FORMAT INSTRUKSI

Intruksi bahasa mesin

Struktur umum.

Opcode	**Operand$_1$**	**Operan$_2$**	**.....**

Opcode (kode Operasi) :

Operation code, biner tak bertanda yang uni untuk menerangkan operasi yang harus dieksekusi.

Set intruksi :

Set dari seluruh kode operasi yang diperlukan prosessor pada waktu mengekseskusi kode operasi i bit, maka jumlah kode operasi (unik)2^i.

Menyimpan nilai-nilai yang diperlukan prosessor pada waktu, mengeksekusi kode operasi. Kolom operand, disebut juga address field, dapat berisi :

1. **Data/konstanta yang langsung dipakai pada operasi kode operasi.**
2. **alamat loksi dimana data tersebut disimpan.**

Jumlah maksimum operand dalam suatu computer menunjukkan organisasi prosessor mesin tersebut.

Ada 5 macam organisasi prosessor(berdasarkan format instruksi) :

1. **Prosesor dengan format 4 alamat**
   - **1.)Operand kiri**
   - **2.)Operand kanan**
   - **3.)Hasil operasi**
   - **4.)Alamat instruksi selanjutnya.**

Opcode	**Opd$_1$**	**Opd$_2$**	**Opd$_3$**	**Opd$_4$**
	Opd kiri	**Opd kanan**	**Hasil**	**next addr.**

Contoh : a := b + c – d

0110	ADD	A	B	X	0111
0111	SUB	X	B	X	0000

1. **ADD b,c,tmp,2**
2. **SUB tmp,d,a,3**
3. **.....**

Format ini terlalu panjang. Jadi tidak lagi dipergunakan.

2. **Prosesor dengan format 3-alamat**

1.)Operand kiri

2.)Operand kanan

3.)Hasil operasi

Opcode	Opd_1	Opd_2	Opd_3

Opd kiri Opd kanan hasil

Contoh : a := b + c – d

0110	ADD	A	B	X
0111	SUB	X	B	X

Program biasanya diletakan berurutan dalam satu blok memori. Instruksi dilaksanakan secara berurutan, kecuali secara ekseplisit dinyatakan (seperti pada instruksi pencabangan). Pada mesin ini diperlukan register khusus untuk menyimpan alamat dari instruksi yang akan dieksekusi. Register ini disebut Program Counter Register(PC).

3. **Prosesor 2- alamat :**

1.)Operand kiri

2.)Operand kanan

Opcode	Opd_1	Opd_2

Opd kiri OPd kanan

Perjanjian yang berlaku pada prosessor 3- alamat juga berlaku. Selain itu hasil sudah ditentukan akan disimpan dimana. Misalnya pada salah satu operand atau pada register khusus. Jika hasil diletakan pada salah satu operand (misalnya operand kedua), maka harus hati-hati karena nilai operand kedua berubah. Gunakan lokasi sementara untuk mencagahnya.

Contoh : A := B + C – D

0110	ADD	B	C
0111	SUB	B	D
0112	MOV	A	B

Hasil disimpan di operand kiri, dan b,c,d tidak boleh berubah.

MOV tmp ,btmp ← b{memindah b ke lokasi sementara}

ADD tmp,c tmp ← tmp + c

SUB tmp,d tmp ← tmp – d

MOV tmp,a a ← tmp

STA

4. **Prosesor dengan foramat 1-alamat**

   Hanya memiliki satu operand.

   Perlu PC

   - **Perlu PC**
   - **Address terurut**
   - **Perlu register untuk menyimpan operand**
   - **Hasil ditempatkan di register tersebut**

Opcode	Opd_1

0110	LOAD	B	Acc ß B
0111	ADD	C	Acc ß Acc + C
0112	SUB	D	Acc ß Acc - D
0113	STORE	A	A ß Acc

Operand kedua dan hasil sudah ditentukan diletakkan dimana, misalnya di R_0.

