

Organisasi Sistem Komputer

Bab 5. Percabangan

5.1 Percabangan

5.2 Loop



Pembahasan:

- Percabangan Tidak Berkondisi
- Percabangan Berkondisi

Memodifikasi Alur Instruksi

- CPU secara default mengeksekusi instruksi secara berurutan
- CPU melihat register `EIP` untuk mengetahui alamat dari instruksi yang akan dieksekusi
- Untuk mengubah alur instruksi, kita perlu mengubah isi `EIP`
- Kita tidak mengubah langsung, namun menggunakan instruksi-instruksi yang tersedia untuk percabangan
- Dua tipe percabangan:
 - **Percabangan Tidak Berkondisi (unconditional)**: Alur program berpindah ke suatu alamat instruksi tertentu tanpa kondisi
 - **Percabangan Berkondisi (conditional)**: Alur program berpindah jika suatu kondisi tertentu terpenuhi (statemen if-else pada bahasa high-level)

Percabangan Tidak Berkondisi

- Instruksi `JMP` digunakan untuk lompat ke suatu label kode
- Sintaks:

`JMP destination`

- Destination berupa label kode yang menjadi tujuan lompatan (**ingat**: label kode hanyalah referensi ke alamat instruksi)
- Instruksi `JMP` mengubah isi dari Instruction Pointer (EIP) dengan alamat dari label kode

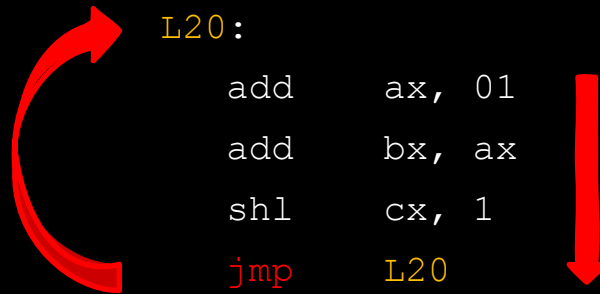
```
...  
mov    eax, [data]  
add    eax, ebx  
jmp    here  
sub    al, bl  
movsx  ax, al  
here:  
call   print_int
```

Kedua instruksi ini tidak akan pernah dieksekusi

Contoh Instruksi JMP

- Lihat contoh berikut:

```
...  
    mov     ax, 00  
    mov     bx, 00  
    mov     cx, 01  
  
L20:  
    add     ax, 01  
    add     bx, ax  
    shl     cx, 1  
    jmp     L20
```



- Contoh di atas adalah loop yang tidak akan pernah berakhir (infinite loop)

Percabangan Berkondisi

- Untuk melakukan percabangan kondisi, umumnya kita membentuk kondisi dengan dua instruksi: **instruksi perbandingan** dan **instruksi lompat kondisional**
- Salah satu instruksi perbandingan: instruksi `CMP` (compare)
- Instruksi-instruksi untuk lompat kondisional: instruksi `Jxx` dimana `xx` adalah variasi dari instruksi lompat berdasarkan hasil instruksi `CMP`

Instruksi CMP

- Instruksi CMP (compare) membandingkan nilai dari dua operand dan men-set register EFLAGS berdasarkan hasilnya
- Sintaks:

CMP a, b

- Instruksi CMP bekerja mirip seperti instruksi SUB yaitu menghitung $a-b$ namun tidak menyimpan hasilnya
- Sama seperti instruksi SUB, instruksi CMP men-set nilai-nilai EFLAGS berdasarkan hasil pengurangan:
 - Komparasi antar dua bilangan tidak bertanda: ZF (Zero Flag) dan CF (Carry Flag)
 - Komparasi antar dua bilangan bertanda: ZF (Zero Flag), SF (Sign Flag), dan OF (Overflow Flag)
- Nilai-nilai flag yang diset oleh CMP digunakan oleh instruksi lompat kondisional



Instruksi Jxx

- Lompat berdasarkan hasil dari instruksi CMP
- Sintaks:

