuhkan sebesar $\log_2 12 \approx 4$ bit; dan untuk F5 dibutuhkan sebesar $\log_2 21 \approx 5$ bit, Jadi total panjang mikroinstruksi yang dibutuhkan = $2 + 2 + 2 + 4 + 5 = 15$ bit
- Format mikroinstruksi tersebut adalah sbb:



6.



- Frekuensi clock maksimum yang dapat digunakan oleh kontrol = $1 / (40 \text{ ns} + 10 \text{ ns} + 40 \text{ ns} + 10 \text{ ns}) = 1 / 100 \text{ ns} = 100 \text{ MHz}$
- Frekuensi clock yang akan diperlukan jika *control data register* tidak digunakan = $1 / (40 \text{ ns} + 10 \text{ ns} + 40 \text{ ns}) = 1 / 90 \text{ ns} = 11,1 \text{ MHz}$

JAWABAN SOAL-SOAL LATIHAN BAB 7

1. Memori ROM dengan kapasitas 64KB artinya sama dengan 2^{16} Byte. Untuk membentuk larik register persegi maka dibuat arsitektur internal dengan konfigurasi $2^8 \times 2^8$ byte. Sehingga ROM ini memerlukan dekoder alamat 8 ke 256 sebanyak dua buah untuk dapat mengakses setiap baris dan kolom (terdapat $64 \times 1024 = 256 \times 256 = 65536$ lokasi memori).

2. Karena waktu akses 100 ns, maka untuk penyegaran 256 baris dibutuhkan waktu $256 \times 100 \text{ ns} = 25,6 \mu\text{s}$. Waktu ini digunakan di dalam penyegaran setiap durasi 2 ms. Karena itu *refresh overhead* = Waktu yang digunakan untuk penyegaran semua baris : interval penyegaran

$$= (25,6 \times 10^{-6}) : (2 \times 10^{-3}) = 12,8 \times 10^{-3} \text{ s}$$

$$100\% = 1,28\%$$

Jadi *refresh overhead* sekitar = 1,3 %
3. Waktu akses = waktu siklus + waktu refresh = $80 + 60 = 140 \text{ ns}$
Waktu penyegaran = $1024 \times 140 \text{ ns} = 143,36 \mu\text{s}$
Karena itu *refresh overhead* = Waktu yang digunakan untuk penyegaran semua baris / interval penyegaran

$$= (143,36 \times 10^{-6}) / (16,4 \times 10^{-3}) = 102,4 \times 10^{-3} \text{ s}$$

$$100\% = 10,2\%$$

Jadi *refresh overhead* = 10,2 %
4. Jumlah IC yang diperlukan = $4 \text{ MB} / 256 \text{ K nibble} = (2^{22} \times 8) / (2^{18} \times 4) = 2^4 \times 2 = 32$ buah
5. a) Kapasitas total memori yang dapat diakses = $2^{26} = 64 \text{ MB}$
b) Jumlah IC yang dibutuhkan = $64 \text{ MB} / 4 \text{ M} \times 4 = (2^{26} \times 8) / (2^{22} \times 4) = 2^4 \times 2 = 32$ buah
c) Karena jumlah bank yang terbentuk = $64 \text{ M} / 4 \text{ M} = 16$ bank, jadi dekoder alamat yang diperlukan adalah dekoder 4 ke 16 sebanyak satu buah.
d) Rentang alamat Bank-0 = 0000000H – 03FFFFFFH
Rentang alamat Bank-1 = 0400000H – 07FFFFFFH
Rentang alamat Bank-2 = 0800000H – 0BFFFFFFH
Rentang alamat Bank-3 = 0C00000H – 0FFFFFFH
Rentang alamat Bank-4 = 1000000H – 13FFFFFFH
Rentang alamat Bank-5 = 1400000H – 17FFFFFFH
Rentang alamat Bank-6 = 1800000H – 1BFFFFFFH
Rentang alamat Bank-7 = 1C00000H – 1FFFFFFH
Rentang alamat Bank-8 = 2000000H – 23FFFFFFH
Rentang alamat Bank-9 = 2400000H – 27FFFFFFH
Rentang alamat Bank-10 = 2800000H – 2BFFFFFFH
Rentang alamat Bank-11 = 2C00000H – 2FFFFFFH

