

EC3003 - Sistem Komputer

Bagian 5

Pemrograman Bahasa Mesin

Pembahasan

- ✚ Model Eksekusi Pemrograman Assembly
- ✚ Pengaksesan Data dan Informasi
 - Register
 - Memori
- ✚ Operasi Aritmatika

Prosesor IA32

Mendominasi pasar komputer

Evolusi desain

- Dimulai tahun 1978 dengan 8086
- Sejalan dengan waktu, kemampuan terus bertambah
- Masih mendukung kemampuan lama, walau usang

Complex Instruction Set Computer (CISC)

- Berbagai instruksi berbeda dengan berbagai format berbeda
 - Tetapi hanya sebagian kecil yang digunakan pada program Linux
- Kinerjanya sulit dibandingkan dengan Reduced Instruction Set Computers (RISC)

Evolusi X86 : Sisi Pemrogram

- ✚ 8086 (1978; 29 ribu transistor)
 - Prosesor 16 bit.
 - Dasar untuk IBM PC dan DOS
 - Memori terbatas 1 MB.
 - DOS hanya menyediakan 640K
- ✚ 80286 (1982; 134 ribu transistor)
 - Penambahan skema pengalamatan
 - Dasar untuk IBM PC-AT dan Windows
- ✚ 80386 (1985; 275 ribu transistor)
 - Diperbesar menjadi 32 bit.
 - Flat addressing
 - Dapat menjalankan Unix
- ✚ 80486 (1989; 1,9 juta transistor)

Evolusi X86 : Sisi Pemrogram

- ✚ Pentium (1993; 3,1 juta transistor)
- ✚ Pentium/MMX (1997; 4,5 juta transistor)
 - Terdapat tambahan koleksi instruksi khusus untuk operasi data integer 1,2 atau 4 byte dalam vektor 64 bit
- ✚ Pentium Pro (1995; 6,5 juta transistor)
 - Terdapat penambahan instruksi move
 - Terdapat perubahan besar pada mikroarsitektur
- ✚ Pentium III (1999, 8,2 juta transistor)
 - Penambahan instruksi “streaming SIMD” untuk operasi data floating point atau integer 1,2 atau 4 byte dalam vektor 128 bit
- ✚ Pentium 4 (2001; 42 juta transistor)
 - Penambahan format 8 byte dan 144 instruksi baru untuk mode streaming SIMD

Evolusi X86 : Clone

Advanced Micro Devices (AMD)

Sejarah

- AMD mengikuti di belakang Intel
- Lebih lambat, lebih murah

Saat ini

- Merekrut perancang rangkaian ternama dari Digital Equipment Corp (DEC)
- Intel terpacu untuk membuat IA64
- Sekarang merupakan kompetitor Intel

Mengembangkan dirinya hingga 64 bit

Evolusi X-86 : Clone

Transmeta

- ✚ Baru muncul
 - Anak buah Linus Torvalds
- ✚ Memiliki pendekatan yang sangat berbeda dalam implementasinya
 - Menerjemahkan kode x86 menjadi kode “Very Long Instruction Word” (VLIW)
- ✚ Ditujukan untuk pasar low-power

