Terminologi

Istilah-istilah yang sering digunakan dalam dunia komputer :

1. **Sistem bilangan** adalah metode yg menyepadankan suatu besaran dengan suatu simbol tertentu.
   Contoh: manusia menggunakan sistem bilangan desimal (10). Ini berarti menusia memiliki 10 (sepuluh) buah simbol untuk menyatakan 10 buah besaran, yaitu : '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'.
2. Sistem bilangan biner (**Binary**) menggunakan 2 (dua) buah simbol untuk merepresentasikan 2 besaran, yaitu : '0' dan '1'. Sebuah simbol biner sering disebut sebagai **bit** (binary digit)
3. Sistem bilangan **Hexadecimal** menggunakan 16 (enambelas) buah simbol untuk merepresentasikan 16 besaran, yaitu : '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'.
4. Kesatuan 4 simbol biner yg merepresentasikan suatu informasi disebut dgn **nibble**
5. Kesatuan 8 simbol biner yg merepresentasikan suatu informasi disebut dgn **byte**
6. Kesatuan 16 simbol biner yg merepresentasikan suatu informasi disebut dgn **word**
7. Dunia digital menggunakan 2 buah tegangan (0 volt dan +5 volt). Karena hanya terdapat 2 besaran, kedua besaran tersebut akan direpresentasikan dengan sistem bilangan biner ('0' dan '1')
8. **Transistor** adalah komponen aktif yang digunakan sebagai penguat sinyal (dalam dunia analog) atau sebagai saklar (dalam dunia digital). Tergantung dari bahan dan media pembuatannya, transistor digital dibedakan menjadi pMOS (positive Metal Oxide Semiconductor), nMOS (negative MOS), dan TTL (Transistor^2 Logic)
9. Gerbang logika (**Logic Gate**) adalah komponen dasar dari rangkaian digital yang dibentuk dari beberapa transistor digital untuk membentuk suatu fungsi tertentu (contoh : AND, OR, XOR)
10. Integrated Circuit (**IC**) adalah sebuah komponen yang merupakan gabungan komponen-komponen untuk menjalankan suatu fungsi khusus. Ada beberapa jenis IC dilihat dari komponen yang digabungkan, yaitu :
    - IC analog: integrasi komponen analog (contoh : Operational Amplifier)
    - IC hybrid: integrasi miniatur komponen analog (contoh : Integrated Amplifier)
    - IC digital: integrasi komponen digital ke dalam plat MOS
    - **ASIC** (Application Specific IC): integrasi komponen yang dirancang untuk melakukan suatu fungsi khusus (contoh : Prosesor pada HandPhone)
11. Very Large Scale Integration (**VLSI**) adalah IC digital yang memiliki jumlah transistor digital lebih dari 100.000 buah (contoh : Microprocessor)
12. **Microprocessor** (atau CPU = Central Processing Unit) adalah sebuah VLSI yang didesain khusus untuk memproses/mengerjakan tugas-tugas standar. uP adalah otak dari komputer karena dialah yang bertugas menghitung dan mengontrol peralatan lain disekitarnya (contoh : Zilog dengan Z80, Intel dengan Pentium™)
13. **Microcontroller** adalah sebuah Microprocessor dengan fasilitas memori didalamnya (yang dapat diprogram oleh user) untuk mengerjakan tugas tertentu yang dikehendaki oleh pemrogramnya dan terkadang juga diperlengkapi dengan port Input/Output (contoh : Zilog dengan Z8, Intel dengan 8051)
14. Arithmetic and Logic Unit (**ALU**) adalah bagian dari Microprocessor yang bertugas untuk melakukan proses aritmetika (penjumlahan dan pengurangan) dan proses logika (AND, OR, dan Shift) pada data yang melaluinya
15. **Flip-Flop** adalah rangkaian digital yg dapat digunakan untuk menyimpan suatu nilai biner ('0' atau '1'). Karena kemampuannya untuk menyimpan nilai biner, Flip-Flop menjadi komponen dasar memori (contoh : Data FF)
16. **Memori** adalah serangkaian Flip-Flop yg dikombinasikan untuk menyimpan suatu informasi.
17. **Register** adalah memori yg diimplementasikan di dalam microprocessor sehingga memiliki kecepatan yg sama dgn microprocessor.
18. Control Unit (**CU**) adalah bagian dari Microprocessor yang bertugas untuk mengontrol kerja dari bagian-bagian khusus Microprocessor di atas (ALU, Register)
19. **Peripheral** adalah adalah semua perangkat yang digunakan untuk menambah utilitas/kegunaan komputer (contoh : printer, plotter, mouse, joystick, gamepad, VR console)
20. Input/Output (**I/O**) adalah saluran transmisi yg digunakan oleh komputer untuk berinteraksi dgn peripheral (contoh : ISA (IBM Standard Architecture), EISA (Enhanced ISA), PCI, RS-232, Paralel Port)
21. Random Access Memory (**RAM**) adalah tempat penyimpanan sementara bagi data dan code (program) untuk dapat digunakan oleh Microprocessor sebagai scratch book (contoh : SDRAM).
    Data di dalam RAM akan hilang jika powernya dimatikan (volatile)
22. Read Only Memory (**ROM**) adalah tempat penyimpanan program kecil untuk menjalankan fungsi tertentu (contoh : EEPROM untuk bootstrap loader). Perbedaan dengan RAM adalah jika power supply ke ROM diputuskan isinya tidak akan hilang.
23. **Bus** adalah jalur pengiriman sinyal antar komponen.
    Dilihat dari jenis informasi yang dibawa, terdapat bus data, bus address, dan bus control.
    Dilihat dari lokasinya ada bus internal dan bus eksternal.
24. **HardDisk** adalah tempat penyimpanan sekunder untuk penyimpanan data dan program dalam jumlah/ukuran yang besar

Survival Guide using DEBUG

Debug adalah suatu program kecil yang telah ada sejak MS-DOS versi 3.0 dikeluarkan. Sampai sekarangpun program Debug tetap disertakan pada saat anda menginstall MS Windows. Program ini digunakan untuk melihat isi suatu blok memori (view), mengubahnya (edit), dan menjalankan (run) instruksi-instruksi yang ada di blok tersebut.
Cara menggunakan :

1. Di lingkungan Windows, klik 'Start' dan kemudian pilih 'Run'. Dari window 'Run' ketikkan 'Debug' dan klik tombol 'OK'.
2. Di lingkungan DOS, pindahlah ke subdirectory yang berisi instruksi-instruksi DOS. Jika Windows terinstall, pindahlah ke C:\Windows\Command, dan kemudian ketik 'Debug' dan tekan tombol 'Enter'.

Salah satu dari kedua cara tersebut akan memanggil program Debug dengan menampilkan cursor berbentuk strip (-)

```
C:\>Debug
-
```

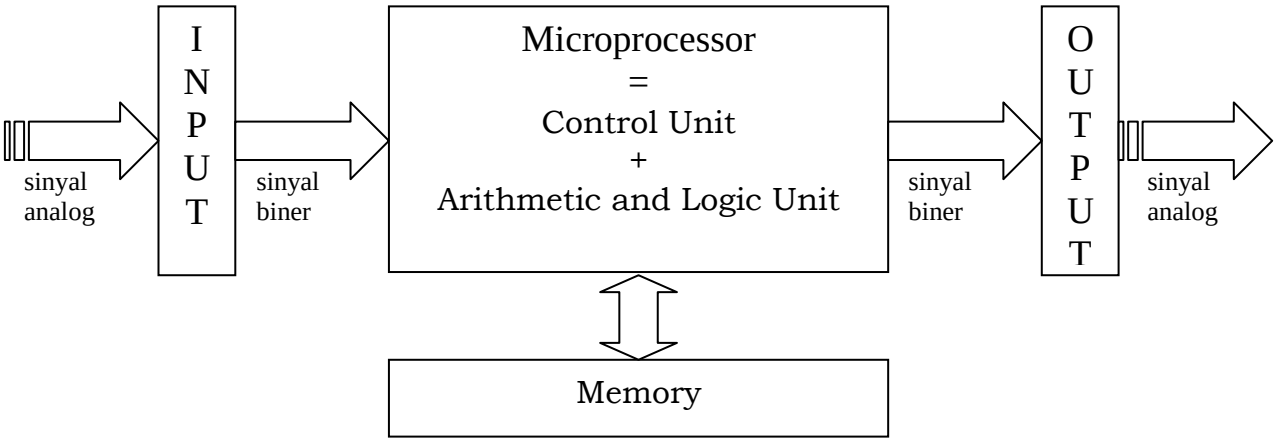
Disini Debug menanti perintah (command) dari kita.

command	arti	keterangan
a	assemble	menulis instruksi-instruksi yang akan dijalankan ke memori
u	unassemble	melihat instruksi-instruksi yg ada di memori
d	dump	melihat isi dari memori (128 bytes ditampilkan)
f	fill	mengisi secara langsung suatu blok memori
t	trace	menjalankan instruksi-instruksi yang ada di memori instruction-by-instruction (1 't' menjalankan 1 instruksi)
g	go	menjalankan semua instruksi yg ada di memori
n	name	memberikan nama file yg akan di-edit atau di-save
l	load	me-load file ke memori (yg telah didefinisikan nama-nya terlebih dahulu)
w	write	menulis isi memori ke file
r	register	menampilkan isi semua register
rxx	register xx	mengubah isi suatu register xx (contoh : rax, rip, rss)

```
C:\WINDOWS>debug
-f 0000 ffff 90   ← mengisi memori dari alamat 0000 sampai ffff dengan 90H

-a                        ← assemble program
0F6C:0100 mov ax,1234
0F6C:0103 mov ax,bx       ←  instruksi yg akan dijalankan
0F6C:0105 mov ax,[bx]
0F6C:0107

-u                        ← un-assemble program
0F6C:0100     B83412        MOV     AX,1234
0F6C:0103     89D8          MOV     AX,BX
0F6C:0105     8B07          MOV     AX,[BX]
  alamat  kode-instruksi       instruksi

-d                        ← dump (view) isi memori
0F6C:0100  B8 34 12 89 D8 8B 07 90-90 90 90 90 90 90 90 90   .4..............
0F6C:0110  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   ................
0F6C:0120  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   ................
0F6C:0130  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   ................
0F6C:0140  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   ................
0F6C:0150  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   ................
0F6C:0160  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   ................
0F6C:0170  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   ................
  alamat              isi memori (dalam Hexadecimal)        isi memori (ASCII)
-t                        ← trace program (run per instruction)
```

```
AX=1234  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0103   NV UP EI PL NZ NA PO NC
0F6C:0103 89D8          MOV     AX,BX      ← next instruction

-t                                         ← run that 'next instruction'
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0105   NV UP EI PL NZ NA PO NC
0F6C:0105 8B07          MOV     AX,[BX]                     DS:0000=20CD

-t
AX=20CD  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0107   NV UP EI PL NZ NA PO NC
0F6C:0107 8B4701        MOV     AX,[BX+01]                  DS:0001=0020

-n tes.aaa  ← set filename = 'tes.aaa'

-rcx
CX 0000
:0007       ← mengubah isi register CX dari 0000 menjadi 0007

-r
AX=20CD  BX=0000  CX=0007  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0107   NV UP EI PL NZ NA PO NC

-w          ← menulis isi memori sebanyak 7 bytes (reg. CX) ke file 'tes.aaa'
Writing 00007 bytes
-q
C:\WINDOWS>_
```


Sistem mikroprosesor bekerja pada 2 level tegangan, yaitu 0 volt dan +5 volt. Karena menggunakan 2 besaran, sistem mikroprosesor hanya dapat mengolah informasi dalam format biner (*binary*). Sehingga jika kita ingin mengolah besaran yg bukan biner, besaran tersebut harus dikonversi terlebih dahulu ke sistem bilangan biner (mata kuliah teknik digital). Pengubahan ini dilakukan oleh blok Input dan Output.



Fungsi dari masing-masing bagian :

1. Blok Input : mengubah besaran yg berlaku di luar sistem menjadi besaran biner.
2. Control Unit (CU) : mengatur operasi seluruh sistem dengan menghasilkan atau memproses sinyal kontrol
3. Arithmetic and Logic Unit (ALU) : membantu CU didalam melakukan perhitungan aritmetika (ADD, SUB) dan logika (AND, OR, XOR, SHL, SHR)
4. Memory : digunakan untuk menyimpan informasi biner
5. Blok Output : mengubah besaran biner menjadi suatu besaran tertentu

Cara Kerja

1. Sistem diluar sistem uP bekerja pada besaran analog. Blok Input berfungsi untuk mengubah besaran tersebut menjadi besaran biner yang dapat dimengerti oleh uP. Setelah blok Input mengubahnya menjadi besaran biner, maka informasi biner tersebut dikirimkan ke uP. Agar uP mengetahui bahwa blok Input akan mengirim data, terlebih dahulu blok Input mengirim sinyal kontrol ke uP.
2. CU setelah menerima sinyal kontrol tersebut akan membaca informasi yang diberikan oleh blok Input dan menyimpannya di memori.
3. Jika diperlukan, informasi yang disimpan di memori tersebut akan diambil kembali untuk diproses lebih lanjut dengan bantuan ALU.
4. Hasil proses ini dikirimkan oleh CU ke blok Output
5. Blok Output akan mengubah besaran biner menjadi suatu besaran tertentu sesuai keinginan pembuatnya.

Case Study : Alat Pengukur Rata-rata Temperature

1. Blok Input adalah termometer digital
   a) mengukur suhu diluar sistem dan mengubahnya menjadi representasi biner
   b) mempersiapkan informasi biner tersebut agar dapat diambil oleh uP
   c) mengirim sinyal kontrol 'ada data' ke uP untuk mendapatkan perhatian uP
2. CU akan membaca informasi biner tersebut dan menyimpannya di memori
3. Setelah informasi terkumpul selama 1 menit, CU harus memproses semua informasi yang telah terkumpul dan menghitung rata-ratanya
4. Hasil proses ini dikeluarkan ke blok Output untuk diubah menjadi suatu tegangan
5. Tegangan hasil blok Output akan menggerakkan jarum penunjuk untuk memberikan informasi rata-rata temperatur

Interaksi uP dan memori

Dari bagan organisasi komputer di atas, terlihat bahwa uP dapat :

1. mengambil (membaca) informasi dari memori.
2. menyimpan (menulis) informasi ke memori

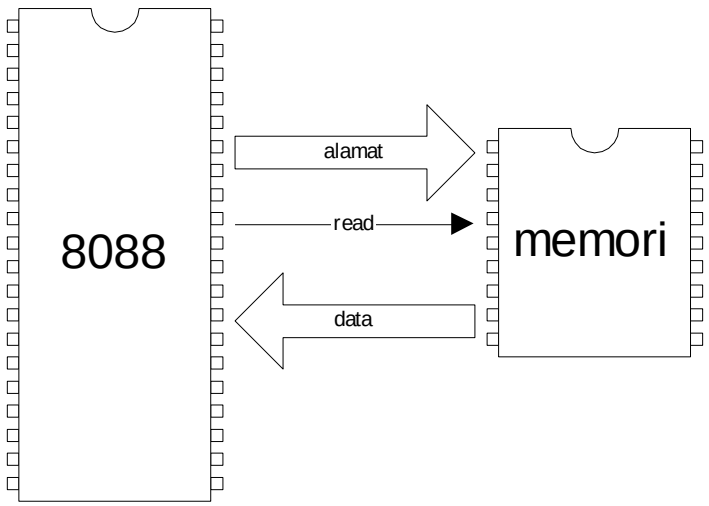
Seperti halnya manusia yg ingin menulis pada suatu buku dan membaca suatu informasi dari buku, pertama-tama manusia harus dapat menentukan dimana lokasi informasi tersebut berada (i.e. nomor halaman, alinea, baris, etc.). Baru setelah itu dapat menuliskan informasinya atau membaca informasinya.

Hal yg sama juga berlaku di uP, dimana :

1. sebelum uP dapat membaca data dari memori, pertama-tama uP harus menyediakan informasi mengenai dimana data tersebut berada.
2. sebelum uP dapat menuliskan suatu data ke memori, pertama-tama uP harus menyediakan informasi mengenai dimana data tersebut akan ditulis.

Informasi mengenai 'dimana data tersebut' disebut dengan Alamat.

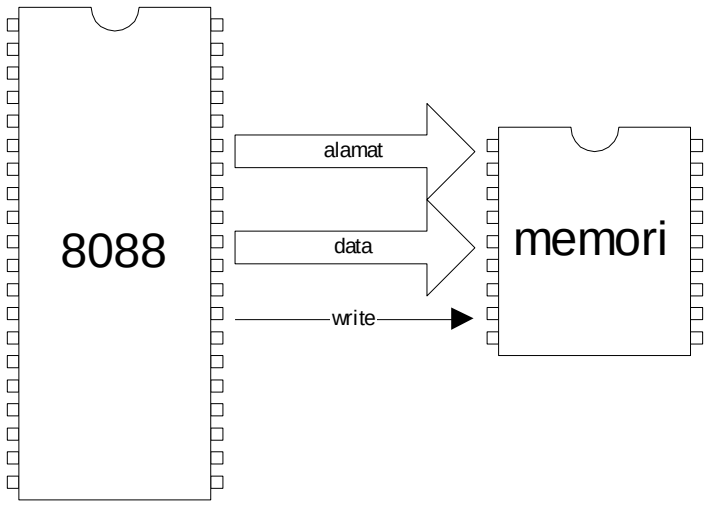
1. Proses pembacaan data oleh uP dari memori (arah data : uP ← memori)



Urutan kerja

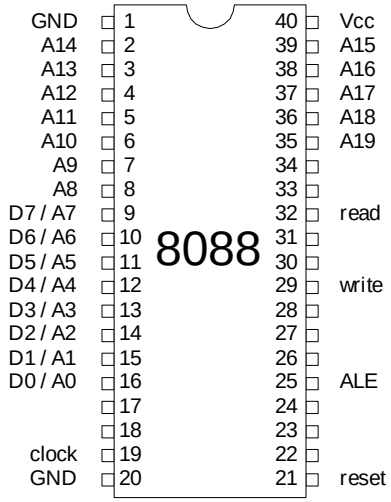
1. uP8088 mempersiapkan alamat (lokasi) dari data yg akan dibaca
2. uP8088 mengirimkan sinyal 'READ' ke memori
3. setelah menerima sinyal 'READ', memori akan mencari data yg diinginkan uP8088 sesuai dengan alamat yg diberikan
4. data yg sesuai dikirimkan ke uP8088

2. Proses penulisan data oleh uP ke memori (arah data : uP → memori)



Urutan kerja
1. uP8088 mempersiapkan alamat (lokasi) dimana data akan ditulis (diletakkan)
2. uP8088 mempersiapkan data yg akan ditulis
3. uP8088 mengirimkan sinyal 'WRITE' ke memori
4. setelah menerima sinyal 'WRITE', memori akan membaca data yg diberikan oleh uP8088 dan meletakkannya sesuai dengan alamat yg ditentukan

Pada proses penulisan dan pembacaan data, uP8088 menggunakan 3 (tiga) buah saluran khusus untuk mengirimkan (1) alamat, (2) data, dan (3) kontrol baca/tulis.
Ketiga saluran ini disebut sebagai (1) bus alamat, (2) bus data, dan (3) bus kontrol.