Rentang alamat Bank-12 = 3000000H – 33FFFFFFH

Rentang alamat Bank-13 = 3400000H – 37FFFFFFH

Rentang alamat Bank-14 = 3800000H – 3BFFFFFFH

Rentang alamat Bank-15 = 3C00000H – 3FFFFFFH

- e). Setiap lokasi IC memberikan 1 bit. Karena itu untuk membentuk satu byte diperlukan 8 buah IC. Karena setiap IC mempunyai 16 K lokasi, maka 8 buah IC memberikan kapasitas 16 KB. Sehingga kita memerlukan $4 \times 8 = 32$ IC untuk membentuk 64 KB RAM, memori ini disusun dalam empat bank.
7. a. Kapasitas total = 32 KB = $32 \text{ K} \times 8 \text{ bit} = 256 \text{ K bit}$. Karena digunakan 16 IC, setiap IC memberikan 256 K bit : $16 = 16 \text{ K bit}$, karena itu kapasitas setiap IC adalah 16 K bit. Karena 14 bit alamat memberikan 16 kilo lokasi, maka setiap chip mempunyai 16 K lokasi. Karena itu organisasi IC adalah $16 \text{ K} \times 1$, yaitu kapasitas setiap IC adalah 16 K bit.
- b. Modul RAM terbentuk dari dua bank, setiap bank memberikan 16 KB RAM.
- c. Untuk mengalami 32 KB memori, diperlukan 15 bit alamat, yang ditunjukkan dengan A14-A0. A14 memilih bank. Bila A14 = 0, bank 0 terpilih, bila A14 = 1 maka bank 1 yang terpilih. Range alamatnya adalah:
- Bank 0: 0000 0000 0000b sampai 0011 1111 1111 1111b;
(0000h sampai 3FFFh)
- Bank 1: 4000h sampai 7FFFh

JAWABAN SOAL-SOAL LATIHAN BAB 8

1. Kasus 1: tanpa prefetch instruksi

Waktu siklus instruksi = waktu pengambilan + waktu eksekusi

$$= 20 \text{ ns} + t_e$$

Anggap terdapat 100 instruksi, total waktu eksekusi:

$$= 100 \times (t_e + 20) \text{ ns} = (100 t_e + 2000) \text{ ns}$$

Kasus 2: ada prefetch instruksi

Dalam 100 instruksi, 10 adalah instruksi cabang dan lainnya adalah instruksi biasa. Selama eksekusi program, pengambilan instruksi oleh prosesor hanya dibutuhkan 10 kali yaitu setelah setiap instruksi JUMP. Untuk sisanya 90 instruksi, prosesor tidak mempunyai waktu

pengambilan (karena antrian instruksi) dan hanya mempunyai waktu eksekusi.

$$\begin{aligned}\text{Total waktu eksekusi} &= \text{waktu eksekusi selama 90 instruksi tanpa lompatan} + \text{waktu siklus instruksi untuk 10 instruksi lompatan}. \\ &= 90 t_e + 10 \times (t_e + 20) \text{ ns} \\ &= 100 t_e + 200 \text{ ns}\end{aligned}$$

Perbedaan dalam waktu eksekusi total = 1800 ns terhadap 100 instruksi

Perolehan rata-rata per instruksi karena prefetch instruksi = 18 ns

2. Ruang alamat = 1048576 byte
Karena itu, jumlah bit alamat = 20
Karena digunakan *interleaving 4-way*, maka setiap modul akan mempunyai 256 K lokasi.
Alamat memori 20 bit dari CPU digunakan sebagai:
A19—A2 : untuk memilih lokasi memori dalam modul
A0—A1 : untuk pemilihan bank
Dekoder 2-ke-4 digunakan untuk mendekodekan bit-bit memilih bank..
Bandwidth tanpa interleaving = 4 byte/80 ns = 50 MB/sec
Bandwidth dengan interleaving = 4 byte/20 ns = 200 MB/sec
Jadi *bandwidth* efektifnya = $(200 - 50) \text{ MB/sec} = 150 \text{ MB/sec}$
3. Sebuah memori mempunyai waktu akses 50 ns. Hitung perolehan kinerja karena *buffer tulis (write buffer)* jika siklus memori untuk baca dan tulis berada dalam ratio 3:1
Anggap ada 100 operasi memori. Berarti 75 (= $\frac{3}{4} \times 100$) di antaranya untuk operasi baca dan 25 (= $\frac{1}{4} \times 100$) operasi tulis.
Total waktu akses memori tanpa *write buffer* = $25 \times 50 \text{ ns} = 1250 \text{ ns}$
Total waktu akses memori dengan *write buffer* = $1 \times 50 \text{ ns} = 50 \text{ ns}$
Perolehan kinerja = $(1250 / 50) = 25$ kali
4. a) *Hit ratio* 0.9 artinya dari 100 akses memori cache hanya 90 yang berhasil (HIT), dan 10 yang gagal (MISS). Karena dianggap terjadinya MISS pada perpindahan informasi pertama dari memori utamake cache dan kemudian dari cache ke CPU, maka:
total waktu MISS = $(10 \times 10t) + (10 \times t) = 110t$ (t dalam satuan waktu).
total waktu HIT = $(90 \times t) = 90t$
Total akses memori (HIT + MISS) = $110t + 90t = 200t$