Spesies Baru : IA64

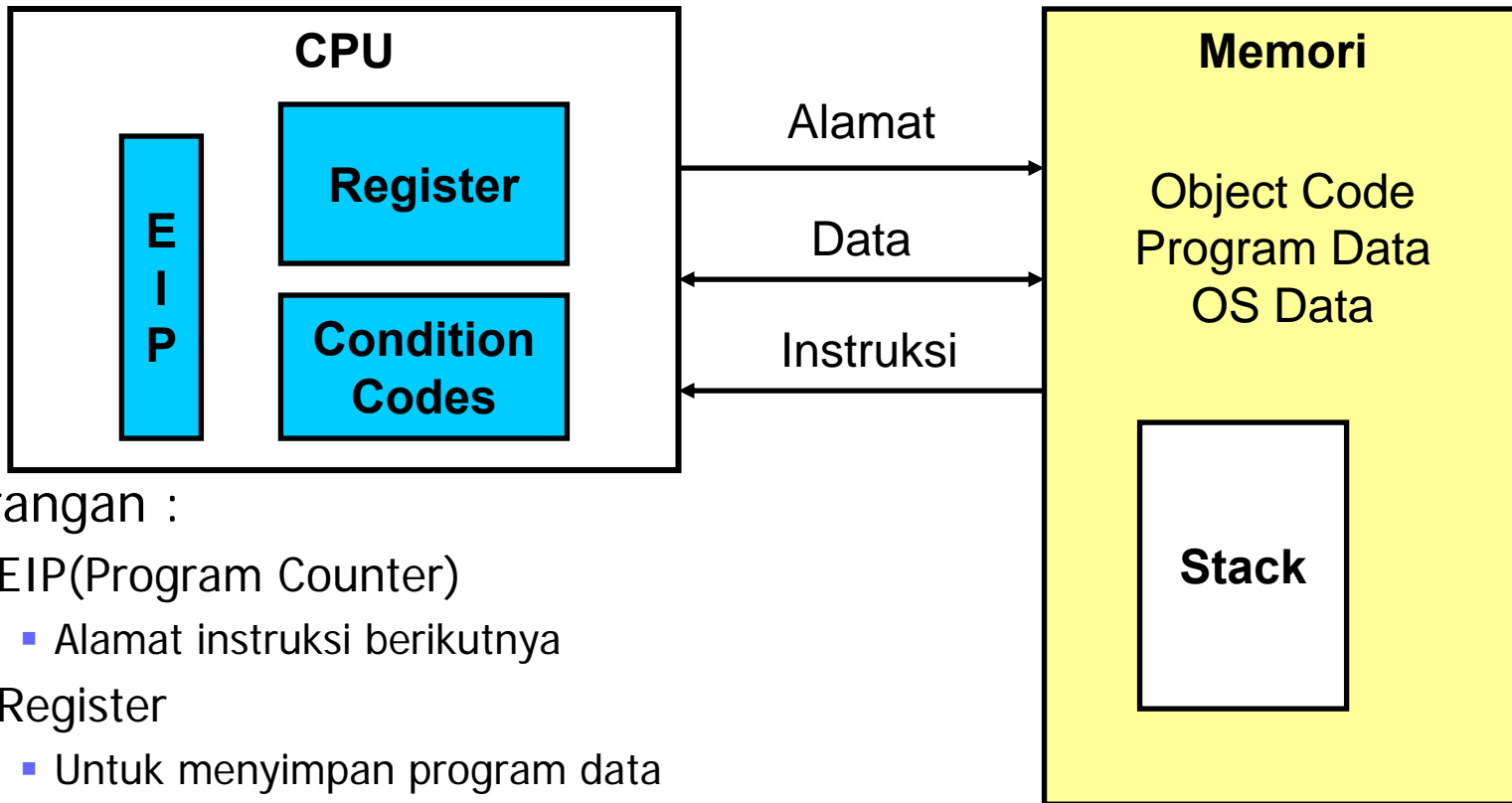
✚ Itanium (2001; 10 juta transistor)

- Arsitektur 64 bit
- Instruksi set baru yang sangat berbeda, dirancang untuk kinerja tinggi
- Dapat menjalankan program IA32 yang telah ada
 - Memiliki x86 engine on-board
- Proyek kerjasama dengan Hewlett-Packard

✚ Itanium 2 (2002; 221 juta transistor)

