**DIAGRAM SIKLUS**

Waktu siklus menunjukkan kecepatan suatu produksi. Dalam suatu lintasan perakitan *single station*, waktu siklus menjadi suatu *constraint* karena tidak boleh ada waktu siklus operasi maupun waktu siklus dari suatu stasiun kerja yang yang lebih besar dari pada waktu siklus yang diharapkan. Hal tersebut berbeda dengan lintasan perakitan yang memiliki stasiun kerja paralel yang menjadikan waktu siklus bukan sebagai *constraint* tetapi menjadi suatu tolak ukur untuk menentukan stasiun kerja mana yang memerlukan stasiun kerja tambahan.

Secara umum, tujuan yang ingin dicapai dalam pembentukan suatu lintasan perakitan adalah untuk mencapai efisiensi lintasan yang tinggi yang dapat dicapai dengan meminimasi waktu *delay* pada stasiun kerja yang terbentuk. Selain itu, tujuan yang ingin dicapai adalah tercapainya target produksi yang diharapkan. Tujuan tersebut dapat dicapai dengan membuat waktu siklus setiap stasiun kerja yang ada tidak melebihi waktu siklus yang sudah ditetapkan. Untuk dapat menyelesaikan masalah penyeimbangan lintasan (*line balancing*) maka harus diketahui terlebih dahulu metode kerja, mesin atau peralatan yang digunakan, serta informasi waktu yang dibutuhkan untuk setiap lintasan kerja.

Prinsip dasar dari suatu lintasan produksi adalah pergerakan atau aliran dari suatu benda kerja dari seorang pekerja kepada pekerja lainnya. Atau dengan kata lain, merupakan rangkaian dari urutan proses pengerjaan yang diperlukan untuk memproduksi suatu produk. Dengan demikian, beberapa pekerjaan yang harus dilakukan untuk menyelesaikan satu unit produk yang dibagi menjadi beberapa stasiun kerja di sepanjang lintasan produksi. Artinya, seorang pekerja melakukan pekerjaan yang sama pada setiap benda kerja yang melewatinya.

Menurut Mikell P. Groover, (1987:144), terdapat beberapa definisi atau istilah yang digunakan pada lintasan produksi, yaitu:

1. *Work element*, adalah suatu bagian dari pekerjaan keseluruhan pada proses perakitan. *Work element* ini merupakan bagian terkecil dari pekerjaan dan tidak dapat dibagi atau diuraikan lagi menjadi bagian yang lebih kecil. Contohnya seperti proses *drilling* untuk membuat lubang pada suatu material.
2. *Assembly product,* adalah suatu produk yang melewati urutan stasiun kerja, dimana produk dibuat hingga menjadi produk jadi pada stasiun kerja yang terakhir.
3. *Work station*, adalah suatu lokasi pada lintasan perakitan yang terdiri dari elemen-elemen pekerjaan untuk mengerjakan suatu produk.
4. *Cycle time*, adalah waktu penyelesaian antara dua perakitan yang berurutan.
5. *Station time*, adalah sejumlah waktu dari elemen-elemen pekerjaan pada stasiun kerja yang sama.
6. *Delay time of station*, adalah perbedaan antara waktu siklus dengan jumlah waktu pada suatu stasiun kerja.
7. *Precedence diagram*, adalah suatu diagram yang menggambarkan elemen-elemen pekerjaan yang dikehendaki untuk terbentuk.

**FORMAT INSTRUKSI**

Sebuah format instruksi mendefinisikan susunan bit-bit dari sebuah instruksi.sebuah format instruksi harus menyertakan sebuah opcode dan, baik secara implisit ataupun eksplisit, 0 atau lebih operan. setiap operan eksplisit direferensikan menggunakan jenis pengalamatan yang telah dibahas sebelumnya. Format yang digunakan harus, baik secara implisit atau eksplisit, mengindikasikanjenis pengalamatan yang digunakan. Sebagian besat set instruksi memiliki lebih dari satu format instruksi.

1. Panjang instruksi

Salah satu isu terbesar yang harus dihadapi adalah panjang format instruksi.Keputusan ini berpengaruh dan dipengaruhi oleh, organisasi memori, struktur bus, kompleksitas processor dan kecepatan processor.Keputusan ini juga menentukan kekayaan dan fleksibilitas sebuah mesin dalam pandangan seorang programmer bahasa assembly.