Total waktu akses memori utama (tanpa memori cache) = $100 \times 10t$
 $= 1000t$

Jadi peningkatan kinerja pada fetch instruksi karena memori cache = $1000t / 200t = 5$ kali

- b. MISS setengah (*hit ratio = 0,5*) artinya dari 100 akses memori cache hanya 50 yang berhasil

(HIT), dan 50 yang gagal (MISS). Karena dianggap terjadinya MISS pada perpindahan informasi pertama dari memori utama ke cache dan kemudian dari cache ke CPU, maka:

total waktu MISS = $(50 \times 10t) + (50 \times t) = 550t$ (*t* dalam satuan waktu).

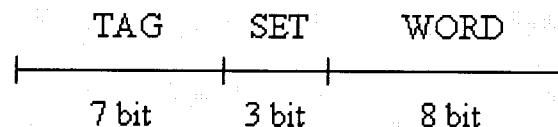
total waktu HIT = $(50 \times t) = 50t$

Total akses memori (HIT + MISS) = $550t + 50t = 600t$

Total waktu akses memori utama (tanpa memori cache) = $100 \times 10t$
 $= 1000t$

Jadi peningkatan kinerja pada fetch instruksi karena memori cache = $1000t / 600t = 1,667$ kali

5. Kapasitas memori utama = 1024 blok $\times 256$ word = 2^{18} . Jadi alamat memori utama = $\log_2 2^{18} = 18$
 WORD = $\log_2 256 = \log_2 2^8 = 8$ bit;
 SET = $\log_2 8 = \log_2 2^3 = 3$ bit;
 TAG = Alamat memori utama – SET – WORD = $18 - 3 - 8 = 7$ bit



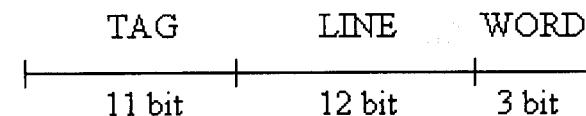
6. Kapasitas memori utama = 64 MB = 2^{26} , kapasitas memori cache = 32 KB = 2^{15} , setiap baris dalam memori cache menyimpan 8 byte (2^3).

- a. – Format alamat memori untuk *direct mapping*

WORD = $\log_2 8 = \log_2 2^3 = 3$ bit

LINE = $2^{15} / 2^3 = 12$ bit

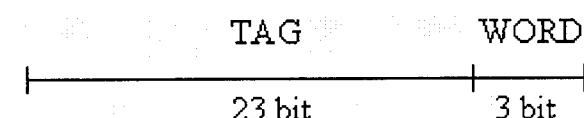
TAG = alamat memori utama – LINE – WORD = $26 - 12 - 3 = 11$ bit



- Format alamat memori untuk *fully associative mapping*

WORD = 3 bit

TAG = alamat memori utama – WORD = $26 - 3 = 23$ bit

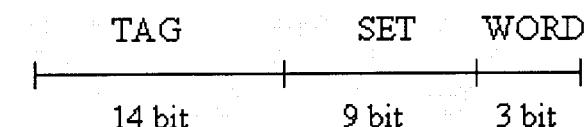


- Format alamat memori untuk *set associative mapping (8-way)*

WORD = 3 bit

SET = $2^{12} / 8 = 9$ bit

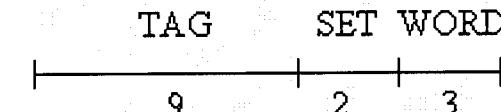
TAG = alamat memori utama – SET – WORD = $26 - 9 - 3 = 14$ bit



- b. Jumlah baris yang terdapat dalam cache = $2^{15} / 2^3 = 12$ bit

7. a) Alamat memori utama = $2K \times 8$ word = 16 K word = $\log_2 2^{14} = 14$ bit
 Field untuk WORD = $\log_2 8 = \log_2 2^3 = 3$ bit
 Field untuk SET = $\log_2 4 = \log_2 2^2 = 2$ bit
 TAG = alamat memori utama – SET – WORD = $14 - 2 - 3 = 9$ bit

Format alamat memori utama:



- b) Karena loop ada tiga kali, maka akses memori utama pada alamat 8 sampai 51 sebanyak satu kali, dan pada memori cache sebanyak dua kali. Sehingga MISS = 1 x dan HIT = 2 x.