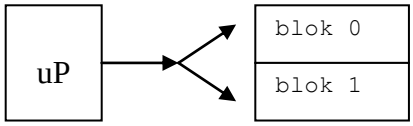


Keterangan

GND : ground (0 volt)
Vcc : +5 volt
D0 – D7: Data bus (8 informasi biner = 1 byte)
ada 2^8=256 kombinasi data → untuk kode ASCII
A0 – A19: Address bus (20 informasi biner)
ada 2^{20}=1048576 kombinasi harga
ada 1048576 lokasi memori yg dpt dialamati
read : sinyal kontrol untuk membaca
write : sinyal kontrol untuk menulis
clock : frekuensi kerja uP8088 (4.77 MHz)
reset : menginisialisasi kembali semua isi register

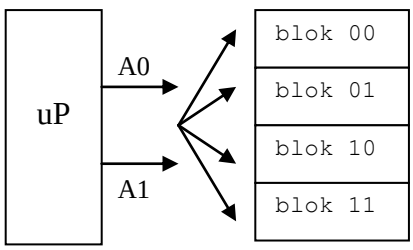
Keterangan tambahan

1. Lebar dari bus data sering digunakan untuk mengklasifikasikan uP.
   Contoh: uP 8088 adalah microprocessor 8-bit karena jumlah pin (konektor) untuk bus datanya sebanyak 8 buah (D0-D1-D2-D3-D4-D5-D6-D7 : pin no. 9 – 16).
2. Lebar dari bus alamat menyatakan jumlah maksimum blok memory yg dapat diakses oleh microprocessor.



uP dengan 1 buah alamat (A) hanya dapat mengakses 2 blok memori yaitu pada saat A=0 dan A=1

uP dengan 2 buah alamat (A0 dan A1) hanya dapat mengakses 4 blok memori yaitu pada saat :



A0=0 dan A1=0
A0=0 dan A1=1
A0=1 dan A1=0
A0=1 dan A1=1

Jumlah jalur alamat	total blok memori	kapasitas memori (1 blok memori = 1 byte)	keterangan
1	2^1=2	2 byte	
2	2^2=4	4 byte	
16	2^{16}=65536	65536 byte = 64 Kbyte	i8080, Z80, MC68020
20	2^{20}=1048576	1048576 byte = 1024 Kbyte = 1 Mbyte	i8088

Contoh: uP 8088 memiliki bus alamat dengan lebar 20 bit sehingga uP 8088 dapat mengakses memori berkapasitas maksimum 2^{20}=1048576 blok memori atau 1 Mb.

3. Sedangkan bus kontrol memiliki lebar 1 saluran untuk setiap fungsi.

Register

Pada bagan organisasi komputer, memori diletakkan terpisah dari mikroprosesor. Jika bagan tersebut diimplementasikan, uP harus mengakses memori setiap saat. Dan karena kecepatan memori jauh lebih lambat dari uP (sebagai contoh uP Pentium IV telah mencapai kecepatan 2 GHz sedangkan DDRAM maksimum hanya memiliki kecepatan 0.8 GHz), maka kecepatan kerja uP akan sangat dipengaruhi oleh kecepatan memori.
Untuk mempercepat pemrosesan data di dalam mikroprosesor, selain CU dan ALU, mikroprosesor juga akan membutuhkan memori yg memiliki kecepatan sama dengan uP. Untuk melakukannya, memori tersebut harus diimplementasikan didalam mikroprosesor. Memori ini disebut dengan register.
Jenis register berdasarkan informasi yg disimpannya :

1. **Register Data**
   digunakan untuk menyimpan data yg diperlukan untuk suatu operasi
   Terdiri dari: AX (Accumulator), BX (Base), CX (Counter), DX (Data)

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0EA0 ES=0EA0 SS=0EA0 CS=0EA0 IP=0100 NV UP EI PL NZ NA PO NC
```

2. **Register Alamat**
   karena jumlah register data sangat terbatas, maka sebagian besar data tetap diletakkan di memori. Untuk dapat mengaksesnya, uP membutuhkan alamat dari data tersebut yg disimpan oleh register alamat.
   Terdiri dari: SP (Stack Pointer), BP (Base Pointer), SI (Source Index), DI (Destination Index), DS (Data Segment), ES (Extra Segment), SS (Stack Segment), CS (Code Segment), IP (Instruction Pointer) dan BX (Base)

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0EA0 ES=0EA0 SS=0EA0 CS=0EA0 IP=0100 NV UP EI PL NZ NA PO NC
```

3. **Register Status (Flags)**
   digunakan untuk menyimpan status dari hasil operasi yg menggunakan ALU.
   Terdiri dari: OF (Overflow Flag), DF, IF, TF, SF (Sign Flag), ZF (Zero Flag), AF, PF, CF (Carry Flag)

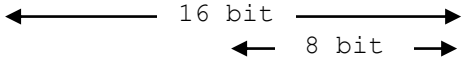
Flag :	-	-	-	-	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0EA0 ES=0EA0 SS=0EA0 CS=0EA0 IP=0100 NV UP EI PL NZ NA PO NC
```

4. **Register Instruksi**
   digunakan untuk menyimpan instruksi yang sedang dikerjakan

Catatan:

1. Register BX selain dapat digunakan sebagai register data dapat juga digunakan sebagai register alamat
2. Lebar semua register uP8088 adalah 16 bit = 2 byte. Sehingga setiap register dapat berharga 0000_H s.d. $FFFF_H$ atau 2^{16} = 65536 kombinasi harga. Setiap register akan diakses (dibaca/ditulis) dalam format 16 bit tersebut, kecuali register data dapat diakses dalam format 8 bit = 1 byte.



AH	AL
BH	BL
CH	CL
DH	DL

Instruksi Mesin

Instruksi mesin (machine instruction) adalah instruksi-instruksi yg dapat dikerjakan oleh suatu uP. Suatu program bahasa Pascal (*.PAS) tidak akan dapat dieksekusi secara langsung oleh uP. Agar komputer dapat mengerti isi program tersebut, terlebih dahulu kita harus meng-compile program tersebut agar menjadi instruksi mesin.

Dalam bahasa PASCAL	Hasil Compile		
`begin`	`0F9C:0000 9A00009E0F`	`CALL`	`0F9E:0000`
	`0F9C:0005 55`	`PUSH`	`BP`
	`0F9C:0006 89E5`	`MOV`	`BP,SP`
	`0F9C:0008 31C0`	`XOR`	`AX,AX`
	`0F9C:000A 9ACD029E0F`	`CALL`	`0F9E:02CD`
`inline($90/`	`0F9C:000F 90`	`NOP`	
`$90/`	`0F9C:0010 90`	`NOP`	
`$90);`	`0F9C:0011 90`	`NOP`	
`exit;`	`0F9C:0012 EB03`	`JMP`	`0017`
`inline($90/`	`0F9C:0014 90`	`NOP`	
`$90/`	`0F9C:0015 90`	`NOP`	
`$90);`	`0F9C:0016 90`	`NOP`	
`end.`	`0F9C:0017 5D`	`POP`	`BP`

Pada contoh diatas tampak bahwa hasil penterjemahan instruksi PASCAL "`exit`" adalah instruksi mesin "`JMP`".
Dilihat dari fungsi yg dilakukannya, instruksi mesin dapat dibedakan menjadi

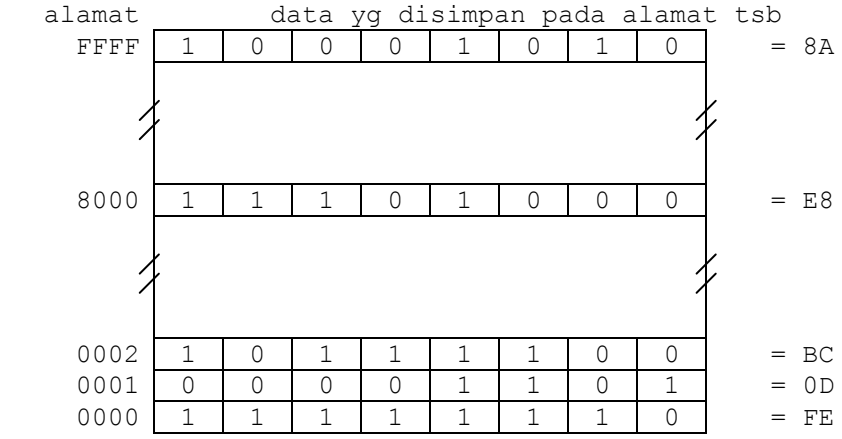
1. **Data transfer**
   digunakan untuk (1) memindahkan data dari suatu elemen memory ke elemen memory lainnya atau (2) mengisi register data dengan suatu data
   contoh: MOV, PUSH, POP
2. **Aritmetika dan Logika**
   digunakan untuk mengkalkulasi suatu perhitungan aritmetika (contoh: ADD, SUB) dan logika (AND, OR, SHL)
3. **Kontrol**
   digunakan untuk memindahkan kontrol instruksi ke suatu lokasi baru (tidak lagi secara sekuensial)
   contoh: JMP, JZ

Struktur Memori pada uP8088

Memori pada sistem uP8088 memiliki dua ciri :

1. diakses dgn alamat selebar 16 bit (00000_H s.d. $FFFFF_H$) atau 2 byte
2. data yg diakses untuk setiap alamat adalah 8 bit atau 1 byte

contoh :



Alamat dari suatu cell memori direpresentasikan dalam format 2 byte (0000_H – $FFFF_H$) yg disimpan dalam register alamat (yg lebarnya juga 2 byte). Karena kapasitas register alamat adalah 2 byte, maka jumlah cell memori yg dapat disimpan alamatnya adalah 2^{16} = 65536 cell memori.
Dan karena suatu cell memori menyimpan data 1 byte (00_H – FF_H) maka suatu register alamat uP8088 dapat mengakses (membaca/menulis) memori berkapasitas 65536 byte = 64 Kbyte.

```
  Alamat                          data (Hexa)                    data (ASCII)
0D9C:0100   0E E8 DC FC 89 46 FA 89-56 FC 0B D0 75 04 33 C0 .....F..V...u.3.
0D9C:0110   C9 C3 8B 46 F8 FF 5E FA-89 46 FE 8B 34 00 8B 0D ...F..^..F..4...
0D9C:0120   C8 08 00 00 C7 46 FE 00-00 C7 46 F8 40 00 0E E8 .....F....F.@...
0D9C:0130   AE FC 89 46 FA 89 56 FC-0B D0 75 04 33 C0 C9 C3 ...F..V...u.3...
0D9C:0140   8B 46 F8 FF 5E FA 89 46-FE 8B 46 FE C9 C3 C8 08 .F..^..F..F.....
0D9C:0150   00 00 C7 46 FE 00 00 C7-46 F8 41 00 0E E8 80 FC ...F....F.A.....
0D9C:0160   89 46 FA 89 56 FC 0B D0-75 04 33 C0 C9 C3 8B 46 .F..V...u.3....F
0D9C:0170   F8 FF 5E FA 89 46 FE 8B-46 FE C9 C3 C8 08 00 00 ..^..F..F.......
```

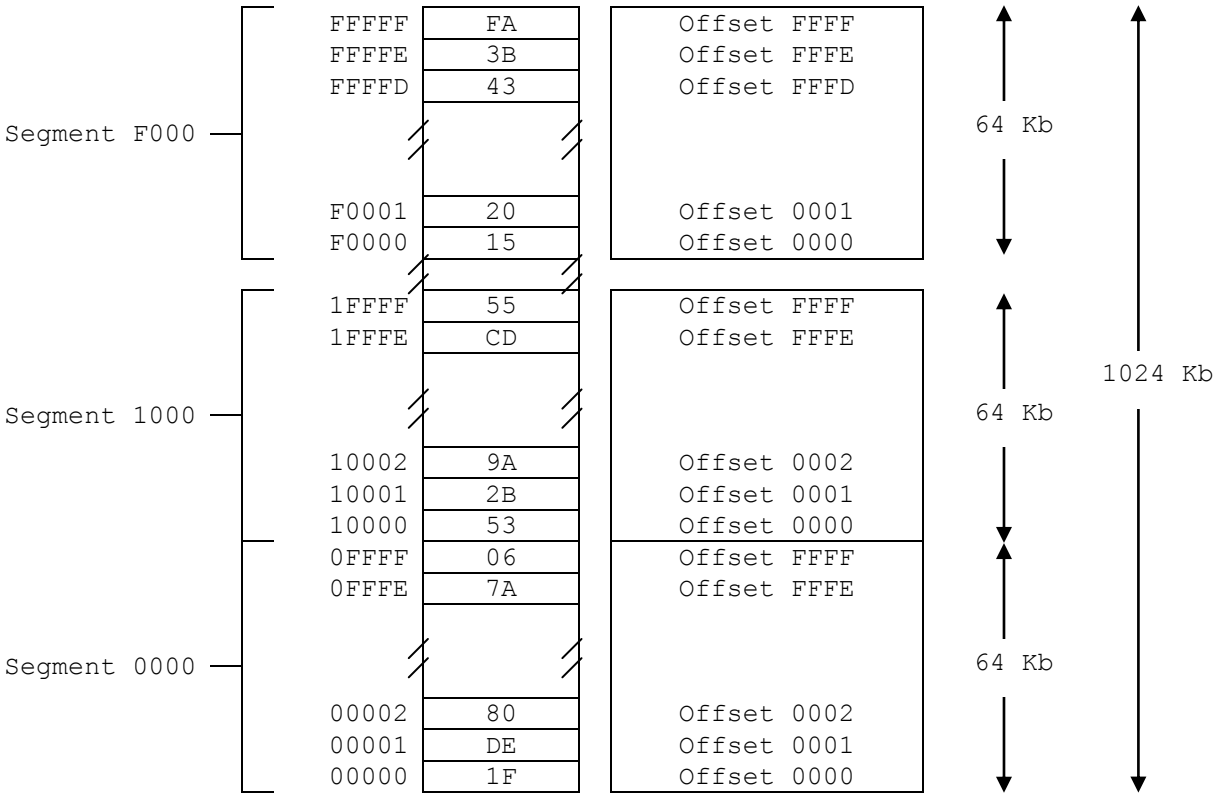
Dari hasil '*dumping*' memori dgn Debug, memori ditampilkan dalam format 128 byte.
Dapat dilihat bahwa pada alamat 0000 isi datanya 0E, dan pada alamat 0001 isi datanya E8, dan pada alamat 0002 isi datanya DC, dst.

Segmentasi

Microprocessor harus dapat mengakses semua cell memori dari alamat terendah sampai alamat tertinggi. Alamat tersebut akan disimpan didalam register alamat. Secara fisik, uP8088 memiliki 20 buah jalur alamat (A0 – A19) untuk menyediakan informasi alamat selebar 20 bit dimana informasi alamat tersebut dapat berharga 00000_H s.d $FFFFF_H$. Ke-20 bit tersebut digunakan untuk mengakses memori dgn kapasitas 2^{20} = 1048576 cell memori.
Namun masalah timbul pada lebar register alamat. uP8088 memiliki register alamat dengan lebar hanya 16 bit dari 20 yang dibutuhkan untuk mengakses semua memori.
Ini berarti jika suatu register alamat menyimpan alamat memori, dia hanya dapat digunakan untuk mengakses alamat 0000 sampai FFFF atau 64 Kb.
Untuk mengatasi kekurangan ini, Intel menggunakan 2 register untuk pengalamatan.
Satu register akan menyimpan alamat segment (suatu area memori seluas 64 Kbyte), dan
Satu register akan menyimpan alamat offset (menentukan byte yang mana di dalam segment tersebut yg akan diakses).

Contoh:

		4 bit	4 bit	4 bit	4 bit	4 bit
Segment address	DS =	0	F	8	C	
Offset address	BX =		0	0	0	5
Physical Address	DS+BX	0	F	8	C	5



uP8088 menyediakan 4 segment untuk menjalankan suatu program.

1. Segment untuk Program (Code/Instruksi) → CS:IP
   CS (Code Segment) menyimpan alamat segment (64 Kb of memory) dari program.
   IP (Instruction Pointer) menyimpan alamat offset dari program
   yang akan menentukan instruksi mana di dalam 64 Kb tadi yang akan dieksekusi
2. Segment untuk Data → DS:BX
   DS (Data Segment) menyimpan alamat segment (64 Kb of memory) dari data.
   BX (Base Register) menyimpan alamat offset dari data
   yang akan menentukan data mana di dalam 64 Kb tadi yang akan diambil
3. Segment untuk Stack → SS:SP
   SS (Stack Segment) menyimpan alamat segment (64 Kb of memory) dari stack.
   SP (Stack Pointer) menyimpan alamat offset dari top of the stack
   yang akan menentukan tumpukan (stack) mana di dalam 64 Kb tadi yang akan diambil (POP)
4. Extra Segment

Suatu program dapat memakai 4 segment yg berbeda-beda (format program *.EXE) atau hanya menggunakan sebuah segment untuk menampung program+data+stack+extra (format program *.COM)

```
C:\WINDOWS\COMMAND>debug
-n CHKDSK.EXE
-l              ← loading file 'chkdsk.exe' ke memori
-r
AX=0000 BX=0000 CX=0AA0 DX=0000 SP=4000 BP=0000 SI=0000 DI=0000
DS=0F8C ES=0F8C SS=1086 CS=0F9C IP=0000 NV UP EI PL NZ NA PO NC
```

Perhatikan bahwa program berekstensi EXE menggunakan 3 segment :

1. segment program ada di segment nomor 0F9C
2. segment stack ada di segment nomor 1086
3. segment data = segment extra = 0F8C

```
C:\WINDOWS\COMMAND>debug
-n FORMAT.COM
-l              ← loading file 'format.com' ke memori
-r
AX=0000 BX=0000 CX=28E7 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0F8C ES=0F8C SS=0F8C CS=0F8C IP=0100 NV UP EI PL NZ NA PO NC
```

Perhatikan bahwa program berekstensi COM hanya menggunakan 1 segment yaitu segment nomor 0F8C

Instruksi MOV

1. Register ← Data (contoh: MOV AX,1234 → mengisi AX dgn data 1234)
2. Register ← Register (contoh: MOV AX,BX → memindahkan isi BX ke AX)
3. Register ← Memory (contoh: MOV AX,[BX] → memindahkan isi memori ke AX, dimana alamat dari datanya ada di BX)
4. Memory ← Register (contoh: MOV [BX],AX → memindahkan isi AX ke memori, dimana datanya akan ditulis di memori pada alamat yg ada di BX))

```
Contoh instruksi MOV untuk transfer antar register dan pengisian langsung
-a
0F6C:0100 mov bx,abcd   → mengisi reg. AX dgn data ABCD_H
0F6C:0103 mov ah,56     → mengisi reg. AH dgn data 56_H
0F6C:0105 mov bl,ah     → mengcopy isi reg. AH ke reg. BL
0F6C:0107 mov ax,bx     → mengcopy isi reg. BX ke reg. AX
0F6C:0109
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0100   NV UP EI PL NZ NA PO NC
0F6C:0100 BBCDAB        MOV     BX,ABCD
-t
AX=0000  BX=ABCD  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0103   NV UP EI PL NZ NA PO NC
0F6C:0103 B456          MOV     AH,56
```