Pertimbangan dasar panjang instruksi salah satunya terletak pada pilihan apakah memiliki instruksi yang powerful atau memilih untuk menghemat ruang memori.Pertimbangan lainya, adalah pilihan apakah panjang sebuah instruksi harus sama dengan lebar jalur komunikasi memori (data bus) atau salah satunya (panjang instruksi atau lebar jalur) harus merupakan kelipatan dari yang lainya (contoh lebar jalur = 2 x panjang instruksi). feature penting lain yang perlu di perhatikan adalah panjang instruksi harus merupakan kelipatan panjang karakter (word). Dengan demikian, tidak ada bit yang tidak digunakan.

b. Alokasi Bit

Isu penting berikutnya setelah panjang bit adalah bagaimana kita mengalokasikan bit-bit dalam sebuah format. Jumlah opcode yang lebih berarti jumlah bit yang lebih pada instruksi, ini juga berarti jumlah bit untuk alamat makin berkurang. Salah satu teknik alokasibit adalah dengan menentukan bit opcode bersifat dinamis (dengan ketentuan minimum jumlah bit). Pada sistem dengan panjang instruksi tetap, ini berarti pengurangan bit untuk operan.

Beberapa faktor yang mempengaruhi penentuan bit alamat diantaranya :

* Jumlah jenis pengalamatan
* Jumlah Operan
* Register vs memory
* Jumlah set register
* Rentang alamat
* Granulitas hardware

1. Variasi Panjang Instruksi

Sesuai dengan namanya, teknik ini mengijinkan instruksi untuk memiliki panjang yang berbeda-beda.Dengan demikian maka teknik ini dapat mengakomodasi format instruksi yang berbeda-beda, bersifat lebih fleksibel, efesien serta ringkas.Penggunaan teknik ini juga tetap memerlukan agar sebuah instruksi berkorelasi secara integral dengan panjang sebuah word.Semua kemampuan teknik ini secara otomatis menaikkan kompleksitas processor.

**JENIS JENIS INTRUKSI**

Dalam program bahasa assembly terdapat 2 jenis yang kita tulis dalam program:

1. Assembly Directive (yaitu merupakan kode yang menjadi arahan bagi assembler/compiler untuk menata program)

2. Instruksi (yaitu kode yang harus dieksekusi oleh CPU mikrokontroler dengan melakukan operasi tertentu sesuai dengan daftar yang sudah tertanam dalam CPU)

Daftar Assembly Directive

|  |  |
| --- | --- |
| Assembly Directive | Keterangan |
| EQU | Pendefinisian konstanta |
| DB | Pendefinisian data dengan ukuran satuan 1 byte |
| DW | Pendefinisian data dengan ukuran satuan 1 word |
| DBIT | Pendefinisian data dengan ukuran satuan 1 bit |
| DS | Pemesanan tempat penyimpanan data di RAM |
| ORG | Inisialisasi alamat mulai program |
| END | Penanda akhir program |
| CSEG | Penanda penempatan di code segment |
| XSEG | Penanda penempatan di external data segment |
| DSEG | Penanda penempatan di internal direct data segment |
| ISEG | Penanda penempatan di internal indirect data segment |
| BSEG | Penanda penempatan di bit data segment |
| CODE | Penanda mulai pendefinisian program |
| XDATA | Pendefinisian external data |
| DATA | Pendefinisian internal direct data |
| IDATA | Pendefinisian internal indirect data |
| BIT | Pendefinisian data bit |
| #INCLUDE | Mengikutsertakan file program lain |

Daftar Instruksi

|  |  |
| --- | --- |
| Instruksi | Keterangan Singkatan |
| ACALL | Absolute Call |
| ADD | Add |
| ADDC | Add with Carry |
| AJMP | Absolute Jump |
| ANL | AND Logic |
| CJNE | Compare and Jump if Not Equal |
| CLR | Clear |
| CPL | Complement |
| DA | Decimal Adjust |
| DEC | Decrement |
| DIV | Divide |
| DJNZ | Decrement and Jump if Not Zero |
| INC | Increment |
| JB | Jump if Bit Set |
| JBC | Jump if Bit Set and Clear Bit |
| JC | Jump if Carry Set |
| JMP | Jump to Address |
| JNB | Jump if Not Bit Set |
| JNC | Jump if Carry Not Set |
| JNZ | Jump if Accumulator Not Zero |
| JZ | Jump if Accumulator Zero |
| LCALL | Long Call |
| LJMP | Long Jump |
| MOV | Move from Memory |
| MOVC | Move from Code Memory |
| MOVX | Move from Extended Memory |
| MUL | Multiply |
| NOP | No Operation |
| ORL | OR Logic |
| POP | Pop Value From Stack |
| PUSH | Push Value Onto Stack |
| RET | Return From Subroutine |
| RETI | Return From Interrupt |
| RL | Rotate Left |
| RLC | Rotate Left through Carry |
| RR | Rotate Right |
| RRC | Rotate Right through Carry |
| SETB | Set Bit |
| SJMP | Short Jump |
| SUBB | Subtract With Borrow |
| SWAP | Swap Nibbles |
| XCH | Exchange Bytes |
| XCHD | Exchange Digits |
| XRL | Exclusive OR Logic |

untuk yang lebih jelas dan detil:  
a. MOV  
Perintah MOV adalah perintah untuk mengisi, memindahkan,memperbaruhi isi suatu register, variable ataupun lokasi memory, Adapun tata penulisan perintah MOV adalah :  
MOV [operand A], [Operand B]  
Contoh :  
MOV AH,02  
Operand A adalah Register AH  
Operand B adalah bilangan 02  
Hal yang dilakukan oleh komputer untuk perintah diatas adalahmemasukan 02 ke register AH.  
  