- Kinerja sangat tinggi

Pemrograman Assembly

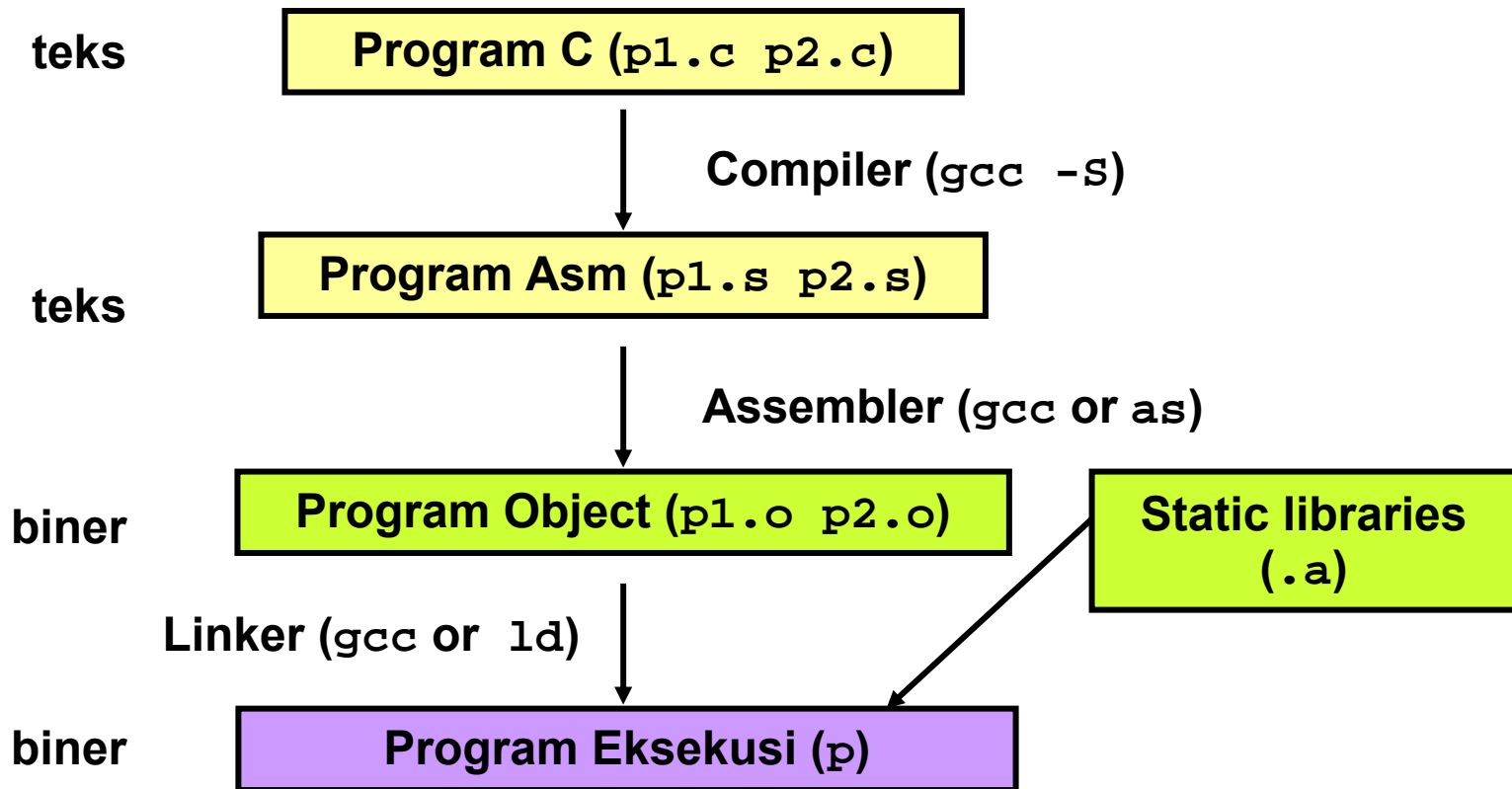


Keterangan :

- EIP(Program Counter)
 - Alamat instruksi berikutnya
- Register
 - Untuk menyimpan program data
- Condition Codes
 - Menyimpan informasi status dari operasi aritmatika
 - Digunakan pada percabangan
- Memori
 - Byte addressable array
 - Menyimpan kode, data, sistem operasi
 - Terdapat stack yang digunakan pada prosedur

Menerjemahkan C Ke Kode Object

- Kode disimpan dalam file : `p1.c p2.c`
- Dikompilasi dengan perintah : `gcc -O p1.c p2.c -o p`
 - Menggunakan optimasi (`-O`). Hasil biner disimpan dalam file `p`



Kompilasi ke Assembly

Kode C

```
int sum(int x, int y)
{
    int t = x+y;
    return t;
}
```

Assembly diperoleh

```
_sum:
    pushl %ebp
    movl %esp,%ebp
    movl 12(%ebp),%eax
    addl 8(%ebp),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

Diperoleh dengan perintah

```
gcc -O -S code.c
```

File yang dihasilkan `code.s`

Karakteristik Assembly

+ Tipe Data Minimal

- Data integer 1,2 atau 4 byte
 - Data dan alamat (pointer)
- Data floating point 4, 8 atau 10 byte
- Tidak ada tipe khusus untuk array atau structure
 - Hanya alokasi byte berurutan dalam memori

+ Operasi Dasar

- Melakukan fungsi aritmatika pada register atau memori
- Transfer data antara memori dan register
 - Load data dari memori ke register
 - Store isi register ke memori
- Transfer kontrol
 - Unconditional jump ke/dari prosedur
 - Percabangan conditional

Kode Object

Kode sum

0x401040 <sum> :

0x55

0x89

0xe5

0x8b

0x45

0x0c

0x03

0x45

0x08

0x89

0xec

0x5d

0xc3

- **Total 13 byte**
- **Masing2x instruksi tdd 1, 2, or 3 byte**
- **Dimulai dari alamat 0x401040**



Assembler

- Menerjemahkan .s menjadi .o
- Membuat kode biner setiap instruksi
- Merupakan kode eksekusi yang hampir lengkap
- Belum terdapat hubungan (link) dengan kode dari file berbeda



Linker

- Mengelola referensi antar file
- Menggabungkan dengan static run-time libraries
 - Mis., kode untuk malloc, printf
- Beberapa library merupakan *dynamically linked*
 - Link terjadi ketika program mulai dieksekusi

Contoh Instruksi Mesin

```
int t = x+y;
```

```
addl 8(%ebp),%eax
```

**Sama dengan
ekspresi
x += y**

```
0x401046:    03 45 08
```

+ Kode C

- Menjumlahkan dua signed integer

+ Assembly

- Menjumlahkan 2 bil integer 4 byte
 - Memiliki instruksi yang sama untuk bilangan signed atau unsigned

Operand:

x: Register %eax

y: Memory M[%ebp+8]

t: Register %eax

- Return value dalam %eax

+ Kode Object

- 3 byte instruksi
- Disimpan pada alamat 0x401046

Disassembling Kode Object

Disassembled