```
-t
AX=5600  BX=ABCD  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0105   NV UP EI PL NZ NA PO NC
0F6C:0105 88E3          MOV     BL,AH
-t
AX=5600  BX=AB56  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0107   NV UP EI PL NZ NA PO NC
0F6C:0107 89D8          MOV     AX,BX
-t
AX=AB56  BX=AB56  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0109   NV UP EI PL NZ NA PO NC
```

catatan:
Instruksi berikut akan menimbulkan Error :
mov ch,5678 : tidak bisa karena CH = 1 byte dan datanya 2 byte
mov dl,ax : tidak bisa karena AX = 2 byte dan DL = 1 byte
mov dx,al : tidak bisa karena AL = 1 byte dan DX = 2 byte

Contoh instruksi MOV untuk :
1. transfer Register ← Memory (membaca data dari memori)
2. transfer Memory ← Register (menulis data ke memori)

```
-a
0F6C:0100 mov bx,0002    → mengisi reg. BX dgn data 0002H
0F6C:0103 mov ah,[bx]    → membaca memori pada alamat BX sebanyak 1 byte (AH)
0F6C:0105 mov ax,[bx]    → membaca memori pada alamat BX sebanyak 2 byte (AX)
0F6C:0107 mov ax,[bx+1]  → membaca memori pada alamat BX+1 sebanyak 2 byte (AX)
0F6C:010A mov [bx],ax    → menulis isi reg. AX ke memori pada alamat BX
0F6C:010C
-d
0F6C:0000  00 01 02 03 04 05 06 07-08 09 0A 0B 0C 0D 0E 0F  ................
0F6C:0010  10 11 12 13 14 15 16 17-18 19 1A 1B 1C 1D 1E 1F  ................
0F6C:0020  20 21 22 23 24 25 26 27-28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0F6C:0030  30 31 32 33 34 35 36 37-38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0F6C:0040  40 41 42 43 44 45 46 47-48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0F6C:0050  50 51 52 53 54 55 56 57-58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0F6C:0060  60 61 62 63 64 65 66 67-68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
0F6C:0070  70 71 72 73 74 75 76 77-78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0100   NV UP EI PL NZ NA PO NC
0F6C:0100 BB0200        MOV     BX,0002
-t
AX=0000  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0103   NV UP EI PL NZ NA PO NC
0F6C:0103 8A27          MOV     AH,[BX]                         DS:0002=02
-t
AX=0200  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0105   NV UP EI PL NZ NA PO NC
0F6C:0105 8B07          MOV     AX,[BX]                         DS:0002=0302
-t
AX=0302  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0107   NV UP EI PL NZ NA PO NC
0F6C:0107 8B4701        MOV     AX,[BX+01]                      DS:0003=0403
-t
AX=0403  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=010A   NV UP EI PL NZ NA PO NC
0F6C:010A 8907          MOV     [BX],AX                         DS:0002=0302
```

```
-t
AX=00A0  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=010C   NV UP EI PL NZ NA PO NC
0F6C:010C 90            NOP
-d
0F6C:0000  00 01 03 04 04 05 06 07-08 09 0A 0B 0C 0D 0E 0F  ................
0F6C:0010  10 11 12 13 14 15 16 17-18 19 1A 1B 1C 1D 1E 1F  ................
0F6C:0020  20 21 22 23 24 25 26 27-28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
0F6C:0030  30 31 32 33 34 35 36 37-38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
0F6C:0040  40 41 42 43 44 45 46 47-48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
0F6C:0050  50 51 52 53 54 55 56 57-58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
0F6C:0060  60 61 62 63 64 65 66 67-68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
0F6C:0070  70 71 72 73 74 75 76 77-78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
```

```
catatan:
Instruksi berikut akan menimbulkan Error :
mov [bx],[bx+1]   :     transfer data dari memori ke memori secara langsung
mov [bx],12       :     transfer data langsung ke memori
```

```
kesimpulan : semua transfer yg melibatkan memori harus via register
```

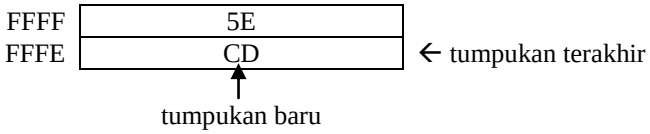
```
mov ah,[bl]       :     register alamat harus digunakan dalam format 2 byte
mov [ax],bx       :     reg. AX bukan register alamat
mov [cx],bx       :     reg. CX bukan register alamat
mov [dx],bx       :     reg. DX bukan register alamat
```

Instruksi PUSH & POP

Sebelum membahas tentang instruksi PUSH dan POP, kita harus mempelajari terlebih dahulu mengenai STACK

Stack

Karena jumlah register data terbatas (hanya 4 buah : AX, BX, CX, DX), maka diperlukan suatu lokasi penyimpanan data untuk sementara yang disebut stack. Kelebihan stack adalah dari kesederhanaannya didalam menyimpan dan mengembalikan kembali data yg telah tersimpan dibandingkan dengan penyimpanan ke memori.
Dari arti katanya stack adalah tumpukan. Ini berarti jika kita akan menyimpan data di stack, data tersebut akan ditumpuk berdasarkan urutan siapa yang terakhir datang. Oleh karenanya, sistem akses data di stack disebut LIFO (Last In First Out) dimana data yang akan diambil adalah data yang ditumpuk terakhir.
Penumpukan data di stack dilakukan dari bawah.



Untuk mengidentifikasi tumpukan data terakhir (paling bawah), digunakan register SP (Stack Pointer). Jadi SP akan ‘menunjuk’ ke tumpukan terendah dari stack. Setiap kali ada data yang ditumpuk, isi SP akan berkurang (counting down). Instruksi yang digunakan untuk operasi stack adalah PUSH dan POP.
PUSH akan ‘mendorong’ data ke stack dan POP akan ‘mengeluarkan’ data dari stack.

```
-a
0EA0:0100 mov ax,1234
0EA0:0103 mov bx,5678
0EA0:0106 push ax
0EA0:0107 push bx
0EA0:0108 pop ax
0EA0:0109 pop bx                                    SP=FFEE
-d ffe0                                               ↓
0EA0:FFE0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ← isi stack
-t
```

```
AX=1234  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=0103   NV UP EI PL NZ NA PO NC
0EA0:0103 BB7856        MOV     BX,5678
-t
AX=1234  BX=5678  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=0106   NV UP EI PL NZ NA PO NC
0EA0:0106 50            PUSH    AX
-t
AX=1234  BX=5678  CX=0000  DX=0000  SP=FFEC  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=0107   NV UP EI PL NZ NA PO NC
0EA0:0107 53            PUSH    BX
-d ffe0                                        ↓
0EA0:FFE0  00 00 00 00 00 00 00 00-00 00 00 00 34 12 00 00  ← AX ada di stack
-t
AX=1234  BX=5678  CX=0000  DX=0000  SP=FFEA  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=0108   NV UP EI PL NZ NA PO NC
0EA0:0108 58            POP     AX
-d ffe0                                  ↓
0EA0:FFE0  00 00 00 00 00 00 00 00-00 00 78 56 34 12 00 00  ← BX ada di stack
-t
AX=5678  BX=5678  CX=0000  DX=0000  SP=FFEC  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=0109   NV UP EI PL NZ NA PO NC
0EA0:0109 5B            POP     BX
-t
AX=5678  BX=1234  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=010A   NV UP EI PL NZ NA PO NC
```

```
Contoh Penggunaan Instruksi Control (JMP dan JZ)
-a
0F6C:0100 MOV     AX,ABCD
0F6C:0103 MOV     BX,DCBA
0F6C:0106 XOR     AL,AL
0F6C:0108 JMP     010C
0F6C:010A MOV     AL,BH
0F6C:010C JZ      010E
0F6C:010E
-t    → trace MOV AX,ABCD
AX=ABCD  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0103   NV UP EI PL NZ NA PO NC
-t    → trace MOV BX,DCBA
AX=ABCD  BX=DCBA  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0106   NV UP EI PL NZ NA PO NC
-t    → trace XOR AL,AL
AX=AB00  BX=DCBA  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0108   NV UP EI PL ZR NA PE NC
-t    → trace JMP 010C
AX=AB00  BX=DCBA  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=010C   NV UP EI PL ZR NA PE NC
-t    → trace JZ 010E
AX=AB00  BX=DCBA  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=010E   NV UP EI PL ZR NA PE NC
```

DATA TRANSFER

MOV – Move

1. Reg/Mem to/from Reg	1000 10dw	modregr/m		
2. Immediate to Register	1011 wreg	data	data (w=1)	
3. Memory to Accumulator	1010 000w	addr-low	addr-high	
4. Accumulator to Memory	1010 001w	addr-low	addr-high	

```
contoh 1:
MOV AL,BL   = 88D8 (10001000 11011000): d=0,w=0,mod=11,reg=011(BL),r/m=000(AL)
MOV AX,BX   = 89D8 (10001001 11011000): d=0,w=1,mod=11,reg=011(BX),r/m=000(AX)
MOV [BX],AL = 8807 (10001000 00000111): d=0,w=0,mod=00,reg=000(AL),r/m=111([BX])
MOV [BX],AX = 8907 (10001001 00000111): d=0,w=1,mod=00,reg=000(AX),r/m=111([BX])
MOV AL,[BX] = 8A07 (10001010 00000111): d=1,w=0,mod=00,reg=000(AL),r/m=111([BX])
MOV AX,[BX] = 8B07 (10001011 00000111): d=1,w=1,mod=00,reg=000(AX),r/m=111([BX])
contoh 2:
MOV AX,1234 = B83412(10111000 34H 12H): w=1,reg=000(AX),Low-data=34H,Hi-data=12H
MOV AL,78   = B078  (10110000 78H    ): w=0,reg=000(AL),data=78H
contoh 3:
MOV AX,[1234] = A13412 (10100001 34H 12H)): w=1,addr-low=34H,addr-high=12H
contoh 4:
MOV [1234],AX = A33412 (10100011 34H 12H)): w=1,addr-low=34H,addr-high=12H
```

PUSH - Push

1. Register	0101 0reg
2. Segment Register	000reg110

```
contoh 1: PUSH AX = 50   (0101 0000): reg=000(AX)
contoh 2: PUSH CS = 0E   (0000 1110): reg=01(CS)
```

POP - Pop

1. Register	0101 1reg
2. Segment Register	000reg111

```
contoh 1: POP AX  = 58   (0101 1000): reg=000(AX)
contoh 2: POP CS  = 0F   (0000 1111): reg=01(CS)
```

ARITHMETIC

ADD - Add

1. Reg/Mem with Register	0000 00dw	modregr/m		
2. Immediate to Reg/Mem	1000 00sw	mod000r/m	data	data(sw=01
3. Immediate to AX/AL	0000 010w	data	data (w=1)	

```
contoh 1:
ADD BX,CX   = 01CB  (00000001 11001011):d=0,w=1,mod=11,reg=001(CX),r/m=011(BX)
ADD BL,CL   = 00CB  (00000000 11001011):d=0,w=0,mod=11,reg=001(CL),r/m=011(BL)
contoh 2:
ADD BX,1234 = 81C33412 (10000001 11000011 34H 12H): sw=01,mod=11,r/m=011
contoh 3:
ADD AX,1234 = 053412   (00000101 34H 12H): w=1,Low-data=34H,Hi-data=12H
```

SUB - Subtract

1. Reg/Memory and Reg	0010 10dw	modregr/m		
2. Immediate from Reg/Mem	1000 00sw	mod101r/m	data	data(sw=01
3. Immediate from AX/AL	0010 110w	data	data (w=1)	

```
contoh 1:
SUB BX,CX   = 29CB  (00101001 11001011):d=0,w=1,mod=11,reg=001(CX),r/m=011(BX)
SUB BL,CL   = 28CB  (00101000 11001011):d=0,w=0,mod=11,reg=001(CL),r/m=011(BL)
contoh 2:
SUB BX,1234 = 81EB3412 (10000001 11101011 34H 12H): sw=01,mod=11,r/m=011
contoh 3:
SUB AX,1234 = 2D3412   (00101101 34H 12H): w=1,Low-data=34H,Hi-data=12H
```

LOGIC

NOT - Invert	1111 011w	mod01 0r/m
SHL = Shift Logical Left	1101 00vw	mod10 0r/m
SHR = Shift Logical Right	1101 00vw	mod10 1r/m

AND - And

1. Reg/Memory and Reg	0010 00dw	modregr/m		
2. Immediate to Reg/Mem	1000 000w	mod100r/m	data	data (w=1)
3. Immediate to AX/AL	0010 010w	data	data (w=1)	

OR - Or

1. Reg/Memory and Reg	0000 10dw	modregr/m		
2. Immediate to Reg/Mem	1000 000w	mod001r/m	data	data (w=1)
3. Immediate to AX/AL	0000 110w	data	data (w=1)	

XOR - Exclusive Or

1. Reg/Memory and Reg	0011 00dw	modregr/m		
2. Immediate to Reg/Mem	1000 000w	mod110r/m	data	data (w=1)
3. Immediate to AX/AL	0011 010w	data	data (w=1)	

CONTROL TRANSFER

JMP - Unconditional Jump

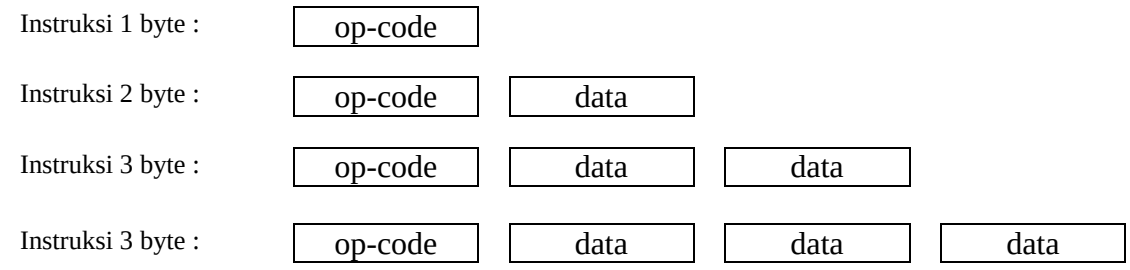
Direct w/in Segment Short	1110 1011	disp
JE/JZ - Jump on Equal/Zero	0111 0100	disp

```
contoh:
-a 100
0D9C:0100 jmp 010f
0D9C:0102 jmp 0100
0D9C:0104 jz 010f
0D9C:0106 jz 0100
0D9C:0108
-u 100
0D9C:0100 EB0D          JMP     010F
0D9C:0102 EBFC          JMP     0100
0D9C:0104 7409          JZ      010F
0D9C:0106 74F8          JZ      0100

jadi:
JMP 010F = EB0D   (11101011 00001101):jump 0D(=+13) bytes forward (IP=IP+13)
JMP 0100 = EBFC   (11101011 11111100):jump FC(=-4)  bytes forward (IP=IP-4)
JZ  010F = 7409   (01110100 00001011):jump 09(=+9)  bytes forward (IP=IP+9)
JZ  0100 = 74F8   (01110100 11111000):jump FB(=-8)  bytes forward (IP=IP-8)

(angka minus menggunakan format 2's complement)
```

```
NOTES:
d=direction:
      if d=1 then 'to' reg (Reg ← Mem)
      if d=0 then 'from' reg (Reg ← Reg, Mem ← Reg)
w=word:
      if w=1 then word operation (1 word = 2 bytes)
      if w=0 then byte operation
mod=mode:
      if mod=11 then r/m is treated as a REG field
      if mod=00 then DISP=0, disp-low and disp-high are absent
disp:
      show how far should the CPU jump from recent point (reg. IP)
r/m:
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
if s:w=01 then 16 bits of immediate data form the operand
if s:w=11 then an immediate data byte is sign extended to form the 16-bit
operand
REG is assigned according to the following table:
```

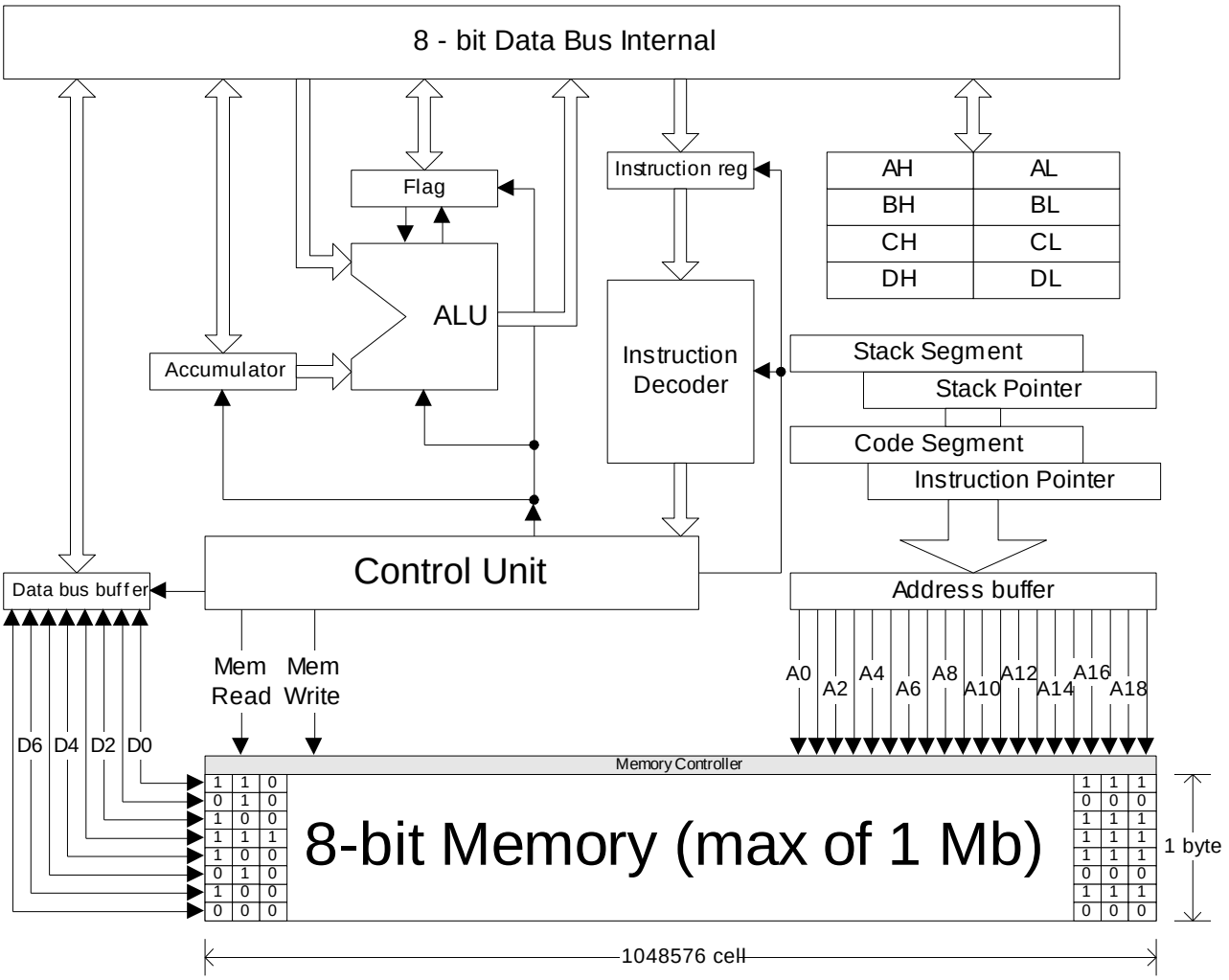
16-Bit (w=1)	8-Bit (w=0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instruksi pada uP selalu memiliki 2 bagian yaitu operation code (op-code) dan data.