ADD a berarti $R_0 = a + R_0$

5. Prosesor dengan format 0-alamat(Stack machine)
   - Menggunakan Stack.
   - Intruksi dengan format inin menasumsikan operand – operand sudah ada di stack. Hasil operasi akan diletakkan di stack juga.
   - Ekspresi aritmatika yang akan dijalankan di mesin sudah harus dalam bentuk/notasi polish.
   - Contoh mesin 0-alamat adalah kalkulator.

Address space misal : address space 4 bit opcode = $2^4 = 16$

Cell size misal : address space 8 bit opcode = $2^8 = 256$

Opcode

Notasi Ekspresi Aritmatika

1. Infiks → A + B
2. Prefiks (Polish notation) → + A B
3. Postfix (Reverse Polish notation) → A B +

Dengan notasi prefiks, operand pada stack menjadi mudah

1. Baca ekspresi dari kanan ke kiri
2. Jika membaca operand, push operand ke stack
3. Jika membaca operator, pop sejumlah operand dari stack dan lakukan operasi aritmatika
4. Simpan hasil stack.

Contoh :

1. **–c + ba**

   PUSH a

   PUSH b

   ADD {Pop a dan b dari stack dan ditambahkan, c := a + b}

   PUSH c

   SUB

Algoritma untuk merubah ekspresi infiks ke postfix.

1. Baca dari kiri ke kanan.
2. Jika baca operand, tulis.
3. Jika baca "(", push ke stack.
4. Jika baca ")", pop seluruh isi stack dan tulis sampai dengan ditemukan "(". Tanda kurung tidak ditulis.
5. Jika baca operator dan isi stack kosong, push ke stack.
6. Jika baca operator(O_n)dan Top berisi (O_{n-1})

   Jika presenden $O_n > O_{n-1}$, Push O_n.

   Jika presenden $On <= O_{n-1}$, Pop(dan tulis) O_{n-1} dari stack, lalu push O_n yang baru di baca.
7. Pada akhir ekspresi pop dabn tulis seluruh isi stack.

Presenden operator

1. **/ = ***
2. **+ = -**
3. **(/,*) > (+,-)**
4. **/, *, +, -, >, (**

Dengan notasi postfix pop dan tulis seluruh isi stack.

1. **Baca ekspresi dari kiri ke kanan.**
2. **Jika membaca operand, push operand ke stack**
3. **Jika membaca operator, pop sejumlah operand dari stack dan lakukan operasi aritmatika (perhatikan urutan).**
4. **Simpan hasil stack**

Transormasi dari infiks ke frefiks?

PROCESSOR

Komponen prosessor :

1. **Register**

   General Purpose

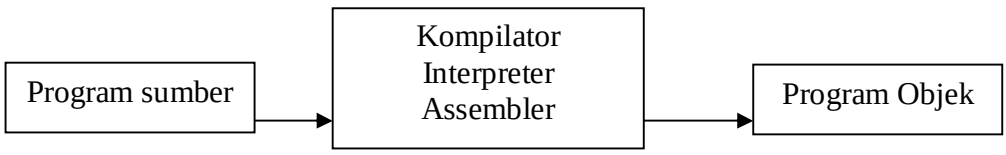
   Special Purpose, a.l. : PC (atau IC : Program/Instruction Counter), SP (Stack Pointer Register), IR(Intruction Register).
2. **Control Unit/Squencer**
3. **ALU (Aritmetic and logic Unit).**

Pada saat ini tidak ada mesin yang mengunakan prosessor 4-alamat.

Jika disebutka "prosessor 3-alamat", kemungkinan instruksinya memiliki 0,1,2,atau 3 alamat. Jika ada kolom operand yang tidak diisi.

Pada umumnya mesin memiliki stack dan sejumlah register.

Prosesor intel 8080 memiliki stack yang dipakai hanya untuk penyimpanan '*return address*'. Mesin ini tidak dikelopokkan dalam mesin 0-alamat.