b. INT (Interrupt)  
Bila anda pernah belajar BASIC, maka pasti anda tidak asing lagi dengan perintah GOSUB. Perintah INT juga mempunyai cara kerja yang sama dengan GOSUB, hanya saja subroutine yang dipanggil telah disediakan oleh memory komputer yang terdiri 2 jenis yaitu :  
- Bios Interrupt ( interput yang disediakan oleh BIOS (INT 0 – INT 1F))  
- Dos Interrupt ( Interrupt yang disediakan oleh DOS (INT 1F – keatas))

c. Push  
Adalah perintah untuk memasukan isi register pada stack, dengan tata penulisannya:POP [operand 16 bit]  
  
d. Pop  
perintah yang berguna untuk mengeluarkan isi dari register/variable dari stack,dengan tata penulisannya adalah : POP [operand 16 bit]  
  
e. RIP (Register IP)  
Perintah ini digunakan untuk memberitahu komputer untuk memulai memproses program dari titik tertentu.  
  
f. A (Assembler)  
Perintah Assembler berguna untuk tempat menulis program Assembler.  
-A100  
0FD8:100  
  
g. RCX (Register CX)  
Perintah ini digunakan untuk mengetahui dan memperbaruhi isi register CX yang merupakantempat penampungan panjang program yang sedan aktif  
  
  
pun, ada yang demikian:

1. Definisi Stack  
Secara harfiah stack berarti tumpukan, yaitu bagian memori yang digunakan untuk menyimpan nilai suatu register untuk sementara, membentuk tumpukan nilai. Stack dapat dibayangkan sebagai tabung memanjang (seperti tabung penyimpan koin). Sedangkan nilai suatu register dapat dibayangkan sebagai koin yang dapat dimasukkan dalam tabung tersebut. Jika ada data yang disimpan maka data-data tersebut akan bergeser ke arah memori rendah, dan akan bergeser kembali ke arah memori tinggi bila data yang disimpan telah diambil.

2. Perintah Perpindahan Data  
Terkait perpindahan data, bahasa assembler mempunyai beberapa perintah yang dapat dibedakan yaitu untuk memindahkan data tunggal seperti huruf atau angka dan untuk memindahkan data string yang berupa deretan huruf. Tetapi di sini hanya akan menjelaskan beberapa perintah yang dipakai dalam aplikasi.

2.1. PUSH/POP  
Syntax :

PUSH Reg16Bit

POP Reg16Bit

PUSH adalah perintah penyimpanan data ke memori stack secara langsung, dan untuk mengambil keluar nilai yang disimpan tersebut gunakan perintah POP. Nilai terakhir yang dimasukkan dalam stack, dengan perintah PUSH, akan terletak pada puncak tabung stack. Dan perintah POP pertama kali akan mengambil nilai pada stack yang paling atas kemudian nilai berikutnya, demikian seterusnya. Jadi nilai yang terakhir dimasukkan akan merupakan yang pertama dikeluarkan. Operasi ini dinamakan LIFO (Last In First Out). Perhatikan contoh berikut ini:

push ax;

push bx;

push cx;

mov ax, $31C;

mov bx; $31D;

mov cx, $31E;

pop cx;

pop bx;

pop ax;

2.2. MOV  
Syntax :

MOV destination, source

Digunakan untuk menyalin data dari memori/register ke memori/register atau dari data langsung ke register. Nilai pada source yang dipindahkan tidaklah berubah. Pada contoh di bawah, register al diberi nilai $31C kemudian nilai register al disalin ke register ax. Jadi sekarang nilai register al dan register axadalah $31C.

mov al, $31C;

mov ax, al;

Hal-hal yang tidak boleh dilakukan dalam penyalinan data:

a. Penyalinan data antarregister segmen (ds, es, cs, ss)

mov ds, es ? tidak dibenarkan

Gunakan register general, misalnya register ax, sebagai perantara

mov ax, es

mov ds, ax

atau gunakan stack sebagai perantara

push es

pop ds

b. Penyalinan data secara langsung untuk register segmen (ds, es, cs, ss)

mov ds, $31C ? tidak dibenarkan

Gunakan register general, misalnya register ax, sebagai perantara

mov ax, $31C

mov ds, ax

c. Penyalinan data langsung antarmemori

mov memB, memA ? tidak dibenarkan

Gunakan register general, misalnya register ax, sebagai perantara

mov ax, memA

mov memB, ax

d. Penyalinan data antarregister general yang berbeda daya tampungnya (8 bit dengan 16 bit) tanpa pointer

mov al, bx ? tidak dibenarkan

2.3. IN/OUT  
Syntax :

IN Reg16Bit, port

OUT port, Reg16Bit

Untuk membaca data dari suatu port dan memasukkan nilainya ke dalam suatu register gunakan perintah IN. Dan perintah OUT digunakan untuk memasukkan suatu nilai ke dalam suatu port. Nilai yang akan dimasukkan diberikan pada register al/ax dan alamat port diberikan pada register dx. Pada contoh berikut ini, pertama kali register dx disimpan pada stack, menyalin nilai $31E pada register dx kemudian perintah IN akan membaca nilai pada register dx (port bernilai $31E) dan memasukkannya ke dalamregister al. Dan terakhir nilai tersebut disalin ke variabel Data.