```
-a 0100
0EA0:0100 mov ax,1234
0EA0:0103 mov bl,f7
0EA0:0105 push ax
-u 0100
0EA0:0100 B83412  MOV     AX,1234
0EA0:0103 B3F7    MOV     BL,F7
0EA0:0105 50      PUSH    AX
```

Yang tercetak tebal merupakan op-code nya dimana :
B8 berarti MOV AX, ?? ??
B3 berarti MOV BL, ??



Elemen didalam mikroprosesor adalah :

1. CU (**Control Unit**) adalah manajer dari semua unit. CU mengatur keselarasan kerja setiap unit. Apa yang harus dilakukan oleh suatu unit, semuanya diketahui oleh CU dengan bantuan microprogram yang ditanamkan padanya. Pengontrolan oleh CU dilakukan melalui Bus Kontrol (panah dari/ke Control Unit).
2. **Instruction Decoder** bertugas untuk menerjemahkan suatu instruksi dengan cara membandingkannya dengan tabel instruksi yang dimilikinya. Hasil dekoding diberikan ke CU, dan CU akan membangkitkan sinyal-sinyal kontrol yang diperlukan untuk melaksanakan instruksi tersebut.
3. **Register** adalah memori khusus di dalam uP. Untuk mengidentifikasikannya, register memiliki nama khusus yang mencerminkan fungsinya.
   Berdasarkan isinya, register dapat dibedakan menjadi :
   - **Register Data** memiliki lebar 16 bit namun dapat diakses dalam format 2x8 bit:
     Accumulator : AX = AH+AL
     Base Register : BX = BH+BL
     Counter Register : CX = CH+CL
     Data Register : DX = DH+DL
   - **Register Alamat** memiliki lebar 16 bit :
     Code Segment : CS, menyimpan alamat segment dari program
     Instruction Pointer : IP, menyimpan alamat offset dari program
     Data Segment : DS, menyimpan alamat segment dari data
     Index Register : BI (Base Index), SI (Source Index), DI (Destination Index),
     Pointer Register : BP (Base pointer),
     Stack Segment : SS, menyimpan alamat segment dari stack

Stack Pointer : SP, menyimpan alamat offset dari stack
Base Register : BX

- Register status (Flag) berfungsi untuk menyimpan status dari suatu operasi
- Register instruksi menyimpan instruksi yang akan dikerjakan oleh CPU (Instruction Register)

4. ALU (Arithmetic and Logic Unit) adalah mesin penghitung (kalkulator) dari CPU. CU akan menggunakan ALU jika instruksi yang dikerjakan membutuhkan perhitungan aritmetika dan logika.
   Jika suatu instruksi aritmetika dan logika dieksekusi, maka hasil operasinya dapat mengubah salah satu bit di register status.

Proses Kerja (Bagaimana uP8088 mengerjakan instruksi)

Didalam menjalankan suatu instruksi, uP 8088 melakukan 3 tahap pengerjaan sbb :

1. Penjemputan Instruksi (IF = Instruction Fetch)
   IR ← [CS+IP]
   Proses kerja dimulai dengan penjemputan instruksi baru dari memori ke IR.
   - CU menerjemahkan isi register CS dan IP untuk menentukan letak dari instruksi baru tersebut di memori.
   - Hasil terjemahan isi CS dan IP ini dikirim CU ke memori melalui bus alamat,
   - CU mengirim sinyal MemREAD untuk memberitahukan memori bahwa CU ingin membaca data
   - Memori setelah mendapat sinyal MemREAD, akan melihat isi dari bus alamat. Kemudian isi dari cell memori yang sesuai dengan alamat tersebut diletakkan di bus data (selebar 1 byte)
   - Beberapa saat setelah mengirim sinyal MemREAD, CU membaca isi dari bus data dan meletakkannya di IR.
2. Dekoding Instruksi (ID = Instruction Decode)
   Isi baru dari IR tersebut kemudian diterjemahkan oleh CU untuk mengetahui apa saja yang diinginkan oleh instruksi baru tersebut. Untuk tugas penterjemahan ini, CU menggunakan bantuan tabel instruksi yang ada di Instruction Decoder untuk dapat memahami maksud dari instruksi tersebut.
3. Eksekusi Instruksi (EX = Execution)
   Tergantung dari hasil penterjemahan instruksi diatas, CU akan melaksanakan satu dari tiga fungsi, yaitu :
   - Operasi Aritmetika atau Logika
   - Data transfer
   - Control

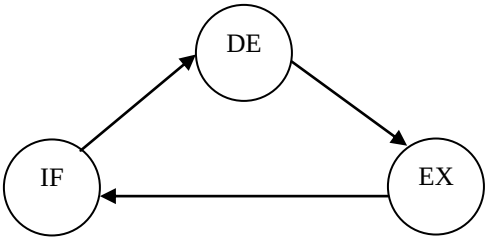
Arsitektur x86 (contoh: uP8088) menggunakan Variable Length Instruction (VLI) dimana instruksi yang berbeda memiliki panjang instruksi yang berbeda pula (bervariasi dari 1 byte sampai 4 byte).
Karena pada saat IF yg dijemput hanya 1 byte, maka kemungkinan besar setelah proses ID, CU harus menjemput beberapa byte lagi dari memori agar instruksinya menjadi lengkap.
Byte pertama dari instruksi yg dijemput disebut dgn op-code (operation code) karena dari penterjemahan op-code tersebut, didapatkan panjang isntruksi sebenarnya.
Contoh : untuk instruksi B8 34 12 (MOV AX,1234) CU pertama akan mengambil byte "B8" untuk mengetahui bahwa instruksi tsb adalah MOV AX,xx yy sehingga CU harus mengambil 2 byte setelahnya untuk dapat mengeksekusi instruksi tsb.
Kelebihan dari VLI :

- hemat space
  dimana jumlah byte yang dibutuhkan untuk merepresentasikan suatu instruksi merupakan jumlah byte minimumnya.

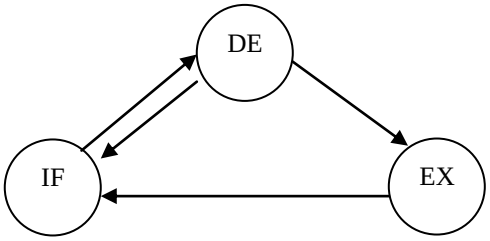
Sementara kekurangannya :

- lebih lambat karena ada suatu tenggang waktu tertentu dimana uP harus menjemput instruksi selengkapnya dari memori setelah dekoding (kekurangan ini diperbaiki melalui Prefetching)

Instruksi 1 byte (contoh: PUSH & POP)



Instruksi >1 byte (contoh: MOV, ADD, JMP)



Encoding Instruksi Mesin

Di dalam CPU (tepatnya pada blok Instruction Decoder), terdapat tabel instruksi yang memuat daftar semua instruksi yang dapat dimengerti oleh CPU tersebut. Daftar ini disebut sebagai microcode dan setiap kali CPU menerima sebuah instruksi, CPU akan memecah kode instruksi tersebut (tahap decoding) dan kemudian melihat arti dari masing-masing pecahan tersebut di microcode.
Ini menunjukkan bahwa tidak semua CPU memiliki microcode yang sama. Beda arsitektur berarti berbeda juga microcode-nya. Sebagai contoh, program yang dapat dijalankan di IBM PC tidak akan jalan di Apple Macintosh, begitu pula sebaliknya.

Hampir semua instruksi memerlukan data untuk dioperasikan. Berdasarkan Addressing Mode-nya (bagaimana uP mendapatkan data yg dibutuhkan oleh suatu instruksi), instruksi-instruksi uP 8088 dapat dibedakan menjadi :

Addressing Mode	Contoh instruksi	Arti
Immediate	Add AX, 3F 5B	AX ← AX + 3F 5B
Register Direct	Add AX, BX	AX ← AX + BX
Register Indirect	Add AX, [BX]	AX ← Mem[BX]
Displacement	Add AX, [BX+128]	AX ← Mem[BX+128]
Direct atau Absolute	Add AX, [1001]	AX ← Mem[1001]

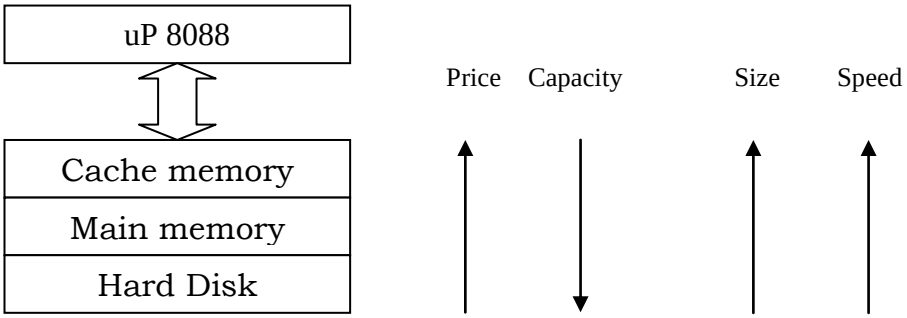
```
-a
0EA0:0100 mov bx,000f   ← Immediate
0EA0:0103 mov ax,bx     ← Register Direct
0EA0:0105 mov ax,[bx]   ← Register Indirect
0EA0:0107 mov ax,[bx-1] ← Displacement
0EA0:010A mov ax,[000d] ← Direct/Absolute
         [0000]                                                 [000D]  [000F]
-d 0000    ↓                                                      ↓       ↓
0EA0:0000  CD  20  00  A0  00  9A  EE  FE - 1D  F0  4F  03  27  08  8A  03
0EA0:0010  25  08  17  03  25  08  2A  07 - 01  01  01  00  02  FF  FF  FF
           ↑
         [0010]
-t
AX=0000  BX=000F  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=0103   NV UP EI PL NZ NA PO NC
0EA0:0103 89D8          MOV     AX,BX

-t
AX=000F  BX=000F  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=0105   NV UP EI PL NZ NA PO NC
0EA0:0105 8B07          MOV     AX,[BX]                 DS:000F=2503

-t
AX=2503  BX=000F  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=0107   NV UP EI PL NZ NA PO NC
0EA0:0107 8B47FF        MOV     AX,[BX-01]              DS:000E=038A

-t
AX=038A  BX=000F  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=010A   NV UP EI PL NZ NA PO NC
0EA0:010A A10D00        MOV     AX,[000D]               DS:000D=8A08