```
00401040 <_sum>:
  0:      55          push   %ebp
  1:      89 e5       mov    %esp,%ebp
  3:      8b 45 0c    mov    0xc(%ebp),%eax
  6:      03 45 08    add   0x8(%ebp),%eax
  9:      89 ec       mov    %ebp,%esp
  b:      5d          pop    %ebp
  c:      c3          ret
  d:      8d 76 00   lea   0x0(%esi),%esi
```

Disassembler

objdump -d p

- Perangkat untuk mempelajari kode object
- Menganalisis pola-pola bit dari suatu urutan instruksi
- Menghasilkan perkiraan dari kode assembly
- Dapat dijalankan pada file .out (file eksekusi) atau file .o

Yang dapat di-Disassembly

```
% objdump -d WINWORD.EXE

WINWORD.EXE:      file format pei-i386

No symbols in "WINWORD.EXE".
Disassembly of section .text:

30001000 <.text>:
30001000:  55                push    %ebp
30001001:  8b ec            mov     %esp,%ebp
30001003:  6a ff            push   $0xffffffff
30001005:  68 90 10 00 30   push   $0x30001090
3000100a:  68 91 dc 4c 30   push   $0x304cdc91
```

- Semua yang dapat diinterpretasikan sebagai kode eksekusi
- Disassembler mempelajari byte demi byte dan merekonstruksi kode assembly

Pengaksesan Data dan Informasi CPU IA32

Format Data

Deklarasi C	Tipe data	GAS suffix	Ukuran (byte)
char	byte	b	1
short	word	w	2
int	double word	l	4
unsigned	double word	l	4
long int	double word	l	4
unsigned long	double word	l	4
char *	double word	l	4
float	single precision	s	4
double	double precision	l	8
long double	extended precision	t	12

✚ Pada instruksi GAS assembler terdapat satu karakter (suffix) yang menentukan ukuran operand

✚ Mis. instruksi `mov` memiliki tiga suffix :

- `movb` (move byte)
- `movw` (move word)
- `movl` (move double word)

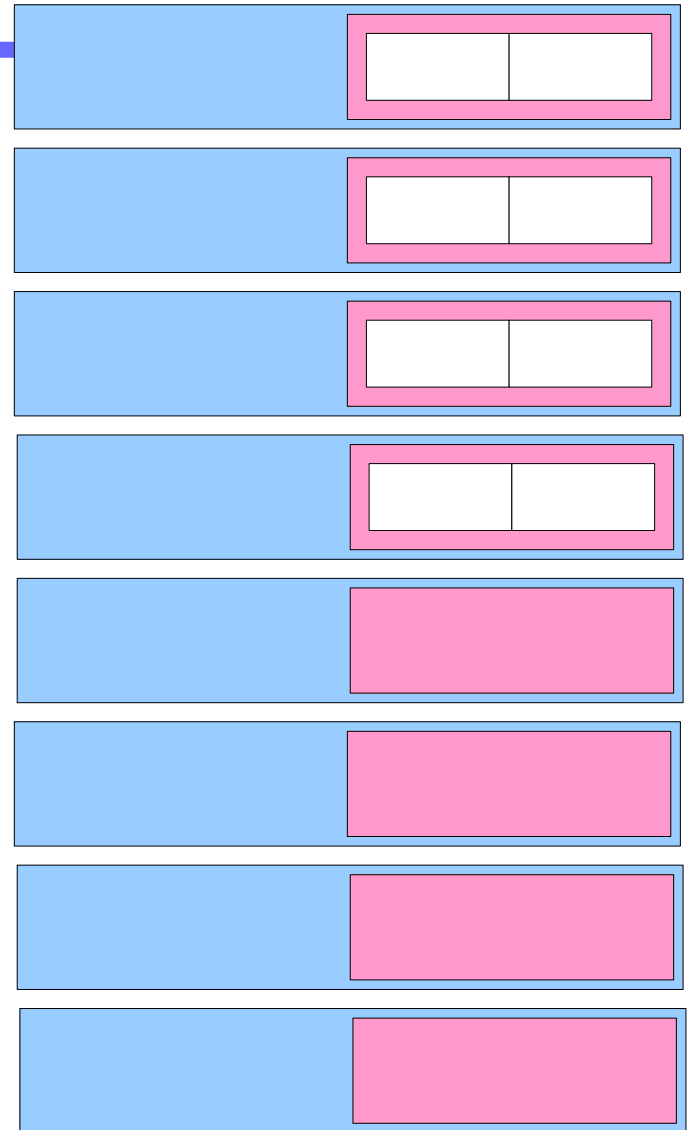
✚ Instruksi `mov` digunakan untuk memindahkan data (*move data*)

Pada CPU Intel, istilah "word" digunakan untuk data 16 bit

- Data 32 bit dikatakan "double word"

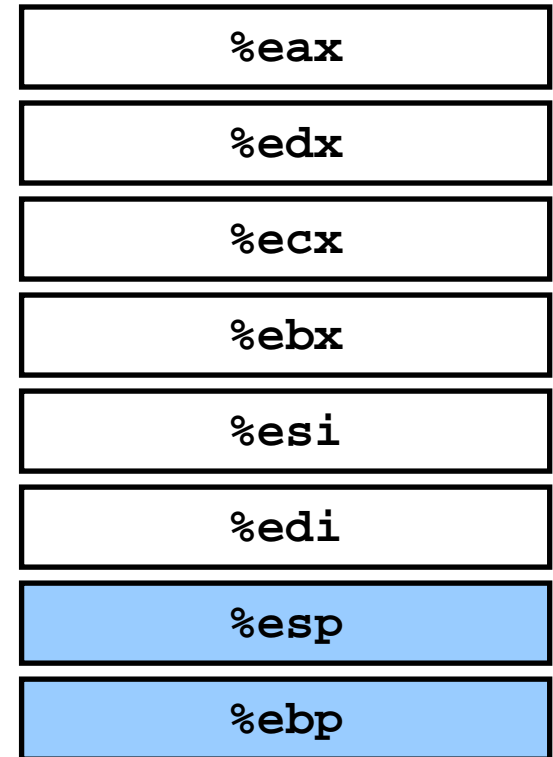
Register Integer

- ✦ CPU IA32 memiliki delapan register 32 bit untuk menyimpan integer dan pointer