Jadi $hit\ rate = \frac{\text{jumlah akses memori cache (HIT)}}{\text{total akses}} = \frac{2}{3} = 0.667$

8. Diketahui:

Prosesor dengan rentang pengalamatan 1 megabyte = 2^{20} byte \Rightarrow
alamat = 20 bit

Kapasitas memori cache = 2^{12} = 4 KB

Kapasitas setiap blok = 64 byte = 2^6 (jadi WORD = $\log_2 2^6 = 6$ bit)

Digunakan pemetaan langsung

Jadi:

- Jumlah baris dalam memori cache = kapasitas memori cache / kapasitas per blok = $2^{12} / 2^6 = 2^6 = 64$ baris (LINE = 6 bit)
- Jumlah bit untuk TAG = alamat memori utama – LINE – WORD = $20 - 6 - 6 = 8$ bit
- CARA 1:**

Gunakan formula: $I_{cm} = b_{mm} \text{ modulo } c$

di mana I_{cm} = nomor baris cache
 b_{mm} = nomor blok memori utama
 c = jumlah total baris cache

3FB0Ah. = 260874 desimal

b_{mm} = nomor alamat div kapasitas word
(div : pembagian integer)

$b_{mm} = 260874 \text{ div } 64 = 4076$

$I_{cm} = 4076 \text{ modulo } 64$

$I_{cm} = 44$

Jadi item tersebut dipetakan pada baris 44 dalam memori cache

CARA 2:

3FB0Ah = 00111111 101100 001010b di
mana $101100_b = 44_d$

TAG LINE WORD

9. Gunakan data Soal no.8, sehingga:

- Karena menggunakan pemetaan asosiatif penuh (*fully associative mapping*) maka tidak diperlukan pembagian baris dalam cache

- Jumlah bit untuk TAG = alamat memori utama – WORD = $20 - 6 = 14$ bit
- Karena menggunakan pemetaan asosiatif penuh, maka item bebas dipetakan di mana saja di dalam memori cache

10. Diketahui:

Prosesor dengan rentang pengalamatan 1 megabyte = 2^{20} byte \Rightarrow
alamat = $\log_2 2^{20} = 20$ bit

Kapasitas memori cache = 4 KB = 2^{12}

Kapasitas setiap blok = 64 byte = 2^6 (WORD = 6 bit)

Kapasitas setiap set = 16 blok

Digunakan pemetaan asosiatif set

Jadi:

- Jumlah baris dalam memori cache = kapasitas memori cache / kapasitas per blok = $2^{12} / 2^6 = 2^6 = 64$ baris (LINE = 6 bit)
- Jumlah set dalam memori cache = jumlah baris / kapasitas set = $64 / 16 = 2^2 = 4$ set
- Jumlah bit untuk TAG = alamat memori utama – SET – WORD = $20 - 2 - 6 = 12$ bit
- CARA 1:**

$$g = b \text{ modulo } s$$

di mana g = nomor set memori cache
 b = nomor blok memori utama
 s = jumlah set memori cache

3FB0Ah. = 260874 desimal

b = nomor alamat div kapasitas word (

div : pembagian integer)

$b = 260874 \text{ div } 64 = 4076$

$g = 4076 \text{ modulo } 4$

$g = 0$

Jadi item tersebut dipetakan pada set nomor 0 dalam memori cache

CARA 2:

3FB0Ah = 00111111011 00 001010b
TAG SET WORD

11. Diketahui:

Prosesor dengan rentang pengalamanan 16 MB = 2^{24} byte \Rightarrow alamat = 24 bit

Kapasitas memori cache = 2^{13} = 8 KB

Jumlah blok = $1048576_{10} = 1M = 2^{20}$ blok

Digunakan pemetaan asosiatif set 4-way

Jadi:

a) Kapasitas setiap blok = kapasitas memori utama / jumlah blok = $2^{24} / 2^{20} = 2^4 = 16$ byte (WORD = 4 bit),

jumlah baris = $2^{13} / 2^4 = 2^9$ baris, SET = jumlah baris / $2^4 = 2^9 / 2^4 = 2^5$ (SET = 5 bit)