-t
AX=8A08  BX=000F  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0EA0  ES=0EA0  SS=0EA0  CS=0EA0  IP=010D   NV UP EI PL NZ NA PO NC
```

Memory : Klasifikasi

1. **ROM (Read Only Memory)**
   merupakan media penyimpan data non-volatile (volatile = menguap) yang berarti datanya tidak akan hilang meskipun power supplynya diputuskan.
   Contoh : ROM BIOS (ROM Basic Input Output System) pada motherboard yang bertugas untuk memeriksa keberadaan dan kondisi semua peripheral yang terpasang, menghitung dan mengecek main memory, dan bootstrap loader (memanggil OS pada Hard Disk).
2. **RAM (Random Access Memory)**
   merupakan media penyimpan data volatile yang berarti datanya akan hilang jika power supplynya diputuskan
   Contoh : SDRAM pada Main Memory komputer anda (yang mencapai 64 Mb, 128 Mb, atau 256 Mb).
   RAM dibagi menjadi 2 jenis :
   1. SRAM (Static RAM) menggunakan hanya transistor digital (± 10 buah transistor)
   2. DRAM (Dynamic RAM) menggunakan ± 2 buah transistor dan kapasitor

   SRAM : Price = Rp 100.000 untuk Capacity 0,5 Mb, Size 50 mikron per cell memory, Speed 2 ns
   DRAM : Price = Rp 100.000 untuk Capacity 64 Mb, Size 20 mikron per cell memory, Speed 10 ns

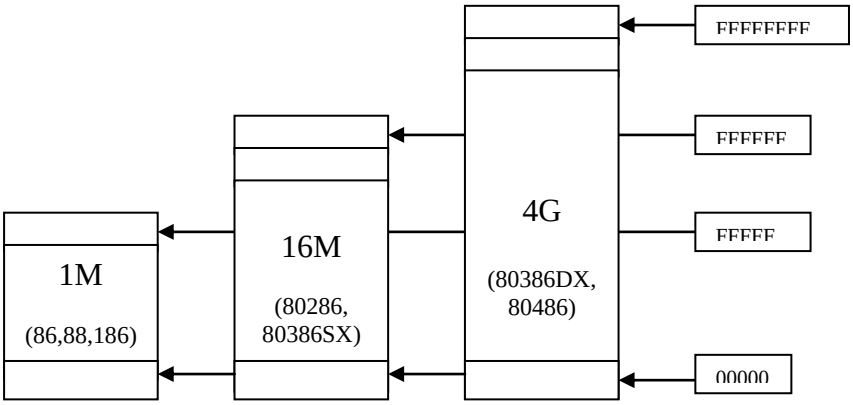
Hirarki Memory
Seorang pengguna komputer akan membutuhkan memori yang cepat dalam jumlah yang tidak terbatas. Namun hal ini akan memakan biaya yang sangat mahal. Solusi ekonomis untuk keinginan tersebut adalah dengan menggunakan hirarki memori. Dengan hirarki memori, kita dapat menyeimbangkan antara Speed, Capacity, Size, dan Price.
Tujuannya adalah untuk menyediakan sistem memori dengan harga serendah-rendahnya dan kecepatan setinggi mungkin.



Pengalamatan Memori

Metode pengalamatan memori untuk tiap-tiap desain uP amat berbeda. Disini kita akan mempelajari metode pengalamatan yang dipakai oleh uP 8088.

Logical Memory
Memori logika adalah system memori yang dilihat dari sudut programmer.
Memori logika biasanya diberikan nomor dalam format Hexadecimal.



Karena semua cell memori pada sistem uP 8088 memiliki lebar 8-bit (1 byte), maka jika uP hendak mengakses 16-bit (2 byte) data dari memori, 2 byte berturutan akan diambil. Peletakan data di memori diurutkan dari LSB (least significant byte) dengan alamat memori terendah sampai MSB (most significant byte) dengan alamat memori tertinggi. Sistem ini disebut Little Endian dan berlaku untuk semua keluarga uP x86 yang berarti alamat memori dari suatu data akan menunjuk ke LSB dari data.

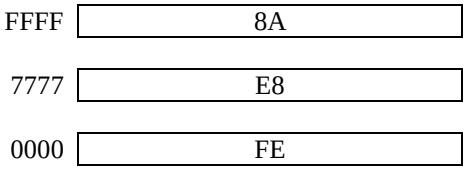
```
-a
11A8:0100 mov ax,[0005]
         [0000]              [0005] [0006]
-d 0000    ↓                   ↓     ↓
11A8:0000  CD  20  00  A0  00  9A    EE  FE - 1D  F0  4F  03  2D  0B  8A  03
      [0005] = 9A (LSB → AL)
      [0006] = EE (MSB → AH)
-t
AX=EE9A  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=11A8  ES=11A8  SS=11A8  CS=11A8  IP=0103   NV UP EI PL NZ NA PO NC
```

Physical Memory

Pada keluarga microprocessor Intel, perbedaan memori secara hardwarenya terletak pada lebarnya dimana 8088 : 8-bit, 8086-80386SX : 16-bit, dan 80386DX-80486 : 32-bit. Meskipun terdapat perbedaan lebar data pada tiap desain microprocessor di atas, seorang programmer tetap mengaksesnya seakan-akan mereka adalah 8-bit. Perbedaan lebar data tersebut hanya menjadi masalah bagi hardware desainer.

Peta memori

Peta memori adalah suatu peta yang menggambarkan lokasi dari data di memori. Peta memori digambarkan sebagai blok yang memiliki alamat dan 8-bit data dimana cell memori dengan alamat terendah digambarkan paling bawah



Untuk uP x86, sistem operasi MSDOS (Real Mode) membagi seluruh area memori menjadi 3 :

unlimited depend on the uP 100000	XMS (eXtended Memory System)	15 Mb for 80286 – 80386SX 4 Gb for 80386DX and up
FFFFF A0000	EMS (Expanded Memory System)	384 Kb System Area
9FFFF 00000	TPA (Transient and Program Area)	640 Kb Conventional Memory

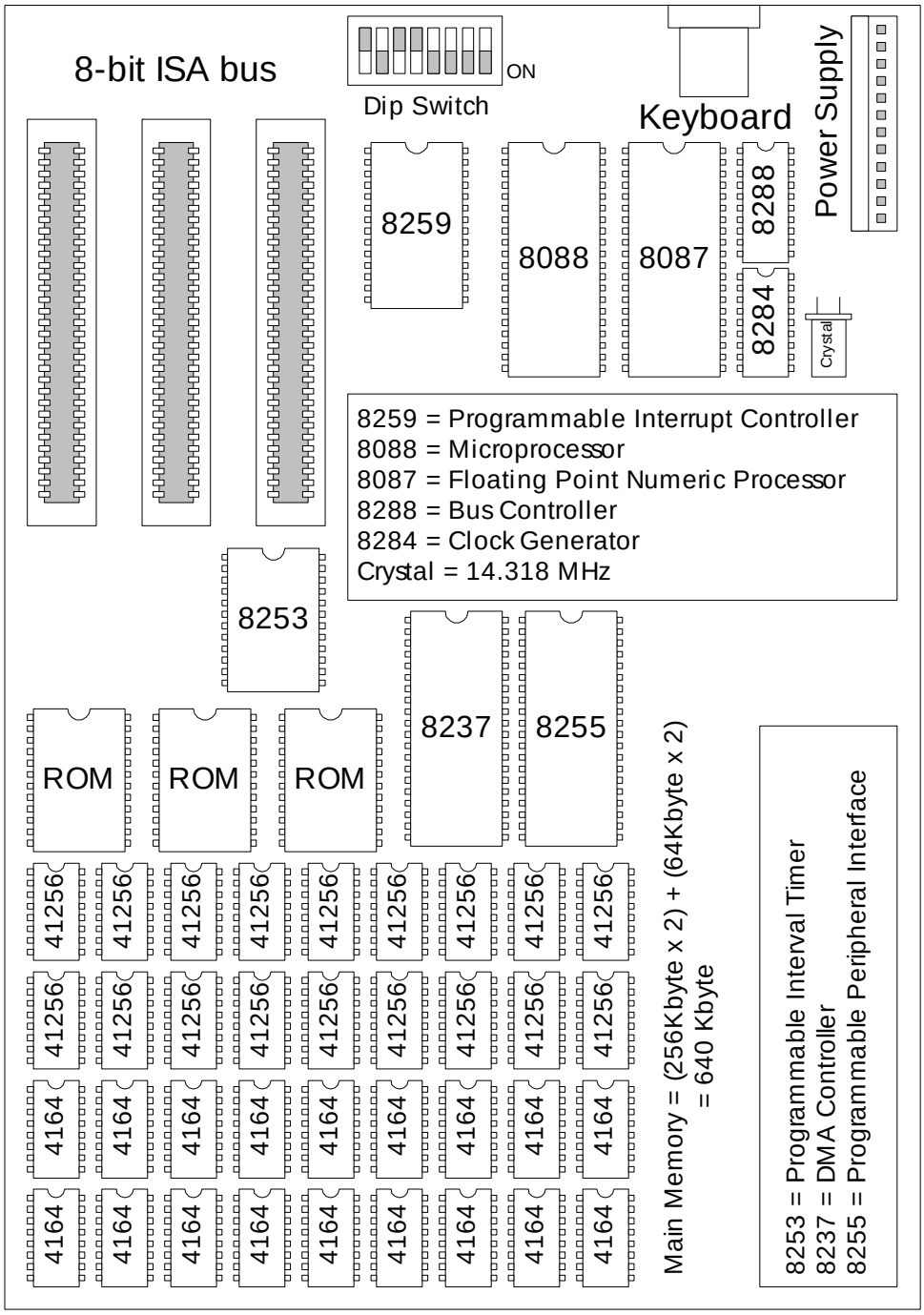
Microprocessor x86 dan sistem operasi produk Microsoft selalu mempertahankan kompatibilitasnya terhadap microprocessor dan sistem operasi pendahulunya. Oleh karenanya Intel Pentium® pun tetap mengacu pada peta memori uP i8088 dengan memori utamanya sebesar 1 Mb (Intel menyebutnya real memory).
Memori sebesar 1 Mb ini dibagi menjadi 2 menurut fungsinya.

1. Daerah 640Kb pertama disebut TPA (Transient Program Area) atau conventional memory. Disini terdapat OS (Operating System), dan program aplikasi yg dijalankan (alamat fisik 00000 – 9FFFF)
2. Diatas TPA terdapat EMS (Expanded Memory System) yang pada dasarnya merupakan BIOS (Basic I/O System) system area untuk pengontrolan I/O (alamat fisik A0000 – FFFFF)

Untuk microprocessor diatas i8088, penggunaan memori diatas 1 Mb dimungkinkan dan daerah ini disebut dengan XMS (Extended Memory System).

PC-XT (Personal Computer eXtended)

PC-XT adalah komputer komersial pertama yang dikeluarkan IBM. Dengan prosesor Intel 8088 dan RAM 640KB, komputer ini dipasarkan sekitar tahun 1985. Motherboard komputer PC-XT memiliki beberapa komponen penting :



8088 : adalah mikroprosesor yang bertugas mengontrol semua komponen
8087 : adalah arithmatic co-prosesor yang bertugas untuk membantu 8088 dalam menangani kalkulasi
8237 : adalah prosesor khusus untuk menangani DMA (Direct Memory Access)
8251 : adalah prosesor khusus untuk transaksi data serial (terpasang pada IO card, tidak pada motherboard)
8253 : adalah prosesor khusus untuk menangani *timing*
8255 : adalah prosesor khusus untuk transaksi data paralel

Mikroprosesor yg akan digunakan disini adalah i8088 yang memiliki karakteristik sebagai berikut (diambil dari spesifikasi teknis i8088 dari Intel®):

- 8-Bit Data Bus Interface ($D_0 – D_7$) → Eksternal Data Bus = 8 jalur
- 16-Bit Internal Architecture → Internal Data Bus = 16 jalur
- Direct Addressing memori sampai 1 Mbyte → Address Bus = $A_0 – A_{19}$
- 14 Register dengan lebar masing-masing 16 bit
  (4 register serba guna → AX, BX, CX, DX dapat diakses dalam 8 bit menjadi AH-AL, BH-BL, CH-CL, DH-DL)
- Operasi data dalam format Byte (8 bit), Word (16 bit), and Block (variable)

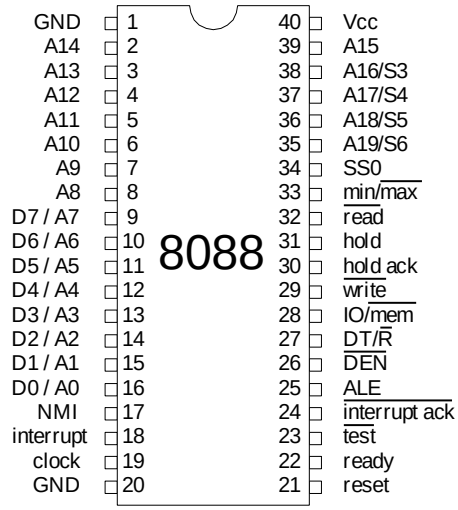
Dua frekuensi kerja :

- 5 MHz untuk type 8088
- 8 MHz untuk type 8088-2

Dua mode kerja :

- Mode Minimum
- Mode Maximum

Mikroprosesor 8088 difabrikasi dgn teknologi N-channel, depletion load, silicon gate (HMOS-II), dan dipasarkan dalam 40-pin CERDIP (Ceramic Dual In Line Package).



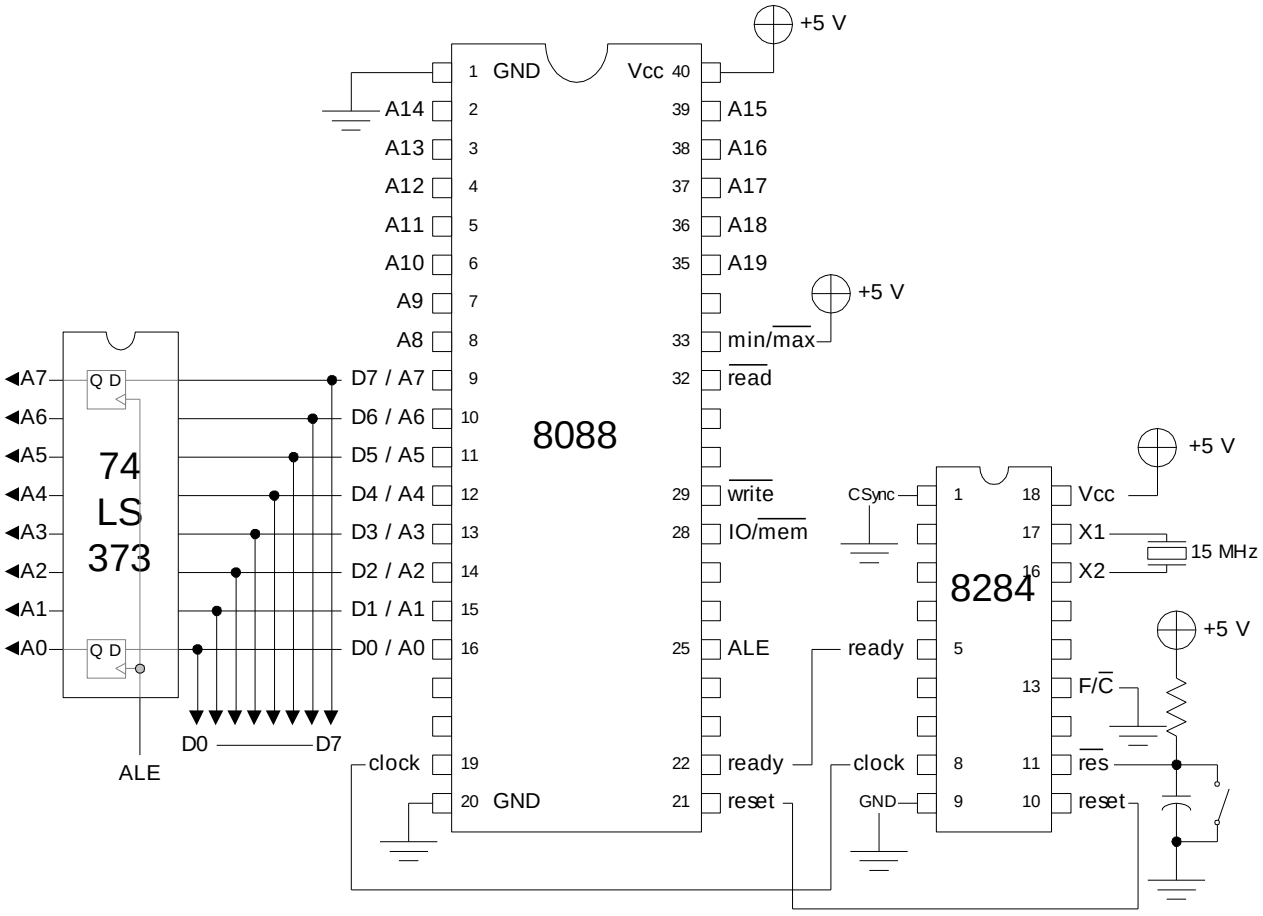
Pada i8088 ada beberapa pin yg harus diperhatikan (mode Minimum) :

Pin	Nama	**Fungsi**	Type
1,20	GND	disambungkan dengan Ground (0 Volt)	Power
40	VCC	disambungkan dengan power +5 V DC	Power
9-16	AD7 – AD0	Address line (A0 – A7) + Data line (D0 – D7) (termultipleks dalam time / TDMA)	I/O
2-8	A8 – A14	Address line (A8 – A14)	Output
35-39	A15 – A19	Address line (A15 – A19)	Output
25	ALE	Address Latch Enable (Active High) jika ‘1’ berarti pin 9 – 16 (AD0 – AD7) = Address Bus (A0 – A7) jika ‘0’ berarti pin 9 – 16 (AD0 – AD7) = Data Bus (D0 – D7)	Output
18	INTR	sinyal interupsi dari suatu I/O (Active High) yang menyebabkan i8088 melakukan service khusus	Input
24	$\overline{INTA}$	Interrupt Acknowledge (Active Low) (balasan dari i8088 ke I/O yg mengirim sinyal interupsi)	Output
19	CLK	masukan untuk frekuensi kerja (5 atau 8 MHz)	Input
21	RESET	Menginisialisasi semua register (Active High)	Input
28	$IO/\overline{M}$	Memori (Low) ataukah I/O (High) yg diakses oleh i8088 jika ‘1’ (High Voltage) berarti yg diakses adalah I/O jika ‘0’ (Low Voltage) berarti yg diakses adalah memori	Output
29	$\overline{WR}$	Sinyal Write (Active Low) berarti i8088 akan menulis data Address bus telah berisi alamat valid dari sel yg akan ditulis Data bus telah berisi data valid dari data yg akan ditulis	Output

30	HLDA	Hold Acknowledge (Active High) yg menandakan bahwa i8088 telah memutuskan dirinya dari data dan address bus	Output
31	HOLD	Sinyal hold (Active High) untuk meminta i8088 untuk memutuskan hubungannya dengan Address Bus dan Data Bus (untuk sistem DMA = Direct Memory Access)	Input
32	$\overline{RD}$	Sinyal Read (Active Low) berarti i8088 akan membaca data Address bus telah berisi alamat valid dari data yang akan dibaca	Output
33	MN/$\overline{MX}$	Menset i8088 untuk bekerja pada salah satu mode jika '1' (High Voltage) berarti bekerja dalam mode Minimum jika '0' (Low Voltage) berarti bekerja dalam mode Maximum	Input

Active High : pin tersebut dianggap aktif jika dalam kondisi High Voltage (VCC)
Active Low : pin tersebut dianggap aktif jika dalam kondisi Low Voltage (Ground)
Input : arah sinyal masuk ke dalam i8088
Output : arah sinyal keluar dari i8088
Inisialisasi register saat Reset diaktifkan: semua register akan berisi data 0000 kecuali CS=FFFF.

Sistem Penunjang i8088



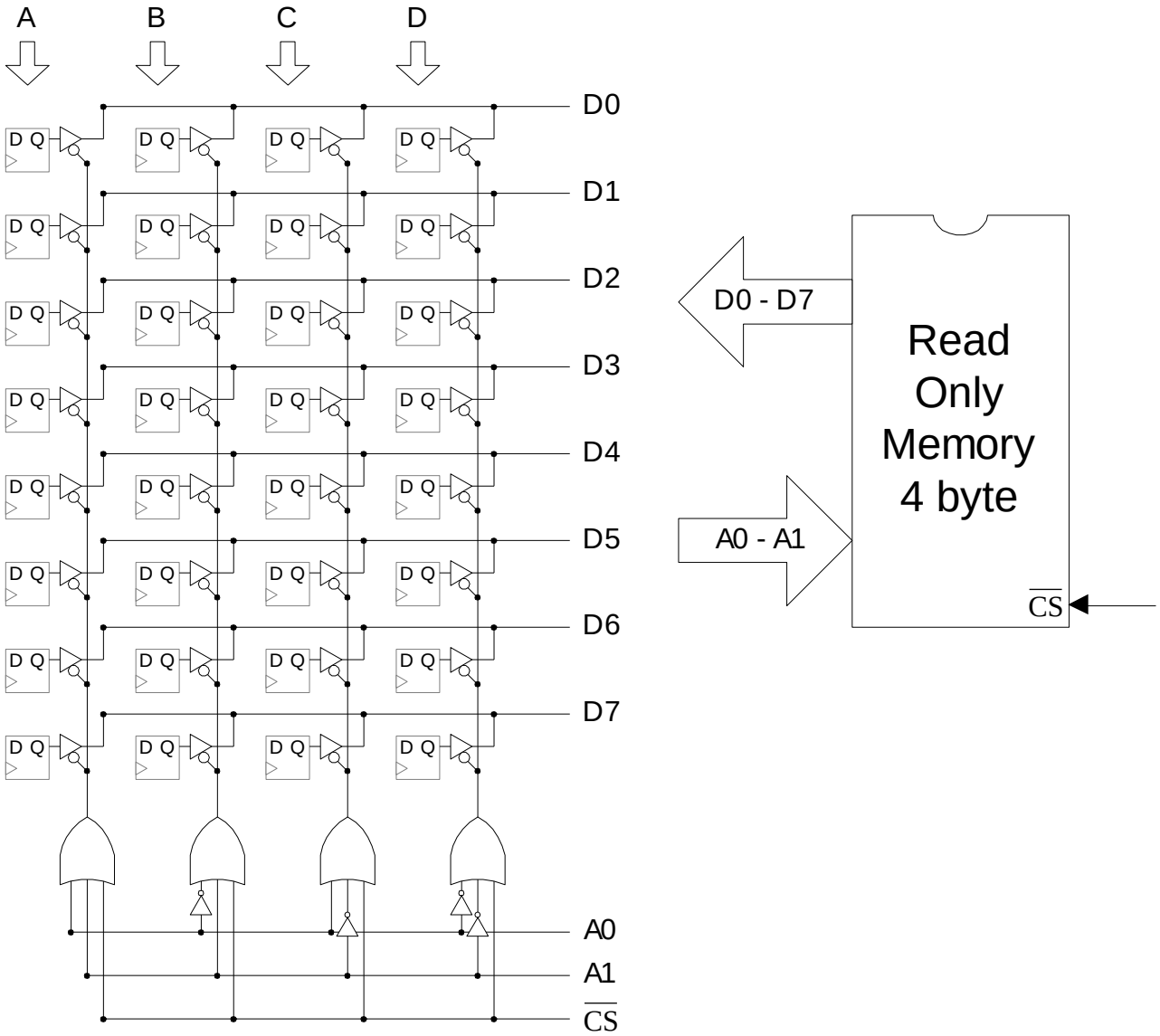
Untuk dapat bekerja, i8088 membutuhkan minimal 2 IC tambahan.

1. IC 74LS373 adalah 8 latch (Flip-Flop) yg digunakan untuk memisahkan (demultipleksing) pin 9 – 16 menjadi 2 buah informasi yaitu Address dan Data. Sinyal ALE (Address Latch Enable) digunakan untuk memisahkan kedua informasi tersebut, dimana jika ALE = High maka pin 9 – 16 membawa informasi Address, sedangkan jika ALE = Low maka pin 9 – 16 membawa informasi Data.