- ✦ Register dapat diakses secara 16 bit (word) atau 32 bit (double word)
- ✦ Empat register pertama dapat diakses secara 8 bit
- ✦ Register :
 - `%eax`, `%ecx`, `%edx`, `%ebx`, `%esi`, `%edi` adalah *general purpose register*
 - `%esp` untuk menyimpan *stack pointer*
 - `%ebp` untuk menyimpan *base pointer*
 - `%eax`, `%ecx`, `%edx` memiliki aturan *save* dan *restore* yang berbeda dengan `%ebx`, `%esi`, `%edi`



Instruksi Perpindahan Data

- ✚ Instruksi : `MOV1 Asal, Tujuan;`
 - Memindahkan 4 byte data
- ✚ Jenis-jenis operasi
 - Immediate : Data integer konstan
 - Seperti konstanta C, memakai prefiks \$
 - Contoh : `$0x400` , `$-533`
 - Kode dalam 1, 2 atau 4 byte
 - Register : Satu dari 8 register integer
 - `%esp` dan `%ebp` disiapkan untuk penggunaan khusus
 - Lainnya digunakan untuk instruksi tertentu
 - Memori : 4 byte berurutan pada memori
 - Terdapat berbagai mode pengalamatan



Kombinasi Operand Instruksi **movl**

Inst	Asal	Tujuan	Contoh instruksi	Penjelasan
movl	Imm	Reg	movl \$0x4, %eax	%eax = 0x4
		Mem	movl \$-147, (%eax)	M(%eax) = M(0x4) = -147
	Reg	Reg	movl %eax, %edx	%edx = %eax = 0x4
		Mem	movl %eax, (%edx)	M(%edx) = M(0x4) = 0x4
	Mem	Reg	movl (%eax), %edx	%edx = M(%eax) = 0x4

- Tidak dapat melakukan transfer memori ke memori dalam satu instruksi

Register	
%eax	0x4
%edx	0x4

Memori	
Alamat	Isi
0x4	-147
0x4	0x4

Mode Pengalamatan Sederhana

✚ Indirect

- Bentuk Umum : $(R) \text{ Mem}[\text{Reg}[R]]$
- Register R berisi alamat memori

`movl (%ecx), %eax` → $\%eax = M(\%ecx)$

✚ Displacement

- Bentuk Umum : $D(R) \text{ Mem}[\text{Reg}[R]+D]$
- Register R berisi awal dari area memori
- Konstanta D menentukan offset-nya

`movl 8(%ebp), %edx` → $\%edx = M(8+\%ebp)$

Mode Pengalamatan Sederhana

Contoh Program : Swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

swap:

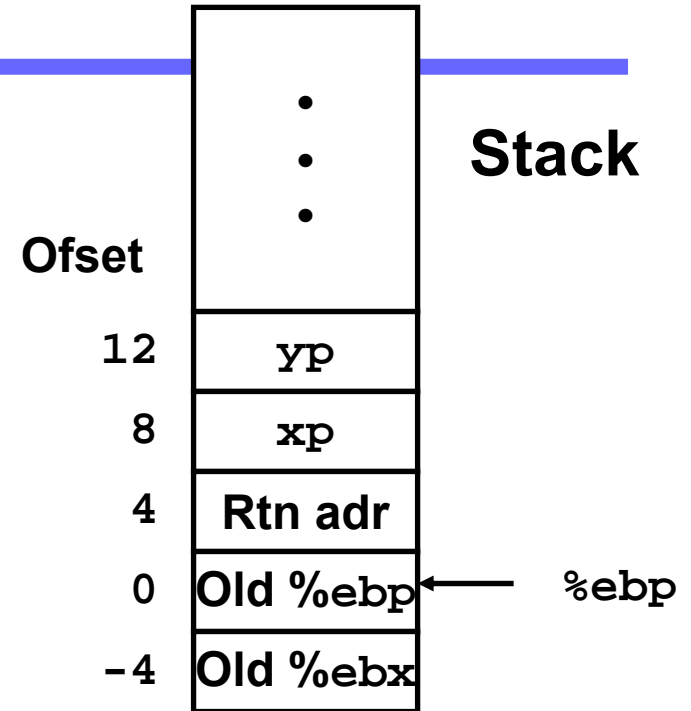
```
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
} Set Up