TAG = alamat memori utama – SET – WORD = $24 - 5 - 4 = 15$ bit

b) Jumlah baris dalam memori cache = 512 baris (= 2^9 baris)

c) Jumlah set dalam memori cache = 32 set (= 2^5 set)

d) Jumlah byte dalam setiap baris cache = jumlah byte dalam setiap blok = 16 byte

e) Gunakan formula $g = b \text{ modulo } s$

diketahui nomor blok 125_{10}

$$g = 125 \text{ modulo } 32$$

$$g = 29$$

Jadi item tersebut dipetakan pada set nomor 29 dalam memori cache

12. Diketahui:

Alamat logika = 24 bit

Alamat fisik = 12 bit

Ukuran page = 2 K = 2^{11}

Jadi:

Jumlah page = alamat logika / ukuran page = $2^{24} / 2^{11} = 2^{13} = 8$ K

Jumlah blok (frame) memori utama = alamat fisik / ukuran page = $2^{12} / 2^{11} = 2^1 = 2$

13. Diketahui:

Alamat logika dengan jumlah segmen = 64 segmen

Ukuran segmen = 64 K word

Alamat fisik dengan jumlah page = 1 K dengan masing-masing 4 K word

Jadi:

Alamat logika = ukuran segmen x jumlah segmen = $64\text{ K} \times 64 = 2^{22}$
word = 22 bit

Ukuran page = 1 K = 2^{10} = 10 bit

Alamat fisik = 2^{12} = 12 bit

Page field alamat logika = $2^{22} / 2^{10} = 2^{12} = 12$ bit

Offset field alamat logika = $22 - 12 = 10$ bit

Offset field alamat fisik = Offset field alamat logika = 10 bit

Frame field alamat fisik = $12 - 10 = 2$ bit

Format alamat logika

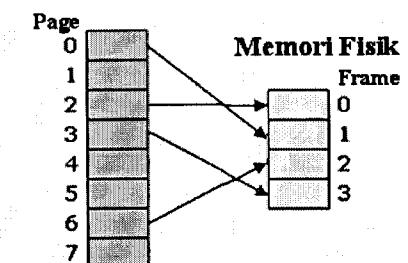
Page	Offset
12	10

Format alamat fisik

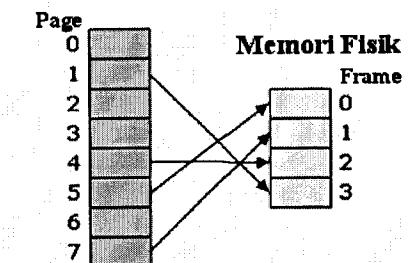
Frame	Offset
2	10

14.

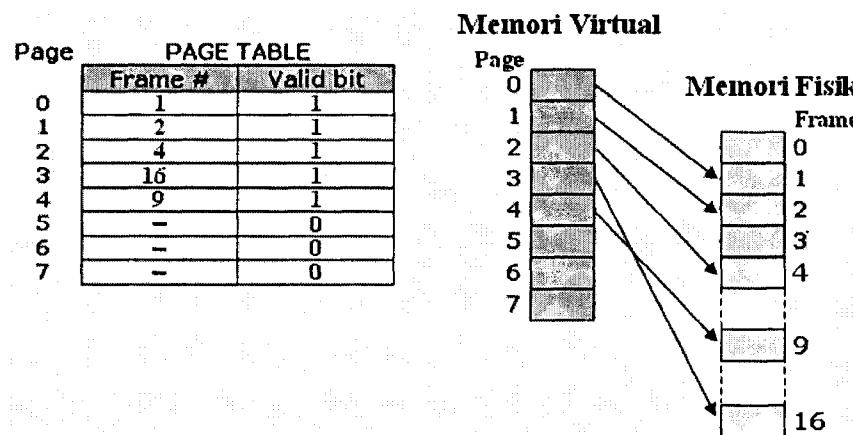
Page	PAGE TABLE	
	Frame #	Valid bit
0	1	1
1	-	0
2	0	1
3	3	1
4	-	0
5	-	0
6	2	1
7	-	0

Memori Virtual**15.**

Page	PAGE TABLE	
	Frame #	Valid bit
0	-	0
1	3	1
2	-	0
3	-	0
4	2	1
5	0	1
6	-	0
7	1	1

Memori Virtual

16. a) Alamat virtual = 16 MB = $\log_2 2^{24} = 24$ bit
 b) Alamat fisik = 2 MB = $\log_2 2^{21} = 21$ bit
 c) Jumlah maksimum entry dalam sebuah page table = $2^{24} / 1024 = 2^{24} / 2^{10} = 2^{14}$ entry
 Ukuran page = $1024 = 2^{10}$
 Page field = 14 bit, offset field = $24 - 14 = 10$ bit