2. IC 8284 digunakan untuk menghasilkan sinyal CLOCK (maksimum 5 MHz), RESET, dan READY.

Peta Memori

Peta memori digunakan untuk menggambarkan lokasi semua data yang ada di memori, dimulai dari data pada alamat terendah (00000 H pada i8088) sampai alamat tertinggi (FFFFF H pada i8088).

Struktur Memori



CS	A1	A0	Set Flip-Flop yang Disambungkan dgn Data Bus (D0 – D7)
0	0	0	A
0	0	1	B
0	1	0	C
0	1	1	D
1	X	X	tidak ada karena output semua gerbang OR = '1'

Dengan memberikan suatu harga tertentu pada Address Bus (A0 – A1) maka salah satu set Flip-Flop akan diaktifkan dan datanya dapat diambil melalui Data Bus (D0 – D7) dgn catatan input CS harus diaktifkan.
Memori diatas memiliki kapasitas 4 x 8bit atau 4 byte.

Kapasitas suatu memori = 2^n x d bit

dimana n = jumlah address bus yang masuk kedalam IC memori
d = jumlah data bus yang ada pada IC memori tersebut

Klasifikasi Memori

Dilihat dari sistem aksesnya, memori dibedakan menjadi 2:

1. ROM (Read Only Memory), adalah media penyimpanan yang bersifat BACA SAJA. Karena sifatnya, maka program yang harus dilakukan oleh uP disimpan didalamnya. Program akan ditulis sekali saja ke dalam ROM karena pada saat operasionalnya program hanya dibaca saja
2. RAM (Random Access Memory), adalah media peyimpanan data yang dioperasikan sehingga sifatnya BACA dan TULIS. RAM digunakan untuk scratch book (buku oret-oretan) karena sifat data adalah variabel (berubah-ubah).

Interfacing i8088 dengan Memori

Memori harus tersedia pada suatu sistem mikroprosesor, baik untuk menyimpan program maupun untuk data. Tergantung dari kebutuhan, memori yg dapat digunakan oleh i8088 berbeda-beda berdasarkan ukurannya. Ada yg hanya ¼ Kbyte (256 Byte) sampai 128 Kbyte per kepingnya (per IC = Integrated Circuit).

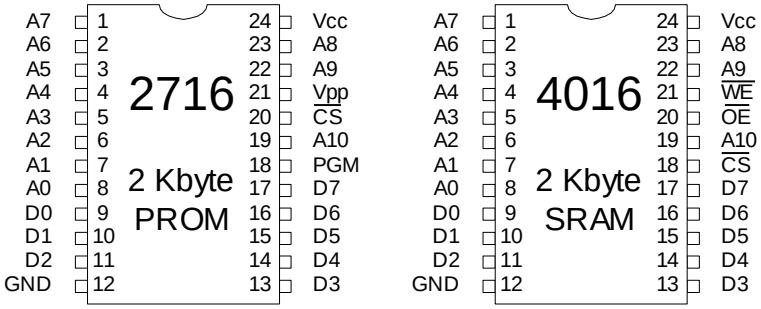
Memory Size	Memory Map	**Address line used**
256 Byte	00 – FF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7
512 Byte	000 – 1FF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
1 Kbyte	000 – 3FF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9
2 Kbyte	000 – 7FF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10}
4 Kbyte	000 – FFF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11}
8 Kbyte	0000 – 1FFF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_{12}
16 Kbyte	0000 – 3FFF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_{12} A_{13}
32 Kbyte	0000 – 7FFF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_{12} A_{13} A_{14}
64 Kbyte	0000 – FFFF	A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_{12} A_{13} A_{14} A_{15}

Data di ROM hanya dapat dibaca saja sedangkan data di RAM dapat dibaca dan juga ditulis. Hal ini menyebabkan secara hardware mereka berbeda.

Contoh :

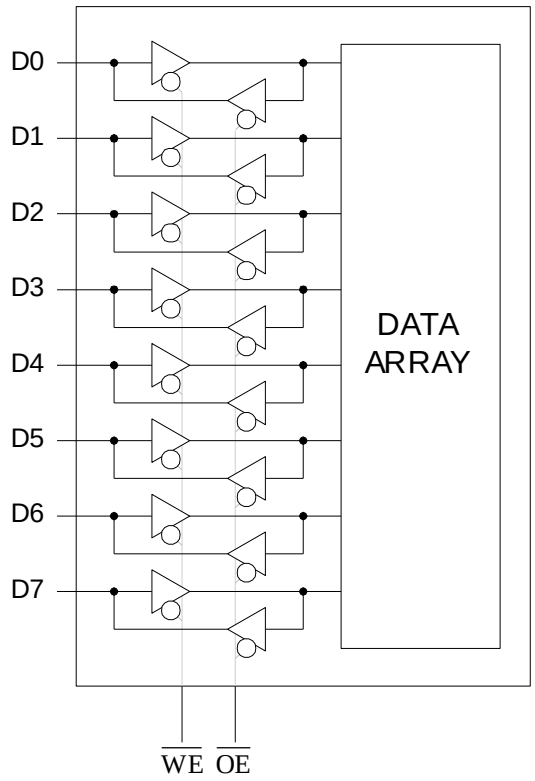
Sebuah keping memori ROM 2716 dengan kapasitas 2 Kbyte

Sebuah keping memori RAM 4016 dengan kapasitas 2 Kbyte



Nama	**Fungsi**	**Type**
$\overline{CS}$	Chip Select (Active Low) digunakan untuk mengaktifkan memori (untuk dapat diakses datanya, CS-nya harus diaktifkan terlebih dahulu)	Input
D0 – D7	Data bus 8 jalur (8 bit) sebagai interface data antara memori dng databus eksternal untuk mengambil data dari memori atau menuliskan data ke memori	Output (ROM) Input+Output (RAM)
A0 – A10	Address bus (A0 – A10) digunakan untuk mengaktifkan salah satu set data-8-bit Kapasitasnya = 2^{11} x 8 bit = 2048 byte (lihat tabel)	Input
$\overline{OE}$	Output Enable (Active Low) jika diaktifkan maka salah satu data-8-bit di dalam RAM yang sesuai dengan kondisi Address Bus dapat diakses	Input
$\overline{WE}$	Write Enable (Active Low) jika diaktifkan maka salah satu data-8-bit di dalam RAM yang sesuai dengan kondisi Address Bus dapat ditulis	Input
PGM	Program (Active +18 Volt) digunakan untuk menulis ke dalam ROM	Input

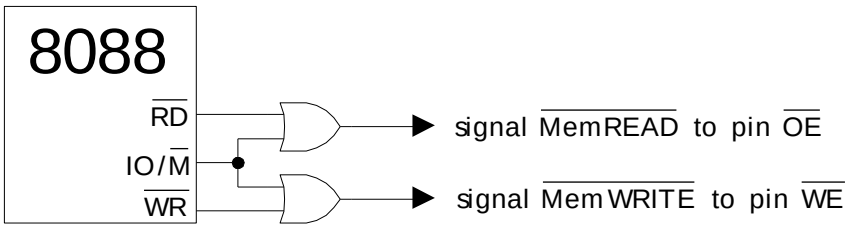
Pada kasus RAM, struktur internal pengaksesannya adalah sebagai berikut :



Sinyal WE dan OE akan mengaktifkan salah satu set dari 3-state buffer (jika WE diaktifkan → arah data masuk ke dalam memori, jika OE diaktifkan → arah data keluar dari memori).
Untuk menyambungkan suatu keping memori dengan mikroprosesor dibutuhkan 3 penyambungan yaitu sambungan untuk data, alamat, dan kontrol.

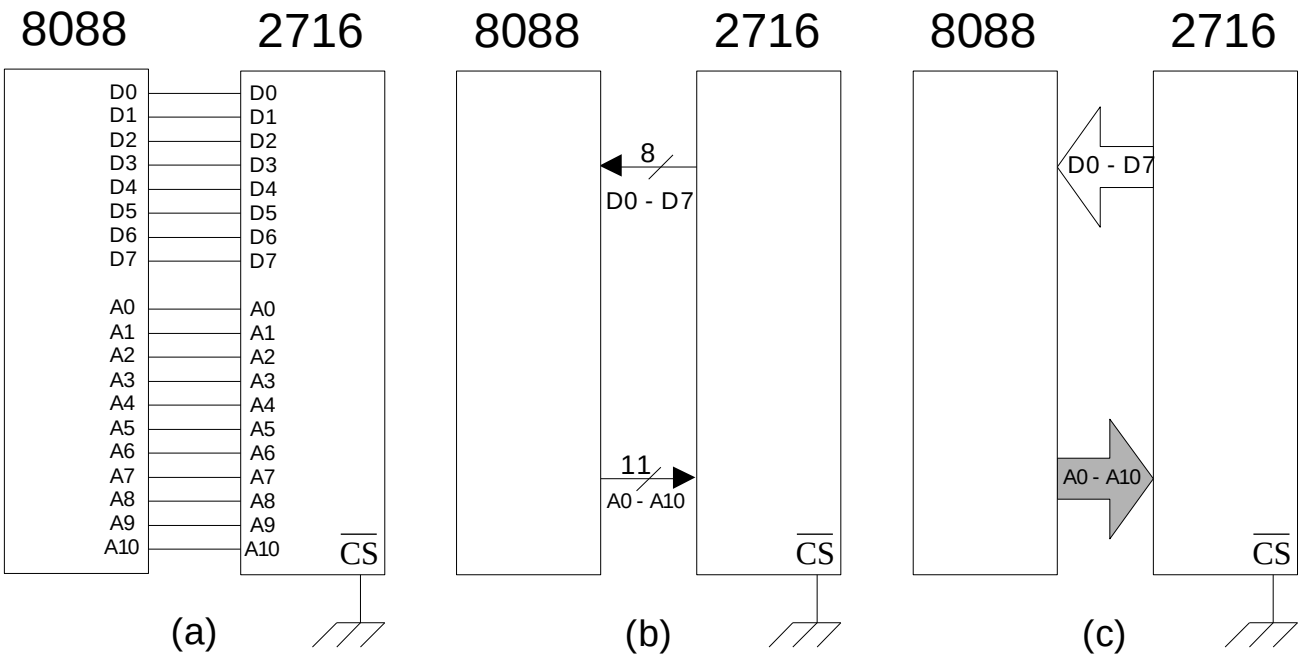
1. Sambungan data adalah langsung karena pada umumnya lebarnya sama-sama 8 bit (D0 – D7) baik dari sisi i8088 maupun dari sisi memori.
2. Sambungan alamat tergantung dari kapasitas memori yg digunakan (lihat tabel). Misalnya memori yg digunakan berkapasitas 2 Kbyte, berarti address line yg digunakan untuk mengakses suatu data pada memori tersebut adalah A0 – A10. Maka yang diambil dari i8088 adalah address line yg bersesuaian yaitu A0 – A10, dimana
   Pin A0 dari i8088 disambungkan pada masukan A0 RAM,
   Pin A1 dari i8088 disambungkan pada masukan A1 RAM,
   Pin A2 dari i8088 disambungkan pada masukan A2 RAM,
   Pin A3 dari i8088 disambungkan pada masukan A3 RAM,
   dst… sampai
   Pin A10 dari i8088 disambungkan pada masukan A10 RAM

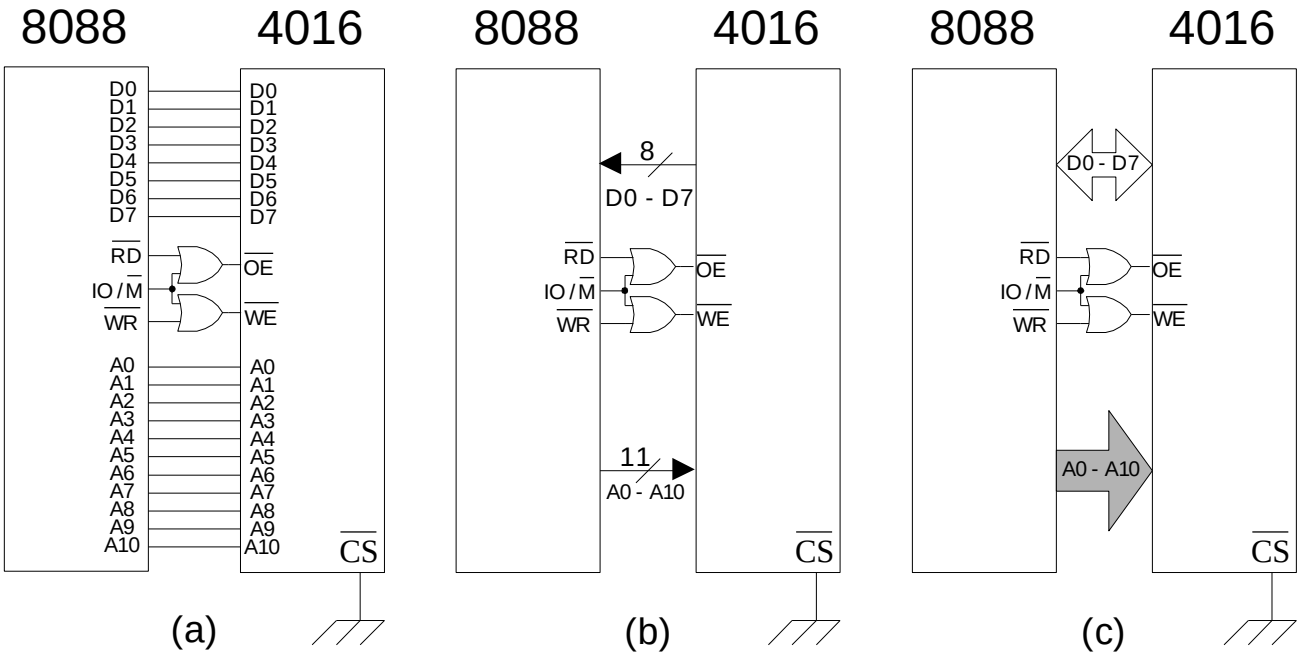
3. Sambungan kontrol yg diperlukan adalah :
   1. sinyal CS: untuk mengaktifkan suatu keping memori. Sisa dari Address line yg tidak digunakan (A12 – A19) akan dipakai untuk mengaktifkan memori tersebut (masukan Chip Select).



   2. untuk kasus RAM, sinyal Memory Read digunakan untuk memberitahu keping memori yg telah diaktifkan pin CS-nya bahwa jenis akses adalah READ. Untuk itu digunakan sebuah gerbang OR untuk menggabungkan sinyal RD dan IO/M dari i8088. Sinyal Memory READ yang active-low ini akan disambungkan pada pin OE yang juga active-low.
   3. untuk kasus RAM, sinyal Memory Write digunakan untuk memberitahu keping memori yg telah diaktifkan pin CS-nya bahwa jenis akses adalah WRITE. Untuk itu digunakan sebuah gerbang OR untuk menggabungkan sinyal WR dan IO/M dari i8088. Sinyal Memory WRITE yang active-low ini akan disambungkan pada pin WE yang juga active-low.

Contoh sambungan 8088 dengan sebuah ROM 2716 (gambar atas) dan sebuah RAM 4016 (gambar bawah)



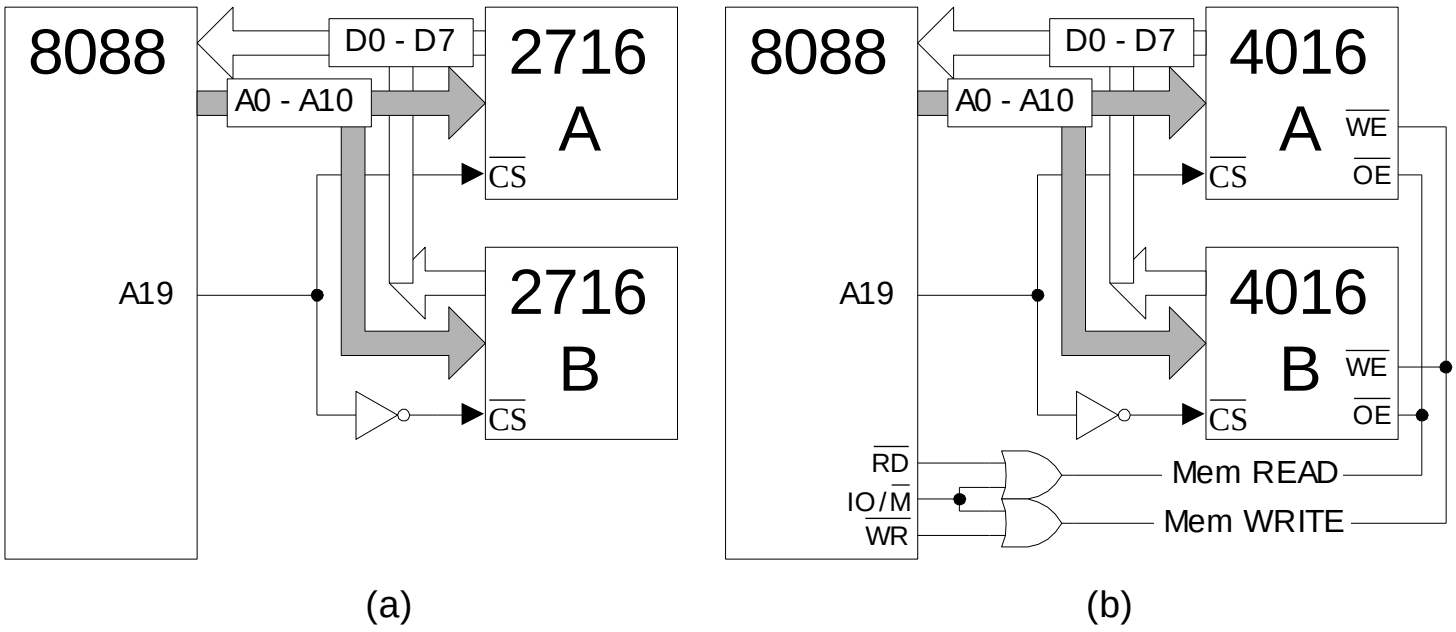


Pada gambar di atas dapat diperhatikan :

1. sambungan untuk data adalah langsung (direct connection) tanpa perantara.
   Dimana D_n dari i8088 disambungkan dengan D_n yg bersesuaian (n = 0..7)
2. sambungan untuk alamat tergantung pada Address Line yg dibutuhkan untuk mengaktifkan salah satu set flip-flop di dalam memori. Untuk kasus diatas: salah satu dari 2048 set flip-flop akan diaktifkan dgn $A_0 – A_{10}$ dari i8088
3. sambungan bus kontrol adalah untuk memberikan sinyal RD atau WR dari 8088 ke 4016 untuk menandakan jenis akses data (apakah READ → membaca data dari memori, ataukah WRITE → menulis data ke memori). Dan untuk sinyal CS, karena hanya ada satu keping memori yg digunakan, maka pengaksesan memori hanya terjadi pada 2716/4016. Sehingga 2716/4016 tersebut selalu dalam keadaan aktif.

Gambar (b) dan (c) merupakan cara lain (yg lebih sederhana) untuk menggambarkan interkoneksi (bus data dan bus alamat) antara 8088 dgn memori.

Jika digunakan dua buah 4016, akan timbul masalah ketika i8088 akan mengakses data pada alamat 00000 dimana akan ada suatu data dari 4016 yg pertama dan ada data lainnya dari 4016 yang kedua. Untuk menghindarinya, diperlukan suatu mekanisme pemilihan (selector) yang akan memilih salah satu 4016. Jadi dalam suatu saat hanya ada satu 4016 yang aktif. Untuk hal ini, disediakan sebuah masukan pada 4016 yang disebut dgn CS (Chip Select) yang aktif Low. Jika CS diaktifkan (diberikan Low Voltage) maka data didalam 4016 akan dapat diakses oleh i8088. Sebaliknya jika CS tidak diaktifkan (diberikan High Voltage) maka data didalam 4016 tidak akan dapat diakses.
Karena hanya ada 2 pilihan (2 buah 4016) maka dapat digunakan sebuah gerbang inverter dimana untuk mengaktifkannya kita gunakan A_{19}.



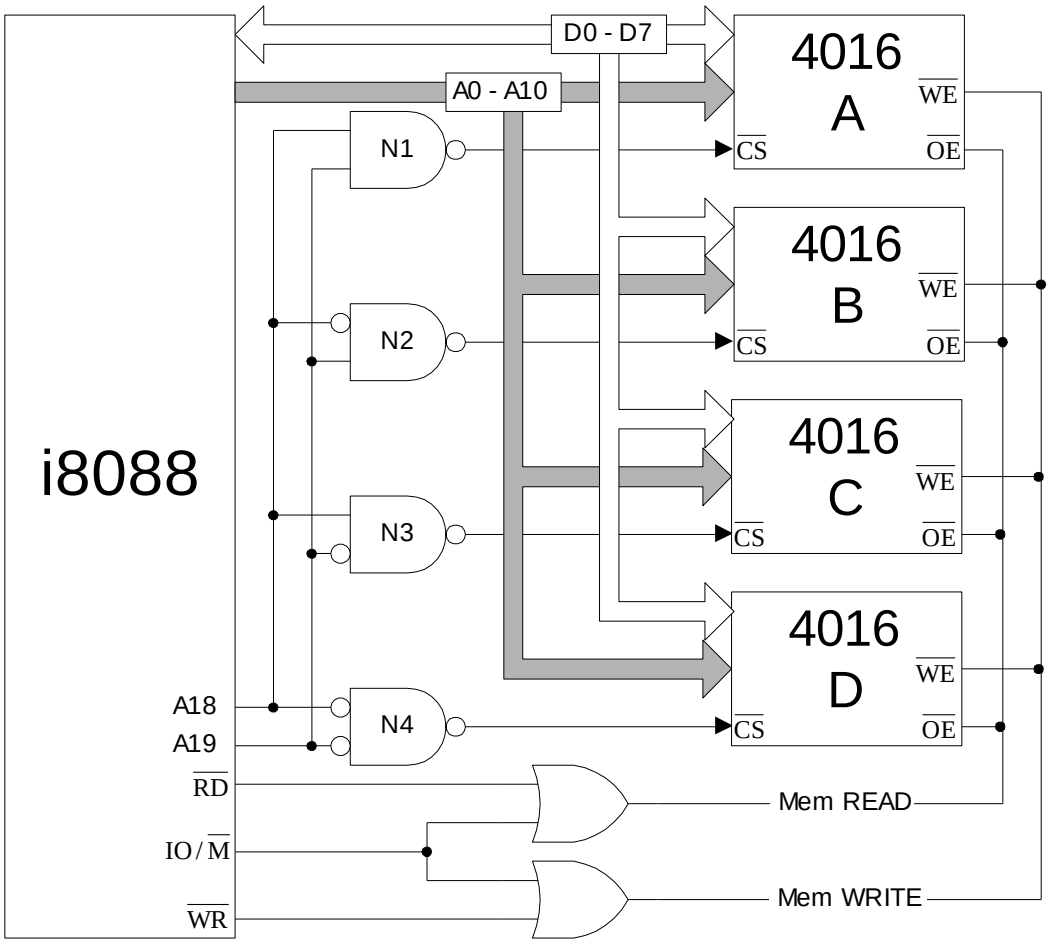
Peta memori untuk sistem diatas adalah sebagai berikut :

A19 ----------A0		
1111-1111-1111-1111-1111 = FFFFF 1000-0000-1000-0000-0000 = 80800	510 Kbyte berisi pengulangan content 4016B (atau 2716B)	second half 512 K byte selected when A19 = 1
1000-0000-0111-1111-1111 = 807FF 1000-0000-0000-0000-0000 = 80000	2 Kbyte dari 4016B (atau 2716B)	
0111-1111-1111-1111-1111 = 7FFFF 0000-0000-1000-0000-0000 = 00800	510 Kbyte berisi pengulangan content 4016A (atau 2716A)	first half 512 K byte selected when A19 = 0
0000-0000-0111-1111-1111 = 007FF 0000-0000-0000-0000-0000 = 00000	2 Kbyte dari 4016A (atau 2716A)	

Terjadinya pengulangan content memori karena tidak semua Address Line sisa (A11 – A19) digunakan untuk menghasilkan sinyal CS. Perhatikan bahwa pada saat Address Bus berisi 00000H dan 00800H, memori 4016A (atau 2716A) tetap diaktifkan. Dan pada kedua kondisi tersebut, set flip-flop yg sama (yg pertama) yg akan diakses.

Untuk menghindari terjadinya pengulangan isi memori, maka seluruh sisa Address Line yg tidak tersambung ke memori harus di-kode-kan untuk menghasilkan sinyal CS. Pada kasus diatas, untuk menghasilkan sinyal CS, maka seluruh A11 – A19 harus di-kode-kan (tidak hanya A19 saja).

Masalah akan menjadi besar jika kita akan menggunakan lebih dari dua buah 4016, dimana harus ada suatu sistem selektor untuk mengaktifkan salah satu memori. Disini kita akan mendesainnya dengan gerbang logika. Misalnya kita akan menggunakan empat buah 4016