push dx

mov dx, $31E

in al, dx

mov Data, al

pop dx

Dan contoh berikut untuk memberi nilai ($8A) pada suatu port.

push dx

mov dx, $31E

mov al, $8A

out dx, al

pop dx

3. Operasi Aritmatika  
3.1. Penjumlahan  
Syntax :

ADD destination, source

ADC destination, source

INC destination

Perintah ADD akan menjumlahkan nilai pada destination dan source tanpa menggunakan carry (ADD), dimana hasil yang didapat akan ditaruh pada destination. Dalam bahasa pascal pernyataan ini sama dengan pernyataan destination := destination + source. Daya tampung destination dan source harus sama misalnya register al (8 bit) dan ah (8 bit), ax (16 bit) dan bx (16 bit). Perhatikan contoh berikut, nilairegister ah sekarang menjadi $10 :

mov ah, $5;

mov al, $8;

add ah, al

Perintah ADC digunakan untuk menangani penjumlahan dengan hasil yang melebihi daya tampungdestination yaitu dengan menggunakan carry (ADD), dalam bahasa pascal sama dengan pernyataandestination := destination + source + carry. Misalnya register ax (daya tampung 16 bit) diberi nilai $1234dan bx (16 bit) diberi nilai $F221, penjumlahan kedua register ini adalah $10455. Jadi ada bit ke 17 padahal daya tampung register bx hanya 16 bit, penyelesaiannya adalah nilai bx = $0455 dengan carry flag = 1.

Perintah INC digunakan untuk operasi penjumlahan dengan nilai 1. Jadi nilai pada destination akan ditambah 1, seperti perintah destination := destination + 1 dalam bahasa Pascal.

3.2. Pengurangan  
Syntax :

SUB destination, source

SBB destination, source

DEC destination

Perintah SUB untuk mengurangkan 2 operand tanpa carry flag. Hasilnya diletakkan pada destination dalam bahasa pasca sama dengan pernyataan destination := destination – source. Untuk mengenolkan suatu register, kurangkan dengan dirinya sendiri seperti contoh berikut ini. Pertama kali register axbernilai $5, kemudian nilai register tersebut dikurangi dengan dirinya sendiri sehingga terakhir nilairegister ax adalah 0.

mov ax, $15;

mov bx, $10;

sub ax, bx;

sub ax, ax;

Perintah SBB mengurangkan nilai destination dengan nilai source kemudian dikurangi lagi dengan carry flag (destination := destination – source – carry flag).

Dan perintah DEC untuk mengurangi nilai destination dengan 1.

3.3. Perkalian  
Syntax :

MUL source

Digunakan untuk mengalikan data pada accumulator dengan suatu operand dan hasilnya diletak pada register source. Register source dapat berupa suatu register 8 bit (misal bl, bh, dan sebagainya), register 16 bit (bx, dx, dan sebagainya) atau suatu variabel.

3.4. Pembagian  
Syntax :

DIV source

Operasi aritmatika ini pada dasarnya sama dengan operasi perkalian.

**MACAM MACAM ALAT**





**Reads51** merupakan Integrated Embedded Applications Development System milik Rigel, dimana sebuah IDE (Integrated Development Environment) yang support untuk keluarga Rigel 8051 embedded controller board. IDE support beberapa toolchain, terutama sebelum READS v3.x assembler dan simulator RChipSim v3.x. Hal ini membuat Reads51 menjadi sebuah pembaharuan bagi software tools sebelumnya. Selain itu, Reads51 berisi SmallC-compatible C compiler untuk keluarga mikrokontroler 8.051.

Reads51 berjalan pada PC IBM atau host yang kompatibel. Reads51 memungkinkan pengguna menulis, mengkompilasi, merancang, debugging, download, dan menjalankan perangkat lunak aplikasi dalam bahasa MCS-51. Reads51 berisi compiler C, relatif assembler, linker / locator, editor, Chip simulator, debugger bahasa assembly, dan komunikasi host-to-board user-friendly.