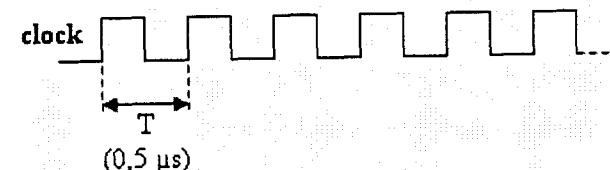


- d) ← Alamat virtual $1524_{10} = 0005F4h$
 Page 1
 ← Alamat Fisik $0009F4h = 2548_{10}$
 Frame 2
- e) ← Alamat virtual = $00000h = 0_{10}$
 Page 0
 ← Alamat Fisik $1024_{10} = 000400h$
 Frame 1

JAWABAN SOAL-SOAL LATIHAN BAB 9

1. Perhatikan gambar clock berikut:

$$f_{clock} = 12 \text{ MHz} \rightarrow T = 1/f = 1/2 \times 10^6 = 0,5 \mu\text{s}$$



Karena pada satu siklus bus ada dua byte yang diakses (dengan zero wait state) dalam dua periode clock. Maka bandwidth maksimum yang ditawarkan sistem adalah:

$$\text{Bandwidth maks.} = 2 \text{ Byte} / (0,5 + 0,5) \mu\text{s} = 2 \text{ Byte}/1 \mu\text{s} = 2 \times 10^6 \text{ Byte/sec} = 2 \text{ MB/sec.}$$

2. Sistem PCI 64 bit yang beroperasi pada frekuensi 60 MHz, bandwidth sebagai berikut::
 $\text{Bandwidth} = (64/8) \text{ Byte} \times 66 \times 10^6 \text{ Hz} = 528 \times 10^6 \text{ Byte/sec} = 528 \text{ MB/sec}$

JAWABAN SOAL-SOAL LATIHAN BAB 10

1. Jumlah bit yang ditulis ke tampilan setiap detik adalah: $1024 \times 1024 \times 60 = 62914560$ bit/s

Waktu maksimum yang diizinkan untuk menulis setiap pixel:
 $1/62914560 \text{ bit/s} \times 10^9 \text{ ns/s} = 15.9 \text{ ns}$

2. a) Kapasitas disk = $4 \times 1024 \times 128 \times 512 = 2^2 \times 2^{10} \times 2^7 \times 2^9 = 2^{28} = 256 \text{ MB}$

$$b) T_a = T_s + \frac{1}{2r} + \frac{b}{Nr}$$

$$T_a = 5 \text{ ms} = 5 \times 10^{-3} \text{ s}$$

$$r = 5000 \text{ rpm} = 5000/60 \text{ rotasi per detik} = 83.33 \text{ rotasi per detik}$$

$$b = 1 \text{ byte}$$

$$N = 128 \times 512 = 2^{16} = 65536 \text{ byte}$$

$$T_a = T_s + \frac{1}{2r} + \frac{b}{Nr} = 5 \times 10^{-3} \text{ s} + 1/(2 \times 83.33) \text{ s} + 1/(65535 \times 83.33) \text{ s} = 11.0001831 \text{ ms}$$

3. a) Kapasitas disk = $5 \times 1024 \times 256 \times 512 = 5 \times 2^{10} \times 2^8 \times 2^9 = 5 \times 2^{27} = 640 \text{ MB}$

$$T_a = T_s + \frac{1}{2r} + \frac{b}{Nr}$$

$$T_a = 8 \text{ ms} = 8 \times 10^{-3} \text{ s}$$

$$r = 7500 \text{ rpm} = 7500/60 = 125 \text{ rotasi per detik}$$

$$b = 1 \text{ byte}$$

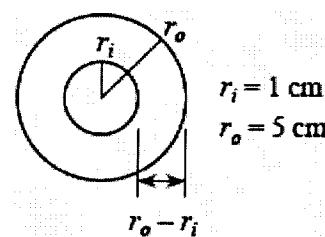
$$N = 256 \times 512 = 2^{17} = 131072 \text{ byte}$$

$$T_a = T_s + \frac{1}{2r} + \frac{b}{Nr} = 8 \times 10^{-3} \text{ s} + 1/(2 \times 125) \text{ s} + 1/(131072 \times 125) \text{ s}$$

$$= 12.00006103515625 \text{ ms}$$

- b) Yang lebih cepat adalah yang No. 1b yaitu: 11.0001831 ms

4. a)



Berdasarkan gambar di atas maka lebar area penyimpanan = $r_o - r_i = 5 \text{ cm} - 1 \text{ cm} = 4 \text{ cm}$

Jumlah track dalam disk = $4 \text{ cm} \times 10 \text{ mm/cm} \times 1/0.1 \text{ track/mm} = 400 \text{ track}$

Track terdalam mempunyai kapasitas penyimpanan yang terkecil, sehingga semua track tidak akan menyimpan data yang lebih besar dari track terdalam. Jumlah bit yang tersimpan pada track terdalam adalah: $10000 \text{ bit/cm} \times 2\pi \times 1 \text{ cm} = 62832 \text{ bit}$.