Peta memori untuk sistem diatas adalah sebagai berikut :

1111-1111-1111-1111-1111 = FFFFF 1100-0000-1000-0000-0000 = C0800	254 Kbyte berisi pengulangan content 4016D	fourth quarter 256 Kbyte selected when A19 = 1 and A18 = 1
1100-0000-0111-1111-1111 = C07FF 1100-0000-0000-0000-0000 = C0000	2 Kbyte dari 4016 D	
1011-1111-1111-1111-1111 = BFFFF 1000-0000-1000-0000-0000 = 80800	254 Kbyte berisi pengulangan content 4016C	third quarter 256 Kbyte selected when A19 = 1 and A18 = 0
1000-0000-0111-1111-1111 = 807FF 1000-0000-0000-0000-0000 = 80000	2 Kbyte dari 4016 C	
0111-1111-1111-1111-1111 = 7FFFF 0100-0000-1000-0000-0000 = 40800	254 Kbyte berisi pengulangan content 4016B	second quarter 256 Kbyte selected when A19 = 0 and A18 = 1
0100-0000-0111-1111-1111 = 407FF 0100-0000-0000-0000-0000 = 40000	2 Kbyte dari 4016 B	
0011-1111-1111-1111-1111 = 3FFFF 0000-0000-1000-0000-0000 = 00800	254 Kbyte berisi pengulangan content 4016A	first quarter 256 Kbyte selected when A19 = 0 and A18 = 0
0000-0000-0111-1111-1111 = 007FF 0000-0000-0000-0000-0000 = 00000	2 Kbyte dari 4016 A	

Untuk menghindari terjadinya pengulangan isi memori, maka seluruh sisa Address Line yg tidak tersambung ke IC memori tersebut harus di-kode-kan untuk menghasilkan sinyal CS.
Sebagai contoh, kita akan membuat Address Decoder yg akan mengaktifkan salah satu dari 4 IC memori yg berbeda-beda kapasitasnya. Karena sambungan data dan kontrol selalu tetap, maka yg perlu diperhatikan hanya sambungan alamatnya saja (disesuaikan dengan tabel).

a. 2708 = 1 Kbyte

	untuk aktivasi CS dari 2708	untuk mengakses memori 2708
address line :	$A_{19}\ A_{18}\ A_{17}\ A_{16}\ A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}$	$A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

b. 2716 = 2 Kbyte

	untuk aktivasi CS dari 2716	untuk mengakses memori 2716
address line :	$A_{19}\ A_{18}\ A_{17}\ A_{16}\ A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}$	$A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

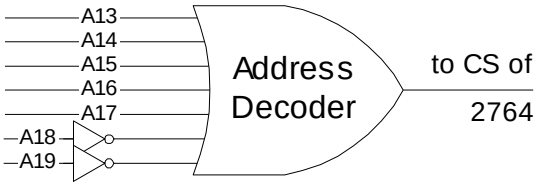
c. 2732 = 4 Kbyte

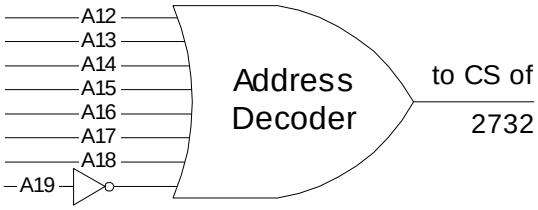
	untuk aktivasi CS dari 2732	untuk mengakses memori 2732
address line :	$A_{19}\ A_{18}\ A_{17}\ A_{16}\ A_{15}\ A_{14}\ A_{13}\ A_{12}$	$A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

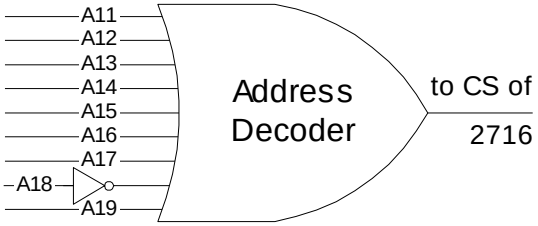
d. 2764 = 8 Kbyte

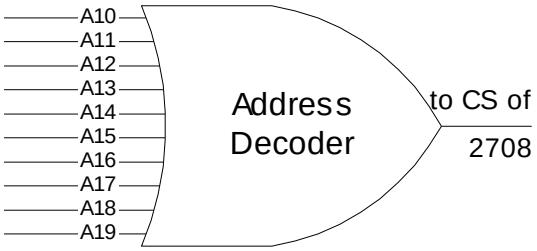
	untuk aktivasi CS dari 2764	untuk mengakses memori 2764
address line :	$A_{19}\ A_{18}\ A_{17}\ A_{16}\ A_{15}\ A_{14}\ A_{13}$	$A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

Kita akan menggunakan 4 buah gerbang OR yg berbeda untuk mengaktifkan masing-masing memori.
untuk 2708 kita membutuhkan gerbang logika OR 10 input (A10 – A19),
untuk 2716 kita membutuhkan gerbang logika OR 9 input (A11 – A19),
untuk 2732 kita membutuhkan gerbang logika OR 8 input (A12 – A19), dan
untuk 2764 kita membutuhkan gerbang logika OR 7 input (A13 – A19).





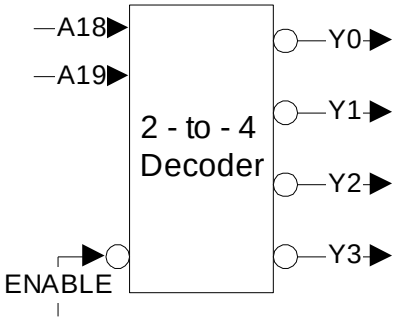




Peta memori untuk sistem diatas adalah sebagai berikut :

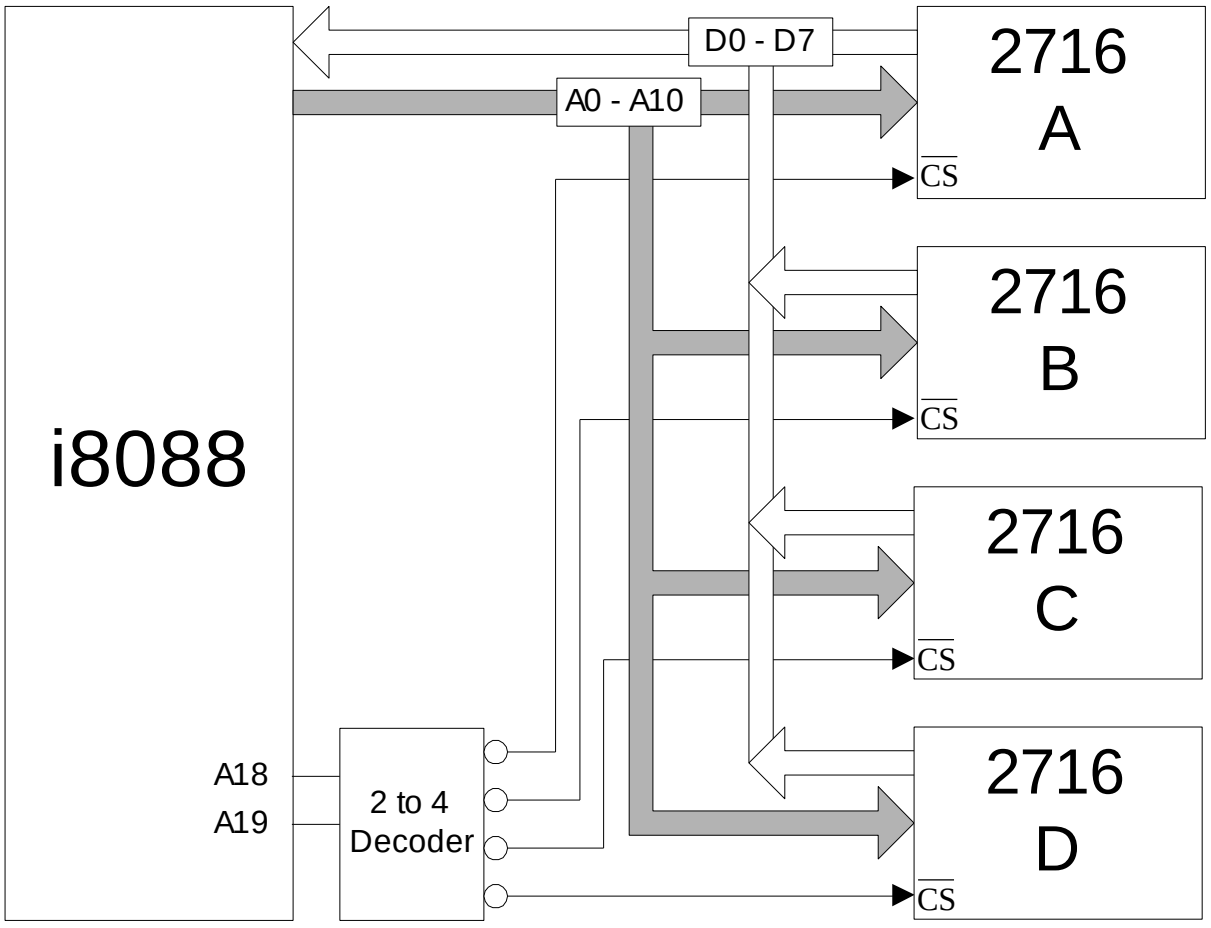
1111-1111-1111-1111-1111 = FFFFF	248 Kbyte empty
1100-0010-0000-0000-0000 = C2000	
1100-0001-1111-1111-1111 = C1FFF	8 Kbyte dari 2764
1100-0000-0000-0000-0000 = C0000	
1011-1111-1111-1111-1111 = BFFFF	252 Kbyte empty
1000-0001-0000-0000-0000 = 81000	
1000-0000-1111-1111-1111 = 80FFF	4 Kbyte dari 2732
1000-0000-0000-0000-0000 = 80000	
0111-1111-1111-1111-1111 = 7FFFF	254 Kbyte empty
0100-0000-1000-0000-0000 = 40800	
0100-0000-0111-1111-1111 = 407FF	2 Kbyte dari 2716
0100-0000-0000-0000-0000 = 40000	
0011-1111-1111-1111-1111 = 3FFFF	255 Kbyte empty
0000-0000-0100-0000-0000 = 00400	
0000-0000-0011-1111-1111 = 003FF	1 Kbyte dari 2708
0000-0000-0000-0000-0000 = 00000	

Alternatif lain (dan juga yg paling banyak diterapkan) untuk mengaktifkan salah satu keping memori adalah dengan menggunakan address decoder. Decoder adalah suatu alat yang akan menterjemahkan kondisi input dengan mengaktifkan salah satu outputnya. Setiap output dari decoder akan dihubungkan ke masukan CS dari salah satu keping memori. Karena hanya ada satu output yg aktif, maka hanya ada satu keping memory yg diaktifkan.
Pada contoh diatas (4 buah 4016), karena ada 4 buah keping yang akan diaktifkan, maka dibutuhkan decoder 2-to-4 yang memiliki 4 buah output dan salah satu output akan mengaktifkan salah satu keping memori yang bersesuaian.



ENABLE	A19	A18	Y3	Y2	Y1	Y0	2716
0	0	0	1	1	1	0	A
0	0	1	1	1	0	1	B
0	1	0	1	0	1	1	C
0	1	1	0	1	1	1	D
1	X	X	1	1	1	1	-

Mengacu pada tabel operasi diatas, kita dapat menggunakan output Y0 untuk mengaktifkan 4016-A dimana ini terjadi pada saat A19 = 0 dan A18 = 0. Dan output Y1 untuk mengaktifkan 4016-B dimana ini terjadi pada saat A19 = 0 dan A18 = 1. Dan seterusnya.



Interfacing i8088 dengan I/O (Input/Output)

Setiap perangkat Input dan Output akan memiliki nomor tersendiri (unik) untuk menandakan perangkat mana yang akan diakses oleh 8088. Nomor ini disebut nomor port.
Jika kita menggunakan instruksi MOV untuk mengakses memori, maka instruksi I/O sedikit berbeda :

1. jika menggunakan fixed addressing :
   `IN  AL,00` → mengambil data dari perangkat input port 00H dan dimasukkan ke register AL (8 bit)
   `IN  AX,FF` → mengambil data dari perangkat input port FFH dan dimasukkan ke register AX (16 bit)
   `OUT 00,AL` → mengeluarkan data dari register AL (8 bit) ke perangkat output dgn nomor port 00H
   `OUT FF,AX` → mengeluarkan data dari register AX (16 bit) ke perangkat output dgn nomor port FFH
   Keterangan :
   - Disini nomor port yg akan diakses ditulis langsung pada instruksinya.
   - Maksimum jumlah port yang bisa diakses adalah 256 (IN AL,00H sampai IN AL,FFH).
   - Untuk pengaksesannya digunakan Address Bus A0 – A7 (A0 – A7 menyimpan nomor port)
   - Tergantung dari kemampuan I/O yg diakses, Data Bus yg digunakan bisa 8 bit (reg AL) atau 16 bit (reg AX)
2. jika menggunakan variable addressing :
   IN AL,DX → mengambil data dari I/O dan dimasukkan ke register AL (nomor port ada di register DX)
   OUT AL,DX → mengeluarkan data dari register AL (8 bit) ke I/O yg nomor portnya ada di register DX
   Keterangan :
   - Disini nomor port yg akan diakses ditulis terlebih dahulu ke register DX
   - Maksimum jumlah port yang bisa diakses adalah 65536 (reg DX bisa berharga 0000 sampai FFFF).
   - Untuk pengaksesannya digunakan Address Bus A0 – A15 (A0 – A15 menyimpan nomor port)
   - Tergantung dari kemampuan I/O yg diakses, Data Bus yg digunakan bisa 8 bit (reg AL) atau 16 bit (reg AX)

Sedangkan berdasarkan peta alamatnya :

1. Isolated I/O: dimana peta alamat I/O berbeda dengan peta alamat untuk memori

Peta memori :

FFFFF	1 Kbyte dari 4016 A
FFC00	
A07FF	2 Kbyte dari 4016 B
A0000	
30FFF	4 Kbyte dari 4016 C
30000	
01FFF	8 Kbyte dari 4016 D
00000	

Peta I/O

F7	LED (Output)
00	8-bit Dip Switch (Input)

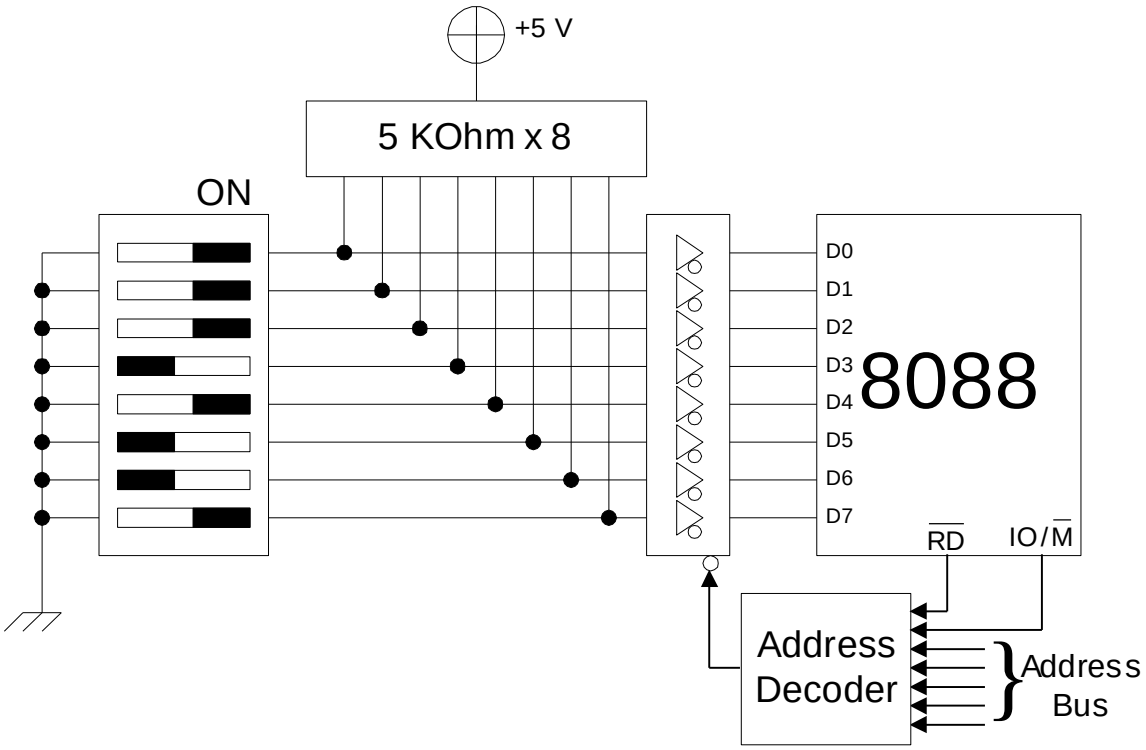
2. Memory-mapped I/O: dimana peta alamat I/O dimasukkan ke dalam peta alamat untuk memori

Peta memori + Peta IO

FFFFF	1 Kbyte dari 4016 A
FFC00	
30FFF	4 Kbyte dari 4016 C
30000	
000FF	I/O
	[00000] = Dip Switch

00000	[000FF] = LED

Pada Isolated I/O, instruksi untuk mengakses I/O dibedakan dengan instruksi untuk mengakses memori. Instruksi IN dan OUT untuk akses I/O, dan MOV untuk akses memori. Sedangkan pada Memory-mapped I/O, instruksi untuk mengakses I/O sama dengan instruksi untuk mengakses memori yaitu MOV (instruksi IN dan OUT tidak ada lagi). Untuk dapat mengambil data dari suatu I/O diperlukan suatu interface khusus untuk menjembatani data yg disimpan oleh I/O tersebut dengan data bus. Untuk keperluan ini dapat digunakan 3-state buffer untuk perangkat input (datanya akan diambil i8088) dan latch atau flip-flop untuk perangkat output (suatu data akan dikirimkan kepadanya oleh i8088)
Contoh perangkat Input dengan menggunakan Dip Switch :

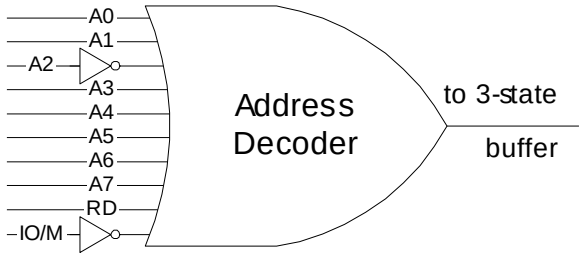


Dip switch seperti halnya switch biasa akan memutuskan atau menyambungkan suatu titik dengan salah satu titik lainnya (SPDT = Single Pole Double Terminal). Jika posisi suatu switch ada dalam posisi ON (di kanan) maka line Data tersebut akan tersambung langsung dengan Ground (0 volt) sebaliknya jika dalam posisi OFF (kiri) maka line Data tersebut akan tersambung dengan VCC (5 volt).
Posisi Dip switch di atas akan menghasilkan D0 = D1 = D2 = D4 = D7 = ground dan D3 = D5 = D6 = VCC. Sehingga data bus dari dip switch berisi (D7)01101000(D0). Agar i8088 dapat mengambil data tersebut, i8088 harus mengaktifkan kedelapan 3-state buffer diatas dengan menggunakan decoder. Setelah semua 3-state buffer diaktifkan, maka data bus i8088 akan tersambung dengan dengan data bus dip switch dan data dati dip switch dapat dibaca (RD) oleh i8088.

Desain decoder dan jumlah address line yg digunakan tergantung dari desain kita. Ada 4 desain yg berbeda :
1. fixed addressing + isolated IO
2. fixed addressing + memory mapped IO
3. variable addressing + isolated IO
4. variable addressing + memory mapped IO

Salah satu dari keempat desain tersebut akan menentukan :
1. Apakah kita menggunakan sinyal IO/M ?
   a. jika ya berarti : isolated IO
   b. jika tidak berarti : memory mapped IO
2. Jumlah address line yg dibutuhkan ?
   a. fixed addressing = 8 bit → A0 – A7