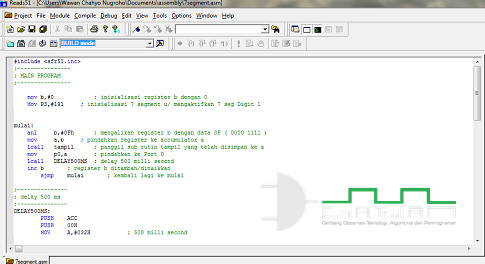
Fungsi Debugging terdiri dari breakpoints, single-stepping, source-level debugging, inspecting & modifying memory, internal registers, dan special function registers. Instruksi MCS-51 terdiri dari sistem bantuan on-line yang komprehensif.

Semua fungsi Reads51 dapat diaktifkan dari menu, toolbar, atau dengan kombinasi hot-key.

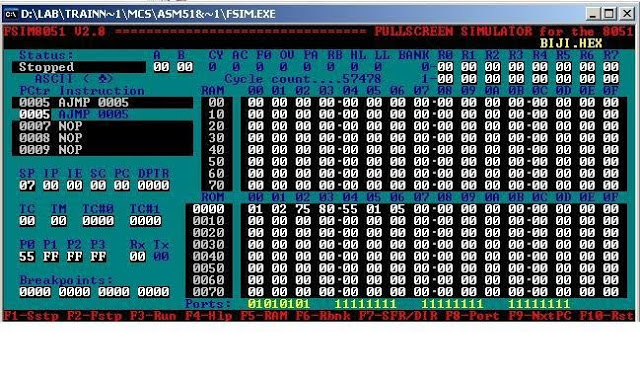
**Contoh Software :**

Disediakan juga source code untuk berexperimen dengan kemampuan Rigel's boards dan Reads51. Beberapa contoh dirancang untuk memberikan gambaran terhadap fitur keluarga mikrokontroler 8.051, khususnya digital dan serial I/O, timer, counter dan interrupt logic. Selain itu juga ada perangkat lunak seperti; routines for keypads, LCDs, stepper motors, dan DC motors.

Pengguna didorong untuk memodifikasi contoh software yang telah disediakan, hal ini dimaksudkan dalam rangka pengembangan aplikasi yang lebih spesifik versi mereka. Source code user dapat diakses melalui system calls.



Contoh screenshot program assembly menggunakan read51

1. .

Apa sih ASM51 ??   
ASM51 itu adalah sebuah kompiler untuk bahasa asembly untuk keluarga 8051. Jadi asm 51 sebagai compiler.   
  
Compiler maksudnya ?   
Compiler disini berarti bertujuan untuk mengubah bahasa mesin yang ditulis di textpad/notepad/text editor. Yang disave dalam betuk namafile.asm di kompile menjadi file bahasa mesin .Hex   
  
.ASM tidak sama dengan .HEX ?   
Tidak. Asembly (asm) tidak sama dengan bahasa mesin (HEX)   
sejarahnya dimulai ketiak perangkat elektronik atau ic hanya dapat mengetahui nilai 1 dan 0.   
1 = keadaan ada listrik   
0 = keadaan tidak ada listrik.   
  
Untuk memerintah sebuah prosesor / micro controller yang didalamnya terdiri dari rangkaian rangkain elktronik maka hanya tahu kondisi listrik mengalir dan listrik tidak mengalir.   
  
Jadi kompiler adalah sebagai jembatan. Menghubungkan program kita ke bahasa mesin. Karena format bahasa mesin haruslah nilai 1 dan 0.



Setelah selesai membuat dan mengconvert pemrograman ke dalam bentuk file .HEX selanjutnya adalah mengirim/mengupload file HEX tersebut ke dalam IC AT89S51 yang telah terpasang pada sistem minimum. Jalur koneksi yang digunakan untuk mengupload seperti yang telah dijelaskan pada sistem minimum diatas, dan selanjutnya adalah penggunaan software EAC-ISP.exe untuk mengupload file tersebut. Komponen software dasar yang dibutuhkan ketika mengupload adalah harus tersedianya software EAC-ISP.exe serta terdapat file sfr51.inc pada direktori dimana program kita akan di upload. Tanpa adanya kedua komponen tersebut kita tidak dapat mengupload program yang telah dibuat.

Pada software AEC-ISP terdapat beberapa item menu diantaranya :

1. Load HEX File to Flash Buffer
2. Load HEX File to EEPROM Buffer
3. Display Flash Buffer
4. Display EEPROM Buffer
5. Program
6. Read Flash & EEPROM to Buffer
7. Save Flash Buffer to Hex File
8. Save EEPROM Buffer to Hex File
9. Reset
10. Setup
11. Show Scematic
12. About Us
13. Exit

**JENIS JENIS OPERAND**

**JENIS OPERAND**

          Instruksi mesin melakukan operasi terhadap data. Pada umumnya data dikategorikan ke dalam angka, karakter dan data logika.