Fortran, pascal, C, **Bahasa Mesin**

Assembler

Contoh : Intruksi dari program sumber x := (a +b)/(c – d)

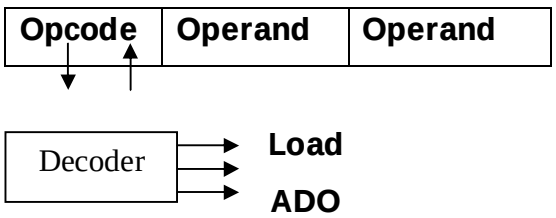
Program objek (ditulis dalam format '*pseudo assembly*')

Mesin 0-Alamat	Mesin 1- alamat	Mesin 2-alamat
PUSH a		LDA R_o,a
PUSH b		ADD R_o,b
ADD		STA t_1,R_o
PUSH c		LDA R_o,c
PUSH d		SUB R_o,d
SUB		STA t_2
DIV		LDA R_o,t_1
POP x		DIV R_0,t_2
		STA x, R_o

Intruction Register (Register Intruksi) dan Dekoder Intruksi

Register yang menyimpan intruksi yang sedang dieksekusi.

2-alamat



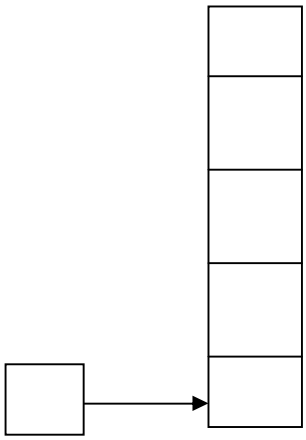
Program Counter (PC) / Intruction Counter

Register yang menyimpan alamat dari intruksi yang akan dieksekusi.

Pada eksekusi program sekuensial, selesai membaca intruksi isi PC diinkrementasi. Pada eksekusi percabangan, PC diisi dengan alamat tujuan pencabangan.

Stack Pointer Register

Stack biasa diimplementasikan dengan satu set memori dengan alamat yang berturutan. Alamat 'Top of Stack' disimpan dalam satu register khusus, biasanya disebut "Stack Pointer Register (SP)".



SP

ACC (Accumulator) AX AH AL

BX Data → 1 cell size MBR

CS

DS

ES Address → 1 address space MAR

SS

Operasi pada Komputer

Siklus operasi : Fetch

Execute

Fetch : baca intruksi di memori (alamat ada di PC) dan load intruksi ke IR

Operasi Mikro :

MAR ← PC

Read (MBR ← RAM – MAR)

IR ← MBR

PC ← PC+1

Execute : decode instruksi di IR dan aktifkan operasi (mikro yang tepat)

Contoh :

LOAD ADD JNE

Setiap operasi mikro berarti mengaktifkan sinyal untuk satu atau lebih komponen. Cara pengaktifannya sinyal control : hardware dan microprogrammed.

Hardware : sinyal-sinyal control dibangkitkan dari perangkat keras(rangkaian logika). Prosesor yang menggunakan pendekatan ini disebut *hardwared processor*.

Microprogrammed atau firm – wired : intruksi bahasa mesin ditranslasikan dalam instruksi mikro(*microcode*/*micro instruction*).

Program bahasa tinggkat tinggi → Program objek → Program dalam microcode.

Setiap microcode mengaktifkan komponen rangkaian tertentu. Proosesor yang menggunakan pendekatan ini disebut *microprogrammed processor*.

Komputer modern umumnya mengunakan konsep ini.

Pemrograman micro

- **Proses menulis program didalam bahasa micro.**
- **Komputer deprogram pada tingkat di bawah bahasa mesin.**
- **Setiap instruksi mikro akan mengaktifkan satu atau lebih rangkaian : register, bus, atau ALU, dst.**

  Contoh : MOV dest, srce

Dahulu apakah computer deprogram mikro atau hardwared tidak tampak oleh pengguna. Beberapa computer masa kini memberikan kesempatan kepada pengguna untuk menulis program mikronya dan disimpan dalam WCS (Writeable Control Store).

Tujuannya :

- **Simulasi set intruksi dari computer lain (emulasi)**
- **Membuat operasi baru**
- **Optimasi eksekusi operasi tertentu.**