    movl 12(%ebp),%ecx
    movl 8(%ebp),%edx
    movl (%ecx),%eax
    movl (%edx),%ebx
    movl %eax,(%edx)
    movl %ebx,(%ecx)
} Body

    movl -4(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
} Finish
```

Program Swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```



Register	Variabel
----------	----------

%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

```
movl 12(%ebp),%ecx # ecx = yp
movl 8(%ebp),%edx # edx = xp
movl (%ecx),%eax # eax = *yp (t1)
movl (%edx),%ebx # ebx = *xp (t0)
movl %eax,(%edx) # *xp = eax
movl %ebx,(%ecx) # *yp = ebx
```

Program Swap

Alamat

%eax	
%edx	
%ecx	
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
	YP	12	0x120
	xp	8	0x124
		4	Rtn adr
	%ebp	→ 0	0x104
		-4	0x100

```

movl 12(%ebp),%ecx # ecx = yp
movl 8(%ebp),%edx  # edx = xp
movl (%ecx),%eax   # eax = *yp (t1)
movl (%edx),%ebx   # ebx = *xp (t0)
movl %eax,(%edx)   # *xp = eax
movl %ebx,(%ecx)   # *yp = ebx
    
```

Program Swap

Alamat

%eax	
%edx	
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
	YP	12	0x120
	xp	8	0x124
		4	Rtn adr
	%ebp	0	
		-4	
			0x108
			0x104
			0x100

```

movl 12(%ebp),%ecx # ecx = yp
movl 8(%ebp),%edx # edx = xp
movl (%ecx),%eax # eax = *yp (t1)
movl (%edx),%ebx # ebx = *xp (t0)
movl %eax,(%edx) # *xp = eax
movl %ebx,(%ecx) # *yp = ebx
    
```

Program Swap

Alamat

%eax	
%edx	0x124
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
	YP	12	0x120
	xp	8	0x124
		4	Rtn adr
	%ebp	→ 0	0x104
		-4	0x100

```

movl 12(%ebp),%ecx # ecx = yp
movl 8(%ebp),%edx # edx = xp
movl (%ecx),%eax # eax = *yp (t1)
movl (%edx),%ebx # ebx = *xp (t0)
movl %eax,(%edx) # *xp = eax
movl %ebx,(%ecx) # *yp = ebx
    
```

Program Swap

Alamat

%eax	456
%edx	0x124
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
YP	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp	→ 0		0x104
	-4		0x100

```

movl 12(%ebp),%ecx # ecx = yp
movl 8(%ebp),%edx # edx = xp
movl (%ecx),%eax # eax = *yp (t1)
movl (%edx),%ebx # ebx = *xp (t0)
movl %eax,(%edx) # *xp = eax
movl %ebx,(%ecx) # *yp = ebx
    
```

Program Swap

Alamat

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
YP	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp	→ 0		0x104
	-4		0x100

```

movl 12(%ebp),%ecx # ecx = yp
movl 8(%ebp),%edx # edx = xp
movl (%ecx),%eax # eax = *yp (t1)
movl (%edx),%ebx # ebx = *xp (t0)
movl %eax,(%edx) # *xp = eax
movl %ebx,(%ecx) # *yp = ebx
    
```

Program Swap

Alamat

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

		456	0x124
		456	0x120
			0x11c
			0x118
			0x114
	YP	12	0x120
	XP	8	0x124
		4	Rtn adr
%ebp	→	0	
		-4	
			0x104
			0x100

```

movl 12(%ebp),%ecx # ecx = yp
movl 8(%ebp),%edx # edx = xp
movl (%ecx),%eax # eax = *yp (t1)
movl (%edx),%ebx # ebx = *xp (t0)
movl %eax, (%edx) # *xp = eax
movl %ebx, (%ecx) # *yp = ebx
    
```

Program Swap

Alamat

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

		456	0x124
		123	0x120
			0x11c
			0x118
			0x114
	YP	12	0x120
	xp	8	0x124
		4	Rtn adr
	%ebp	0	0x104
		-4	0x100