**a.**        **Angka**

       Setiap bahasa mesin mengandung tipe data numerik. Umunya terdapat tiga tipe data angka yang ada pada komputer yaitu:

1.        Binary integer

2.        Binary floating point

3.        Desimal

          Semua operasi pada internal komputer berupa data biner, namun user berinteraksi dengan bilangan desimal. Maka perlu dilakukan konversi dari desimal ke bilangan biner pada input dan konversi dari biner ke desmimal pada output. Bilangan desimal direpresentasikan dalam 4 bit kode biner maka 0=0000, 1=0001,...,8=1000, 9=1001. Sedangkan untuk desimal 246 = 0000 0010 0100 0110. Untuk bilangan negatif direpresentasikan dengan 4 bit yang diletakkan pada awal atau akhir string. Standar tanda yang digunakan adalah 1100 untuk bilangan positif dan 1101 untuk tanda bilangan negatif.

**b.**        **Karakter**

          Umumnya bentuk data adalah teks atau kumpulan karakter. Sedangkan sistem komputer didesain untuk data biner. Maka sejumlah kode dalam urutan bit perlu di tentukan untuk merepresentasikan sebuah karakter. Saat ini standar kode yang digunakan untuk merepresentasikan karakter adalah *American Standart Code for Information Interchange*(ASCII). Setiap karakter pada kode ASCII direpresentasikan dengan 7 bit biner yang unik. Maka terdapat 128 karakter yang berbeda yang dapat direpresentasikan. Selain itu juga ada yang menggunakan *Extended Binary Coded Decimal Interchange Code*(EBCDIC) yang digunakan oleh IBM mainframe.

**c.**         **Data logika**

       Pada umumnya setiap word atau yang lain merupakan satu unit data yang masing masing unit data memiliki nilai 0 atau 1. Ketika dipandang dengan cara ini, maka data tersebut dianggap sebagai data logika. Data logika hanya bernilai *true* “1” atau *false* “0”.

**JENIS OPERASI**

Jumlah *opcode* dari sebuah mesin ke mesin lain beragam. Akan tetapi tipe operasi-operasi umum akan sama untuk semua mesin.  Berikut dikategorikan operasi berdasarkan fungsi dan tipenya:

a.       Transfer data

b.      Aritmatikaa

c.       Logika

d.      Konversi

e.       Input/output

f.       Kendali sistem

g.      Kendali transfer

Tabel 1. Contoh Instruksi Umum Pada CPU

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Tipe** | **Instruksi** | |
| **Nama** | **Aksi** |
| 1 | Transfer data | MOVE | Mentransfer data dari lokasi sumber ke lokasi tujuan |
| LOAD | Mentransfer data dari lokasi memori ke register CPU |
| STORE | Mentransfer data dari register CPU ke lokasi memori |
| PUSH | Mentransfer data dari sumber ke stack |
| POP | Mentransfer data dari stack ke tujuan |
| XCHG | Saling menukar isi sumber dan tujuan |
| CLEAR | Me-reset tujuan dengan semua bit ‘0’ |
| SET | Mengeset tujuan dengan semua bit ‘1’ |
| 2 | Aritmatika | ADD | Penjumlahan, hitung jumlah dari 2 operan |
| SUB | Pengurangan, hitung selisih dari 2 operan |
| MUL | Perkalian, hitung hasil kali dari 2 operan |
| DIV | Pembagian, hitung hasil bagi dari 2 operan |
| NEG | Negasi, ganti tanda operan |
| INC | Tambahkan 1 pada operan |
| DEC | Kurangkan 1 dari operan |
| SHIFT A | Geser operan (kekiri atau kekanan) dengan tanda |
| 3 | Logika | NOT | Komplemenkan (komplemen 1) operan |
| OR | Lakukan operasi logika OR pada operan |
| AND | Lakukan operasi logika AND pada operan |
| XOR | Lakukan operasi logika XOR pada operan |
| SHIFT | Geser operan (kekiri atau kekanan), isi nilai pada ujung bit |
| ROT | Geser operan (kekiri atau kekanan) dengan berputar |
| TEST | Uji kondisi yang ditetapkan dan pengaruhi flag yang sesuai |
| 4 | Kendali Transfer | JUMP | Perpindahan tak bersyarat, masukkan alamat yang ditetapkan ke PC |
| JUMPIF | Perpindahan bersyarat, masukkan alamat yang ditetapkan ke PC jika kondisi terpenuhi |
| JUMPSUB | CALL, simpan ‘status program control’ yang sekarang, pindah kealamat yang ditetukan ke PC |
| RET | RETURN, restore ‘status program control’ dari stack ke PC dan register/flag yang relevan lainnya |
| 5 | Input/Output | IN (read) | Mentransfer data dari perangkat atau port i/o yang ditentukan ke tujuan (memori utama atau register) |
|  |  | OUT (write) | Mentransfer data dari sumber yang ditentukan ke perangkat atau port i/o |
|  |  | START I/O | Mentransfer instruksi ke prosesor i/o untuk menginisiasi operasi i/o |
|  |  | TEST I/O | Mentransfer informasi status dari sistem i/o ke instruksi yang ditentukan |
| 6 | Konversi | TRANSLATE | Menterjemahkan nilai-nilai dalam suatu bagian memori berdasarkan tabel korespodensi |
|  |  | CONVERT | Mengkonversi isi suatu word dari suatu bentuk ke bentuk lainnya (contoh decimal ke biner) |

**a. Transfer data**

Tipe instruksi mesin yang paling dasar yaitu instruksi transfer data. Pada instruksi transfer dataharus ditentukan beberapa hal. Pertama, penentuan lokasi sumber dan tujuan dari operan.Lokasinya dapat terletak di memori, register atau stack. Kedua, panjang data yang akan ditransfer harus diketahui. Ketiga, sama untuk semua instruksi dengan operan, cara pengalamatannya harus ditentukan.