   b. variable addressing = 16 bit → A0 – A15

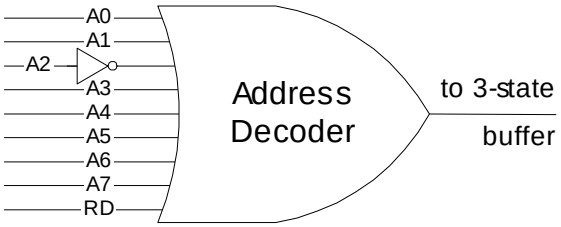
Misal Dip Switch tersebut akan diakses pada alamat 04H = (A7)0000 0100(A0) → instruksi : `IN AL,04`
Contoh untuk desain decoder nomor 1 (**fixed addressing + isolated IO**) :



Pada gerbang OR diatas, instruksi `IN AL,04` akan mengaktifkan outputnya, karena :
- instruksi IN akan mengaktifkan sinyal RD (0 volt) dan juga sinyal IO (IO/M = 5 volt)
- nomor port 04 akan memberikan nilai yg sesuai pada address line A0 sampai dengan A7

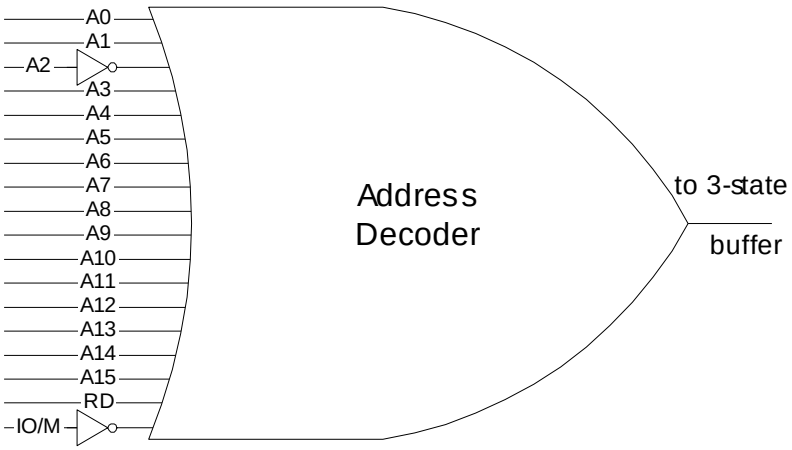
Perbedaannya dengan tiga desain lainnya hanya sedikit :

2. **fixed addressing + memory mapped IO**
   sinyal IO/M tidak perlu di-decode-kan dan instruksinya menjadi `MOV AL,[04]` → addressing mode : absolute



   Tanpa adanya dekoding sinyal IO/M, maka tidak akan ada lagi yg membedakan antara akses ke IO dgn akses ke memori. Dalam hal ini semua akses ke IO dianggap sama dgn akses ke memori. Oleh karenanya tidak ada lagi instruksi IN dan OUT yg menyebabkan sinyal IO/M berharga 5V dan akan digantikan dgn instruksi MOV yg menyebabkan sinyal IO/M berharga 0V

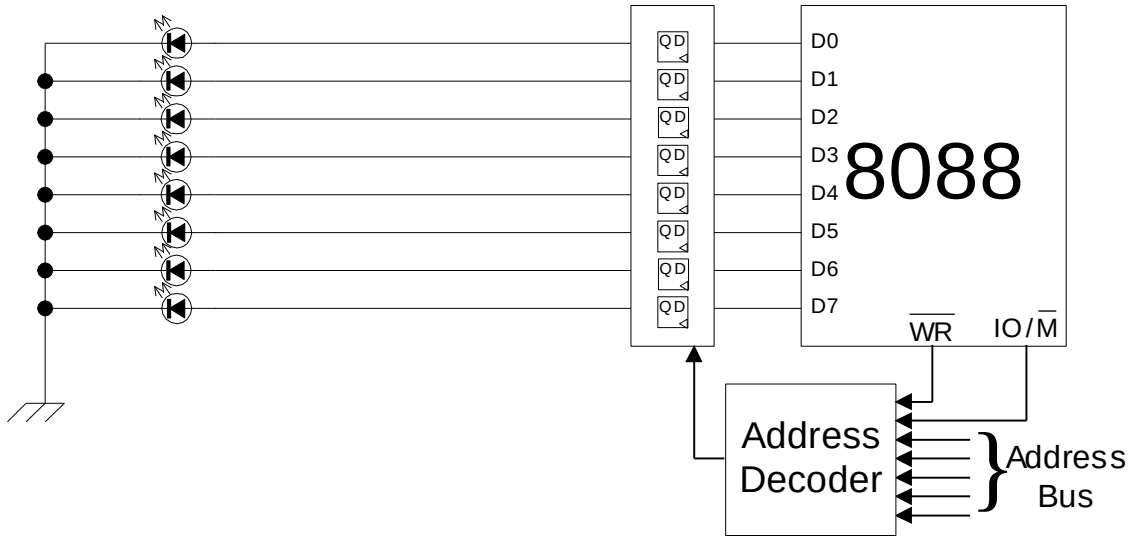
3. **variable addressing + isolated IO**
   address line yg masuk ke Address Decoder (gerbang OR diatas) adalah A0 – A15 tidak hanya sampai A7 saja.



   instruksi untuk mengaktifkannya menjadi:
```
MOV DX,0004
IN AL,DX
```

4. **variable addressing + memory mapped IO**
   address line yg masuk ke Address Decoder (gerbang OR diatas) adalah A0 – A15 tidak hanya sampai A7 saja. Dan sinyal IO/M tidak perlu di-decode-kan dan instruksinya menjadi `MOV AL,[DX]` (setelah terlebih dahulu mengisi DX dengan 0004) → addressing mode : register indirect

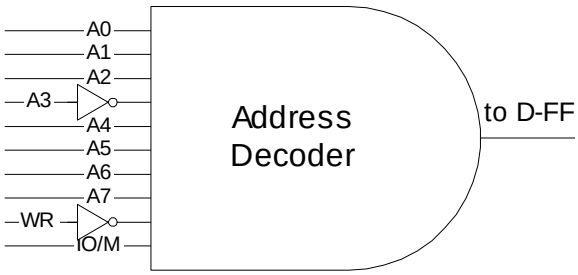
Contoh perangkat Output dengan menggunakan LED (Light Emitting Diode)



Disini akan digunakan D-FF (Data Flip-Flop) untuk menyimpan data keluaran dari i8088 agar datanya dapat dilihat terus sampai kemudian diubah dgn data lainnya. Untuk mengaktifkan Flip-Flop kita tinggal mengaktifkan sinyal CLOCK-nya agar D-FF membaca masukan data dan mengubah outputnya sesuai dgn data yg diberikan. Kemudian sesuai dengan data yg tersimpan pada D-FF, lampu LED akan dinyalakan sehingga kita dapat melihat (visualisasi) data yg dikeluarkan oleh i8088.
Misalnya data yg dikeluarkan adalah FF, maka semua LED akan menyala begitu kesemua D-FF diaktifkan clock-nya.

Sama seperti pada dip switch, ada 4 desain yg berbeda :
1. fixed addressing + isolated IO
2. fixed addressing + memory mapped IO
3. variable addressing + isolated IO
4. variable addressing + memory mapped IO

Misal LED tersebut akan diakses pada alamat F7H = (A7)1111 0111(A0) → instruksi : `OUT AL,F7`
Contoh untuk desain decoder nomor 1 (**fixed addressing + isolated IO**)



Pada gerbang AND diatas, instruksi OUT AL,F7 akan mengaktifkan outputnya, karena :
- instruksi OUT akan mengaktifkan sinyal WR (0 volt) dan juga sinyal IO (IO/M = 5 volt)
- nomor port F7 akan memberikan nilai yg sesuai pada A0 – A7

Perbedaannya dengan tiga desain lainnya hanya sedikit :
2. **fixed addressing + memory mapped IO**
   sinyal IO/M tidak perlu di-decode-kan dan instruksinya menjadi `MOV [F7], AL` → addressing mode : absolute
3. **variable addressing + isolated IO**
   address line yg masuk ke IO decoder (gerbang AND diatas) adalah A0 – A15 tidak hanya sampai A7 saja.
4. **variable addressing + memory mapped IO**
   address line yg masuk ke IO decoder (gerbang NAND diatas) adalah A0 – A15 tidak hanya sampai A7 saja.
   Dan sinyal IO/M tidak perlu di-decode-kan namun instruksinya menjadi `MOV [DX],AL` (setelah terlebih dahulu mengisi DX dengan 00F7) → addressing mode : register indirect

Interupsi adalah upaya untuk mengalihkan perhatian 8088 dari program yg sedang dikerjakan untuk memberikan pelayanan khusus terlebih dahulu pada yang menginterupsinya. Contoh : Keyboard menginterupsi kerja 8088 karena ada tuts keyboard yg ditekan. Data yg dihasilkan oleh tuts tersebut harus diambil sesegera mungkin oleh 8088.

Dilihat dari siapa yg menginterupsi :
1. Software generated : dihasilkan dengan menggunakan instruksi INT
2. Hardware generated : dihasilkan dengan mengaktifkan sinyal Interrupt pada pin 18 di 8088 (active high)

Kedua jenis interupsi di atas akan menyebabkan 8088 mengerjakan suatu routine khusus (**Interrupt Service Routine**).

Software Generated

Dihasilkan oleh instruksi INT yang diikuti nomor interupsinya. Contoh : INT 13 berarti interupsi nomor 13H.
Tipe interupsi ini memiliki 2 bagian :
1. Bagian yg dihasilkan oleh ROMBIOS (untuk nomor interupsi 0 s.d. nomor interupsi 1FH)
2. Bagian yg dihasilkan oleh sistem operasi yg digunakan (untuk nomor interupsi 20H keatas)

Adanya instruksi INT dalam program akan menyebabkan 8088 meninggalkan program yg sedang dikerjakan, dan mengerjakan routine khusus untuk nomor interupsi tersebut. Setelah routine tersebut selesai dikerjakan, maka 8088 akan kembali ke program semula yg tadinya ditinggalkan.
Urutan kerja 8088 saat mengerjakan instruksi INT XX (nilai XX dapat berharga 00H sampai FFH) :
1. Menyimpan isi register ke Stack
2. Mencari alamat routine XX
3. Lompat ke alamat routine tersebut
4. Mengerjakan routine tersebut
5. Kembali ke program semula dgn cara mengembalikan semua isi register dari Stack

Menyimpan isi register ke Stack

Langkah ini ditujukan untuk mengembalikan kembali isi register setelah routine XX selesai dijalankan.
Yang dilakukan : (1) Push Flag, (2) Clear Interrupt Flag, (3) Clear Trap Flag, (4) Push CS, (5) Push IP

Mencari alamat routine XX

Sebelum dapat menjalankan routine XX, 8088 harus mencari terlebih dahulu dimana routine XX tersebut berada.
Untuk mendapatkan alamat routine tsb, 8088 akan mencarinya di **Interrupt Vector Table** yg ada di alamat 00000H sampai 003FFH (setiap interupsi membutuhkan 4 byte : 2 byte untuk alamat Segment dan 2 byte untuk alamat Offset)

address	content	description	
003FF	02	Segment address	Interrupt #FF
003FE	46		
003FD	F0	Offset address	
003FC	00		
00003	00	Segment address	Interrupt #00
00002	C9		
00001	0F	Offset address	
00000	9E		

Lompat ke alamat routine XX

Melompat ke instruksi awal dari routine XX dgn melakukan lompatan `JMP SSSS:OOOO` dimana SSSS adalah alamat Segment dan OOOO adalah alamat Offset.

Mengerjakan routine XX

8088 akan mengerjakan semua instruksi yg ada sampai ditemukan instruksi IRET (Interrupt Return)

Kembali ke program semula

Jika instruksi IRET dikerjakan, maka semua isi regiter yg tadi disimpan, akan dikembalikan.
Yang dilakukan : (1) Pop IP, (2) Pop CS, (3) Set Trap Flag, (4) Set Interrupt Flag, (5) Pop Flag

Hardware Generated

Adanya sinyal +5V pada pin 18 pada 8088 akan menyebabkan 8088 meninggalkan program yg sedang dikerjakan, dan mengerjakan routine khusus untuk nomor interupsi tersebut. Setelah routine tersebut selesai dikerjakan, maka 8088 akan kembali ke program semula yg tadinya ditinggalkan.
Urutan kerja 8088 saat mendapatkan sinyal aktif pada pin 18 (INTR) :

1. Menyimpan isi register ke Stack
2. Mengaktifkan sinyal INTA (Interrupt Acknowledged) di pin 24
3. Membaca nomor interupsi di Address Bus (A0 – A7)
4. Mencari alamat routine untuk nomor interupsi tersebut
5. Lompat ke alamat routine tersebut
6. Mengerjakan routine tersebut
7. Kembali ke program semula dgn cara mengembalikan semua isi register dari Stack

Menyimpan isi register ke Stack

Langkah ini ditujukan untuk mengembalikan kembali isi register setelah routine selesai dijalankan.
Yang dilakukan : (1) Push Flag, (2) Clear Interrupt Flag, (3) Clear Trap Flag, (4) Push CS, (5) Push IP

Mengaktifkan sinyal INTA

Langkah ini ditujukan agar Interrupt Controller (ex:8259) memberitahukan 8088 nomor interupsinya

Membaca nomor interupsi

Membaca kondisi Address Bus A0 – A7 untuk mengetahui siapa yg menginterupsinya

Mencari alamat routine yg sesuai

Sebelum dapat menjalankan routine yg sesuai, 8088 harus mencari terlebih dahulu dimana routine tersebut berada. Untuk mendapatkan alamat routine tsb, 8088 akan mencarinya di **Interrupt Vector Table** yg sama dgn tabel untuk Software Generated Ineterrupt.

Lompat ke alamat routine tersebut

Melompat ke instruksi awal dari routine dgn melakukan lompatan `JMP SSSS:OOOO` dimana SSSS adalah alamat Segment dan OOOO adalah alamat Offset.

Mengerjakan routine tersebut

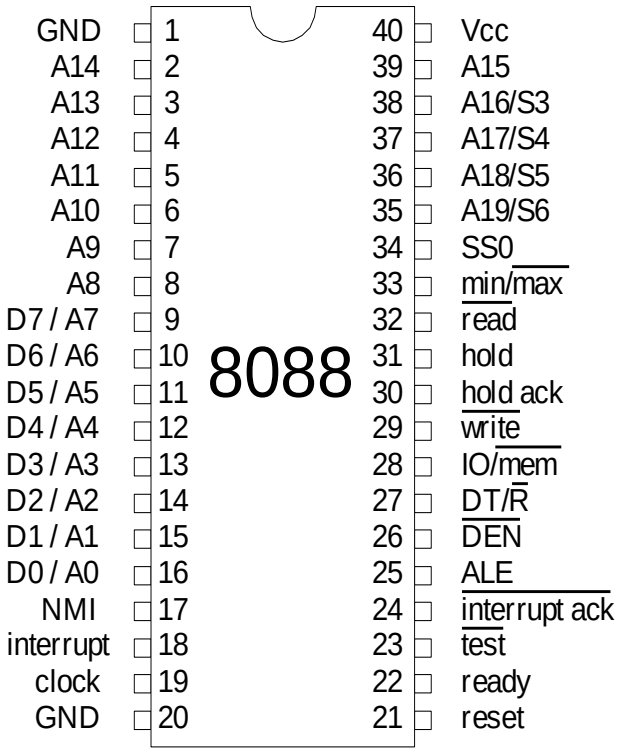
8088 akan mengerjakan semua instruksi yg ada sampai ditemukan instruksi IRET (Interrupt Return)

Kembali ke program semula

Jika instruksi IRET dikerjakan, maka semua isi regiter yg tadi disimpan, akan dikembalikan.
Yang dilakukan : (1) Pop IP, (2) Pop CS, (3) Set Trap Flag, (4) Set Interrupt Flag, (5) Pop Flag

Referensi

1. Douglas V. Hall, “Microprocessors and Interfacing : Programming and Hardware”, 2nd ed, McGraw Hill
2. Intel, “8088 Data Sheet Book”, Agustus 1990
3. Sanjiva Nath, “Assembly Language Interfacing in Turbo Pascal”, MIS Press, 1987


GND 1
A14 2
A13 3
A12 4
A11 5
A10 6
A9 7
A8 8
D7 / A7 9
D6 / A6 10
D5 / A5 11
D4 / A4 12
D3 / A3 13
D2 / A2 14
D1 / A1 15
D0 / A0 16
NMI 17
interrupt 18
clock 19
GND 20
8088
40 Vcc
39 A15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 SS0
33 min/max
32 read
31 hold
30 hold ack
29 write
28 IO/mem
27 DT/R
26 DEN
25 ALE
24 interrupt ack
23 test
22 ready
21 reset