```

movl 12(%ebp),%ecx # ecx = yp
movl 8(%ebp),%edx # edx = xp
movl (%ecx),%eax # eax = *yp (t1)
movl (%edx),%ebx # ebx = *xp (t0)
movl %eax,(%edx) # *xp = eax
movl %ebx,(%ecx) # *yp = ebx
    
```

Mode Pengalamatan Berindeks

✚ Bentuk umum :

$$\mathbf{D(Rb,Ri,S) \quad Mem[Reg[Rb] + S * Reg[Ri] + D]}$$

- D: Kontanta "displacement" 1, 2, atau 4 byte
- Rb: Base register: Salah satu dari 8 register integer
- Ri: Index register: Salah satu register, kecuali %esp
- S: Skala : 1, 2, 4, or 8

✚ Kasus khusus :

- (Rb,Ri) $Mem[Reg[Rb] + Reg[Ri]]$
- D(Rb,Ri) $Mem[Reg[Rb] + Reg[Ri] + D]$
- (Rb,Ri,S) $Mem[Reg[Rb] + S * Reg[Ri]]$

Contoh Perhitungan Alamat

<code>%edx</code>	<code>0xf000</code>
-------------------	---------------------

<code>%ecx</code>	<code>0x100</code>
-------------------	--------------------

Ekspresi	Perhitungan	Alamat	Bentuk
<code>0x8(%edx)</code>	<code>0xf000 + 0x8</code>	<code>0xf008</code>	D(Rb)
<code>(%edx,%ecx)</code>	<code>0xf000 + 0x100</code>	<code>0xf100</code>	(Rb,Ri)
<code>(%edx,%ecx,4)</code>	<code>0xf000 + 4*0x100</code>	<code>0xf400</code>	(Rb,Ri,S)
<code>0x80(,%edx,2)</code>	<code>2*0xf000 + 0x80</code>	<code>0x1e080</code>	D(Rb,Ri,S)

Instruksi Perhitungan Alamat

✚ Instruksi :

`leal Asal, Tujuan`

- *Asal* adalah ekspresi mode alamat
- Men-set *Tujuan* menjadi alamat yang ditentukan oleh ekspresi tersebut

Mis. Isi register `%edx = x`, maka instruksi

`leal 7(%edx, %edx, 4), %eax` berarti $\%eax = x + 4x + 7 = 5x + 7$

✚ Penggunaan

1. Menghitung alamat tanpa melakukan referensi memori
 - Contoh: `p = &x[i];`
2. Menghitung ekspresi aritmatika
 - Contoh : `z = x + k*y`

Ekspresi Aritmatika `leal`

```
int arith
(int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```

Register	Variabel
<code>%edx</code>	<code>y</code>
<code>%eax</code>	<code>x</code>

`arith:`

```
    pushl %ebp
    movl  %esp,%ebp
```

```
    movl  8(%ebp),%eax
    movl  12(%ebp),%edx
    leal  (%edx,%eax),%ecx
    leal  (%edx,%edx,2),%edx
    sall  $4,%edx
    addl  16(%ebp),%ecx
    leal  4(%edx,%eax),%eax
    imull %ecx,%eax
```

```
    movl  %ebp,%esp
    popl  %ebp
    ret
```

} Set Up

} Body

} Finish

Beberapa Operasi Aritmatika

+ Instruksi dengan dua operand

<code>addl</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan + Asal$
<code>subl</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan - Asal$
<code>imull</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan * Asal$
<code>sall</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan \ll Asal$ (disebut juga <code>shll</code>)
<code>sarl</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan \gg Asal$ (Operasi aritmatika)
<code>shrl</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan \gg Asal$ (Operasi logika)
<code>xorl</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan \wedge Asal$
<code>andl</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan \& Asal$
<code>orl</code>	<i>Asal, Tujuan</i>	$Tujuan = Tujuan Asal$