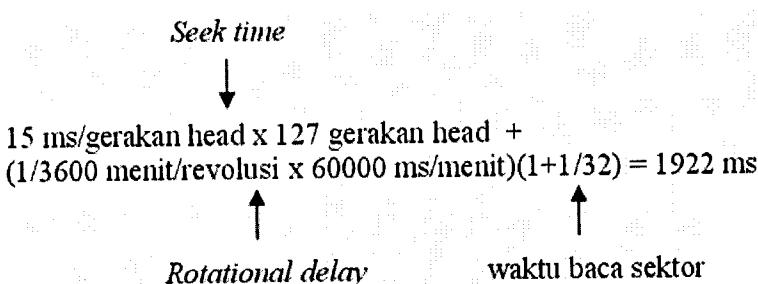
Penyimpanan per permukaan adalah: $62,832 \text{ bit/track} \times 400 \text{ track/permukaan} = 25.13 \times 10^6 \text{ bit/permukaan}$

Peyimpanan pada disk adalah: $2 \text{ permukaan/disk} \times 25.13 \times 10^6 \text{ bit/permukaan} = 50.26 \text{ Mbit/disk}$

- b) $62832 \text{ bit/track} \times 1 \text{ track/revolusi} \times 3600 \text{ revolusi/menit} \times 1/60 \text{ menit/s} = 3.77 \text{ Mbit/sec.}$

5. Dalam keadaan terburuk, head akan harus bergerak antara dua track ekstrem, pada titik mana semua revolusi harus dibuat di atas awal sektor dengan posisi head.

Semua sektor harus kemudian digerakkan di bawah head. Waktu akses keadaan terburuk untuk sebuah sektor berarti terdiri atas tiga bagian:



JAWABAN SOAL-SOAL LATIHAN BAB 11

- Jumlah siklus dalam pipeline = $n + m - 1 = 6 + 200 - 1 = 205 \text{ siklus}$
- Total waktu yang dibutuhkan untuk memproses 200 instruksi dalam pipeline = $nmt_c = 6 \times 200 \times 10 \text{ ns} = 2050 \text{ ns}$

$$18. Speed up aktual = Speed up$$

$$Speed-up = \frac{\text{waktu, pemrosesan non-pipeline}}{\text{waktu, pemrosesan pipeline}} = \frac{nmt_c}{(n+m-1)t_c} = \frac{nm}{n+m-1}$$

$$Speed up = (6 \times 200) / (6 + 200 - 1) = 5,8536$$

$$d) Speed up maksimum = n = 6$$

$$Throughput = \text{jumlah tugas yang dieksekusi per unit waktu} = \frac{m}{(n+m-1)t_c}$$

$$Throughput = 200 / ((6 + 200 - 1) \times 10 \text{ ns}) = 0,0975 \text{ ns}$$

2. Jika tidak instruksi cabang dalam program, $speed-up = n = \text{jumlah tingkat} = 5$

$$p = 0,2$$

$$q = 1,0$$

$$n = 5$$

$$CPI = 1 + pq(n - 1) = 1 + 0,2 \times 1,0 \times (5-1) = 1,8$$

Prosesor non-pipeline memerlukan 5 clock per instruksi

$$Speed-up = 5/2 = 2,5$$

3.

$$p = 0,2$$

$$q = 0,6$$

$$n = 4$$

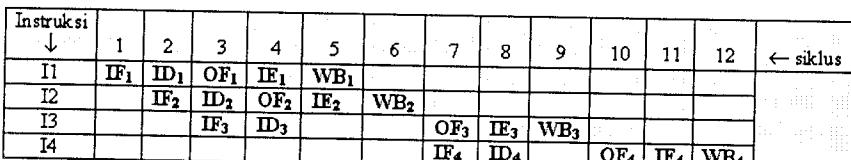
$$CPI = 1 + pq(n - 1) = 1 + 0,2 \times 0,6 \times (4-1) = 1,36$$