Dari sisi aksi prosesor, operasi transfer data mungkin merupakan tipe yang paling sederhana. Jika kedua-duanya baik sumber maupun tujuan adalah register, maka  prosesor hanya menyebabkan data dipindahkan dari satu register ke register lain (operasi internal prosesor). Jika salah satu atau kedua operan berada dalam memori, maka prosesor harus melakukan beberapa atau semua tindakan berikut:

1.      Menghitung alamat memori, berdasarkan mode pengalamatan ( dibahas di bagian selanjutnya)

2.      Jika alamat mengacu pada virtual memori, menerjemahkan dari alamat memori virtual ke alamat memori sebenarnya/fisik.

3.      Menentukan apakah operan yang dituju ada di dalam chace

4.      Jika tidak, berikan perintah ke modul memori.

**b. Aritmatika**

Kebanyakan mesin menyediakan operasi aritmatika / perhitungan dasar  sepertitambah, kurang,kali dan bagi. Dimana operasi tersebut disediakan untuk menangani bilanganinteger bertanda (fixed-point), juga bilangan floating point atau desimal. Berikut contoh lain operasi yang termasuk jenis instruksi dengan satu operan:

·         Absolute : mengambil nilai absolut/mutlak dari operan

·         Negate : menegasikan operan

·         Increment: menambahkan 1 nilai ke operan

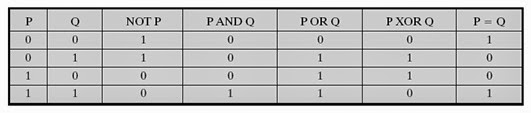
·         Decrement: mengurangi 1 nilai dari operan

Eksekusi instruksi aritmatika dapat melibatkan operasi transfer data untuk menempatkan operan dari input ke ALU, dan untuk mengantarkan output dari ALU.

**c. Logika**

Kebanyakan mesin juga menyediakan berbagai operasi untuk memanipulasi setiap bit dari sebuah word atau unit (yang dapat diberi alamat) lainnya, operasi ini juga di sebut "bit twiddling". Bit-bit tersebut didasarkan pada operasi boolean. Beberapa operasi logika dasar dapat dilakukan pada data boolean atau biner yang ditunjukkan pada tabel berikut.

Tabel 2. Contoh Operasi Logika Dasar

[](http://1.bp.blogspot.com/-siAtCqK5ljo/VEp0SItNlCI/AAAAAAAAAII/FJsRigNAC6E/s1600/contoh%2Boperasi%2Blogika%2Bdasar.jpg)

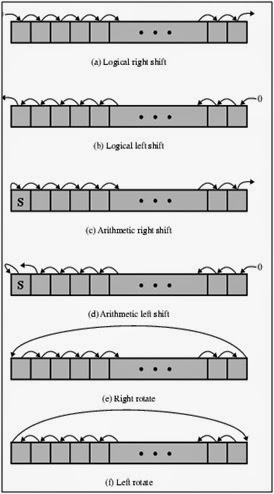
Operasi-operasi logika dapat diterapkan pada bitwise ke n-bit unit data. Dengan demikian, jika dua register berisi data

(R1) = 10100101

(R2) = 00001111

Kemudian

(R1) AND (R2) = 0000101

[](http://4.bp.blogspot.com/-H0U_2I3HVAU/VEp0sTQ_8kI/AAAAAAAAAIQ/AEF3z3c7bDg/s1600/operasi%2Bpergeseran%2Bdan%2Bperputaran.jpg)Selain operasi logika bitwise, kebanyakan mesin menyediakan berbagai fungsi pergeseran dan perputaran.Operasi yang paling dasar digambarkan pada gambar dibawah.Dengan logika pergeseran, setiap bit dari word akan di geser ke kiri atau ke kanan. Pada salah satu ujungnya, bit yang bergeser keluar akan hilang. Pada ujung lainnya, nilai ‘0’ digeser masuk.Pergeseran logis berguna terutama untuk mengisolasi bagian dalam sebuah word.Nilai ‘0’ yang digeser kedalam sebuah word menggantikan informasi yang tidak diinginkan yang digeser dari ujung lainnya.