Beberapa Operasi Aritmatika

Instruksi dengan satu operand

`incl Tujuan` $Tujuan = Tujuan + 1$

`decl Tujuan` $Tujuan = Tujuan - 1$

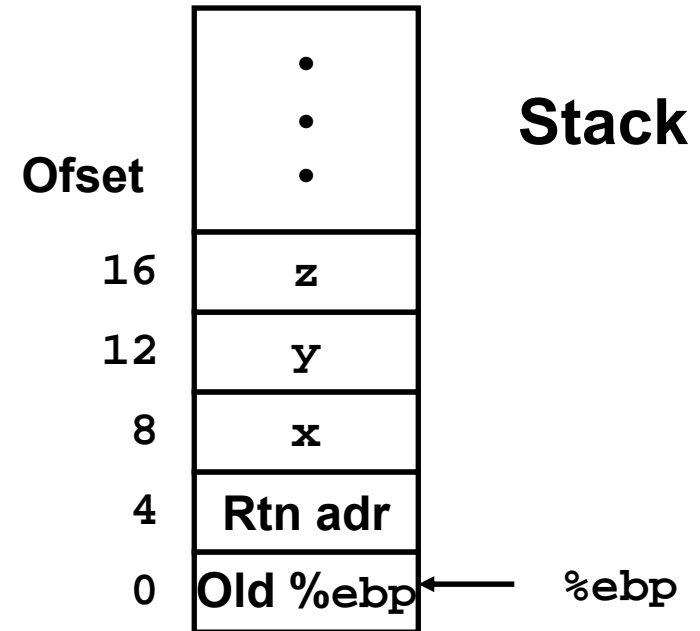
`negl Tujuan` $Tujuan = - Tujuan$

`notl Tujuan` $Tujuan = \sim Tujuan$

Penjelasan Fungsi `arith`

```
int arith
  (int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```

```
    movl 8(%ebp),%eax      # eax = x
    movl 12(%ebp),%edx     # edx = y
    leal (%edx,%eax),%ecx  # ecx = x+y (t1)
    leal (%edx,%edx,2),%edx # edx = 3*y
    sall $4,%edx          # edx = 48*y (t4)
    addl 16(%ebp),%ecx     # ecx = z+t1 (t2)
    leal 4(%edx,%eax),%eax # eax = 4+t4+x (t5)
    imull %ecx,%eax       # eax = t5*t2 (rval)
```



Penjelasan Fungsi `arith`

```
int arith
  (int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```

```
# eax = x
movl 8(%ebp),%eax
# edx = y
movl 12(%ebp),%edx
# ecx = x+y (t1)
leal (%edx,%eax),%ecx
# edx = 3*y
leal (%edx,%edx,2),%edx
# edx = 48*y (t4)
sall $4,%edx
# ecx = z+t1 (t2)
addl 16(%ebp),%ecx
# eax = 4+t4+x (t5)
leal 4(%edx,%eax),%eax
# eax = t5*t2 (rval)
imull %ecx,%eax
```

Contoh Lain

```
int logical(int x, int y)
{
    int t1 = x^y;
    int t2 = t1 >> 17;
    int mask = (1<<13) - 7;
    int rval = t2 & mask;
    return rval;
}
```

$2^{13} = 8192$, $2^{13} - 7 = 8185$

```
movl 8(%ebp),%eax
xorl 12(%ebp),%eax
sarl $17,%eax
andl $8185,%eax
```

logical:

```
pushl %ebp
movl %esp,%ebp
```

} Set Up

```
movl 8(%ebp),%eax
xorl 12(%ebp),%eax
sarl $17,%eax
andl $8185,%eax
```

} Body

```
movl %ebp,%esp
popl %ebp
ret
```

} Finish

```
eax = x
eax = x^y      (t1)
eax = t1>>17  (t2)
eax = t2 & 8185
```

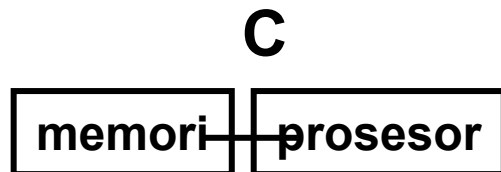
Ringkasan

Sifat-sifat CISC

- ✚ Instruksi dapat mereferensikan jenis-jenis operand yang berbeda
 - Immediate, register, memory
- ✚ Operasi aritmatika dapat membaca atau menulis ke memori
- ✚ Komputasi untuk mereferensikan memori dapat dilakukan secara kompleks
 - $Rb + S * Ri + D$
 - Dapat digunakan juga pada ekspresi matematika
- ✚ Suatu instruksi dapat memiliki panjang yang berbeda-beda
 - Instruksi IA32 dapat berkisar antara 1 hingga 15 byte

Ringkasan

Model Mesin



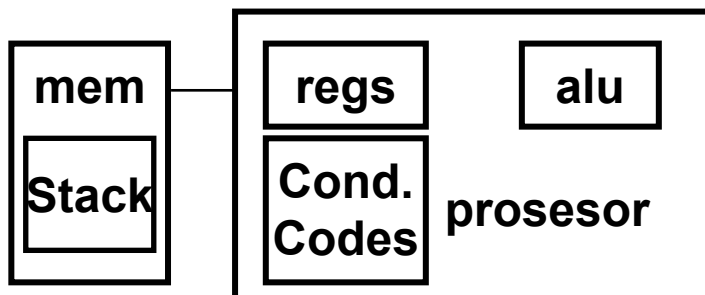
Data

- 1) char
- 2) int, float
- 3) double
- 4) struct, array
- 5) pointer

Kontrol

- 1) loops
- 2) conditionals
- 3) switch
- 4) Proc. call
- 5) Proc. return

Assembly



- | | |
|-------------------------------|----------------|
| 1) byte | 1) branch/jump |
| 2) 2-byte word | 2) call |
| 3) 4-byte long word | 3) ret |
| 4) contiguous byte allocation | |
| 5) address of initial byte | |