Prosesor non-pipeline memerlukan 4 clock per instruksi

$$Speed-up = 4/2 = 2$$

Speed-up maksimum jika tidak ada pencabangan = $n = 4$

4. Space-time diagram yang dihasilkan adalah sebagai berikut:



Tampak dari space-time diagram di atas terjadi penundaan pembacaan operand oleh instruksi I3 selama dua siklus clock karena adanya ketergantungan data (data hazard) dari I2. Sedangkan pada instruksi I4 juga terjadi penundaan pembacaan operand karena adanya ketergantungan data (data hazard) dari I3

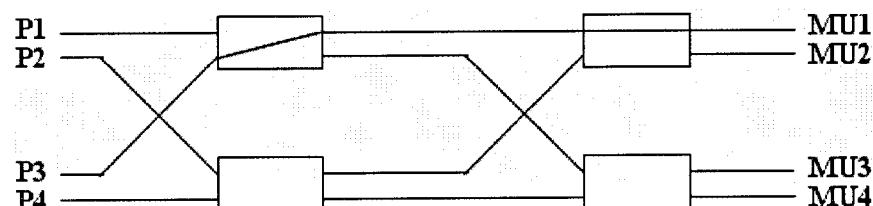
JAWABAN SOAL-SOAL LATIHAN BAB 12

1. Jumlah titik switch yang berada dalam sebuah jaringan crossbar switch yang menghubungkan p prosesor ke m modul memori = $p \times m$
2. Jumlah stage = $\log_2 n$, jumlah switch per stage = $n/2$

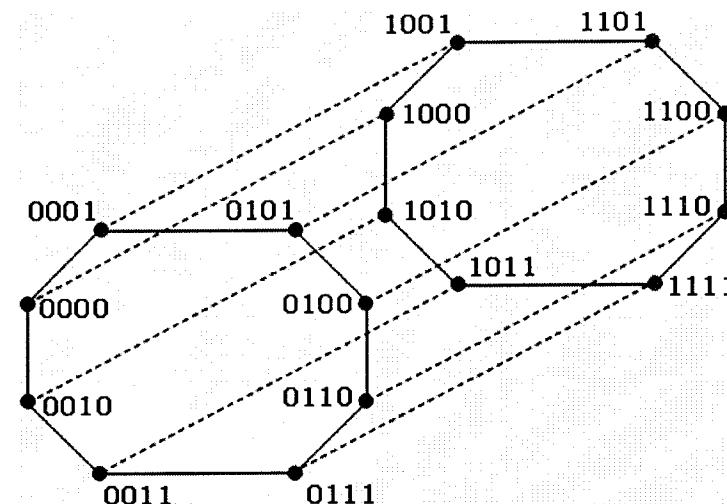
3.
 - ① Jika baris satu, kolom dua terputus → lintasan yang akan terputus adalah: P1 – MU1; P1 – MU2; P1 – MU3; P1 – MU4; P5 – MU1; P5 – MU2; P5 – MU3; P5 – MU4; P3 – MU1; P3 – MU2; P3 – MU3; P3 – MU4; P7 – MU1; P7 – MU2; P7 – MU3; P7 – MU4.

- ② Jika baris dua, kolom tiga terputus → lintasan yang akan terputus adalah: P1 – MU3; P1 – MU4; P5 – MU3; P5 – MU4; P3 – MU3; P3 – MU4; P7 – MU3; P7 – MU4; P2 – MU3; P2 – MU4; P6 – MU3; P6 – MU4; P4 – MU3; P4 – MU4; P8 – MU3; P8 – MU4.

4. Lintasan masukan P3 ke keluaran MU1 adalah seperti yang terlihat pada gambar berikut:



5. Bangun yang diperoleh adalah bangun oktagon berikut:



PROFIL PENULIS

Penulis lahir di Makassar 25 Januari 1963, menyelesaikan studi jenjang S1 di Universitas Hasanuddin Makassar tahun 1988 pada bidang Elektro-Teknik Telekomunikasi & Elektronika, kemudian pada tahun 1999 menyelesaikan jenjang S2 bidang Elektro-Teknik Biomedika di Institut Teknologi Bandung, ITB.

Penulis menjadi staf dosen sejak tahun 1991 dan saat ini mengajar dan membimbing tugas akhir mahasiswa pada Jurusan Teknik Komputer, Universitas Komputer Indonesia (UNIKOM) Bandung. Bidang yang ditekuni adalah Teknik Kontrol Elektronika-*Embedded System*. Mata kuliah yang diajarkan meliputi *Organisasi & Arsitektur Komputer*, *Sistem Mikroprosesor/Mikrokontroler*, serta *Antarmuka Komputer*.