Gambar 1. Operasi Pergeseran dan Perputaran

Operasi pergeseran aritmatika menangani data sebagai integer bertanda dan tidak menggeser bit tanda. Pada pergeseran aritmatika ke kanan, bit tanda disalin pada bit yang berada dikanannya. Pada pergeseran aritmatika ke kiri, pergeseran logika kiri dilakukan pada semua bit kecuali bit tanda tetap ditahan. Operasi ini dapat mempercepat operasi aritmatikaa tertentu.

Rotate, atau pergeseran memutar, operasi ini menjaga seluruh bit dioperasikan. Salah satu penggunaan dari rotasi yaitu membawa setiap bit berturut-turut ke bit paling kiri, dimana itu dapat diidentifikasi dengan menguji tanda dari data (diperlakukan sebagai angka). Sama dengan operasi aritmatikaa, operasi logika melibatkan aktifitas ALU dan mungkin melibatkan operasi transfer data.

**d. Konversi**

Instruksi konversi adalah instruksi-instruksi yang mengubah format atau beroperasi pada format data.Contohnya yaitu mengkonversi dari desimal ke biner.

**e. Input/Output**

Seperti yang kita ketahui, ada beberapa pendekatan I/O yang bisa diambil, diantaranya programmed I/O (isolated&memory mapped), DMA, dan penggunaan prosesor I/O. Implementasi instruksi I/O banyak dilakukan dengan hanya menyediakan beberapa instruksi I/O, dengan tindakan spesifik yang ditentukan oleh parameter, kode, atau kata perintah.

**f. Kendali Sistem**

Instruksi kendali sistem adalah instruksi yang dapat dieksekusi hanya ketika prosesor dalam keadaan tertentu atau mengeksekusi program pada area khusus dalam memori.Biasanya, instruksi ini dipesan untuk digunakan sistem operasi.Berikut beberapa contoh operasi kendali sistem. Sebuah instruksi kendali sistem boleh membaca atau mengubah kendali register. Contoh lainnya adalah instruksi untuk membaca atau memodifikasi penyimpanan protection key, seperti yang digunakan pada sistem memori EAS/390. Contoh lain adalah akses untuk memproses blok kontrol dalam sistem multiprogramming.

**g. Kendali Transfer**

Untuk semua tipe operasi yang sudah dibahas sejauh ini, instruksi selanjutnya yang akan dibahas tepat setelah ini, pada memori, adalah kendali transfer. Namun, pecahan yang signifikan dari instruksi dalam setiap program memiliki fungsi mengubah urutan eksekusi instruksi. Untuk instruksi ini, operasi yang dilakukan oleh prosesor

Beberapa hal yang perlu diperhatikan yaitu:

1.      Dalam praktek penggunaan komputer, sebenarnya kita mengeksekusi tiap instruksi lebih dari sekali dan mungkin ribuan kali. Ini membutuhkan ribuan atau bahkan jutaan instruksi untuk mengimplementasikan aplikasi. Hal ini tidak mungkin jika tiap instruksi harus ditulis secara terpisah. Jika tabel atau daftar item akan diproses, dibutuhkan program looping. Satu urutan eksekusi akan dieksekusi berulang kali untuk memproses semua data.

2.      Hampir semua program melibatkan beberapa pembuatan keputusan. Kita setuju komputer akan melakukan sesuatu jika suatu kondisi terpenuhi, dan melakukan hal lain jika dalam kondisi lain. Sebagai contoh, sebuah urutan instruksi mengitung akar kuadrat dari sebuh nilai. Pada awal urutan, tanda dari nilai tersebut diuji. Jika negatif, komputasi tidah dilakukan, tetapi kondisi eror yang akan dilaporkan.

3.      Untuk mengubah dengan benar program yang besar atu sedang adalah tugas yang sangat sulit. Hal ini akan lebih mudah jika ada mekanisme untuk memecah tugas-tugas tersebut menjadi bagian-bagian kecil yang dapat dikerjakan sekali dalam satu waktu.

            Sekarang kita akan membahas operasi transfer kendali yang ada pada set instruksi : seperti *branch*, *skip*, dan *procedurecall*. Instruksi*brach* (cabang) sering juga disebut instruksi*jump*, memiliki sebuah operan yang berisi alamat dari instruksi selanjutnya yang akan dieksekusi. Instruksi ini dapat dibedakan menjadi *conditional branch*dan *unconditional branch.* Instruksi *skip*digunakan untuk melewati baris instruksi dan tidak membutuhkan alamat tujuan. Contohnya intruksi ISZ *(increment-skip-if-zero*). Instruksi*procedure call* digunakan untuk pemanggilan*procedure*(subprogram). Mekanisme *procedure*terdiri dari 2 instruksi utama : instruksi *call* yang berarti melakukan percabangan ke lokasi yang menunjuk ke procedure, dan instruksi *return*yang berarti kembali dari *procedure* ke lokasi dimana *procedure* tersebut dipanggil. Kedua instruksi tersebut menggunakan instruksi percabangan.