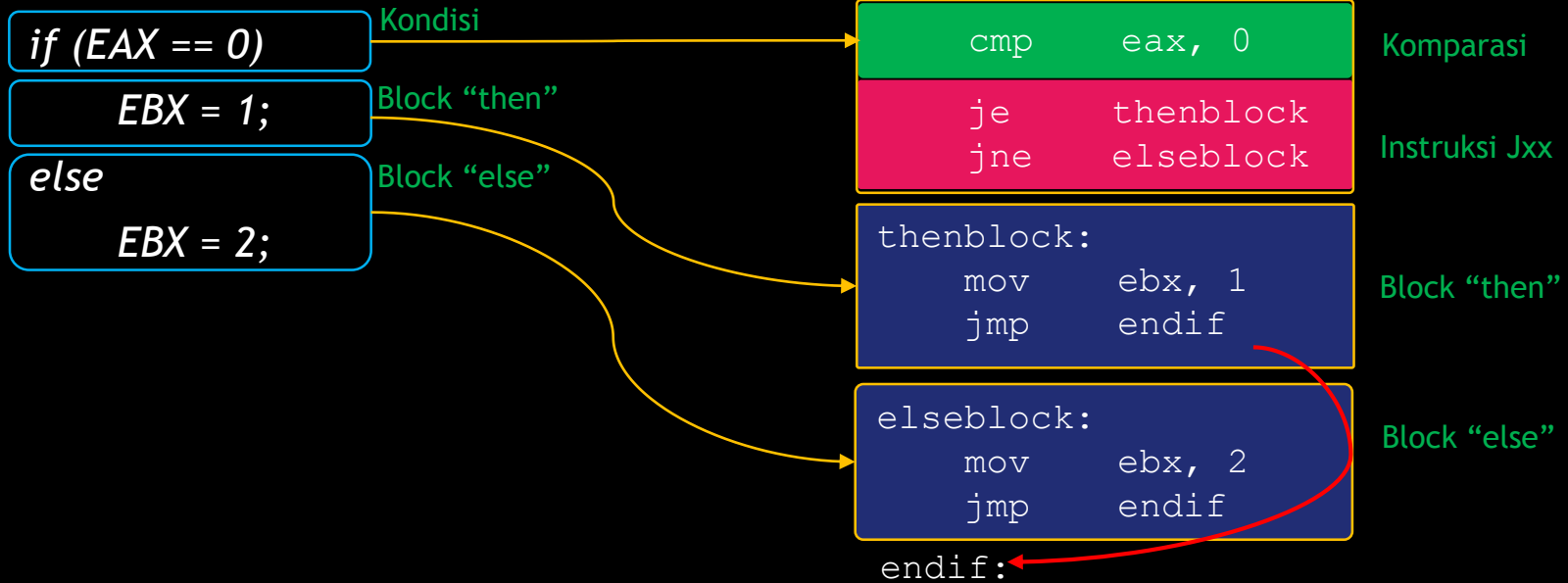
Jxx *destination*

xx adalah varian sesuai tabel

CMP a, b				
Kondisi	a dan b tidak bertanda		a dan b bertanda	
	Jxx	Keterangan	Jxx	Keterangan
a = b	JE	Jump if Equal	JE	Jump if Equal
a != b	JNE	Jump if Not Equal	JNE	Jump if Not Equal
a < b	JB	Jump if Below	JL	Jump if Less
	JNAE	Jump if Not Above or Equal	JNGE	Jump if Not Greater or Equal
a <= b	JBE	Jump if Below or Equal	JLE	Jump if Less or Equal
	JNA	Jump if Not Above	JNG	Jump if Not Greater
a > b	JA	Jump if Above	JG	Jump if Greater
	JNBE	Jump if Not Below or Equal	JNLE	Jump if Not Less or Equal
a >= b	JAE	Jump if Above or Equal	JGE	Jump if Greater or Equal
	JNB	Jump if Not Below	JNL	Jump if Not Less

Contoh Percabangan Berkondisi

Pseudocode



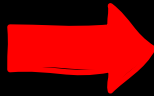
Contoh Percabangan Berkondisi

```
cmp    eax, 0
jne   thenblock
jne   elseblock

thenblock:
    mov    ebx, 1
    jmp    endif

elseblock:
    mov    ebx, 2
jmp    endif

endif:
```



```
cmp    eax, 0
jne     elseblock ; lompat jika tdk terpenuhi

; kode untuk block then
mov     ebx, 1
jmp     endif

; block else
elseblock:
    mov     ebx, 2

endif:
```

Struktur if-then

Pseudocode:

```
if (condition) then  
    then_block;
```

Assembly:

```
; instruksi untuk men-set flag (spt. cmp ...)  
...  
jxx    endif    ; xx sehingga lompat dilakukan  
                ; jika kondisi tidak terpenuhi  
  
; kode untuk block then  
...  
endif:
```

Struktur if-then-else

Pseudocode:

```
if (condition) then
    then_block;
else
    else_block;
```

Assembly:

```
; instruksi untuk men-set flag (spt. cmp ...)
...

jxx    else_block    ; xx sehingga lompat dilakukan
                    ; jika kondisi tidak terpenuhi

; kode untuk block then
...

jmp    endif
else_block:
    ; kode untuk block else
    ...

endif:
```



Demo Percabangan Berkondisi

- Demo 5.1.1: Lebih dari 100 ?
- Demo 5.1.2: Genap atau Ganjil ?



Jxx Berdasarkan Nilai EFLAGS

- Terdapat instruksi J_{xx} yang lebih generik: lompat dilakukan berdasarkan nilai flag yang ter-set

Mnemonic	Keterangan	Bercabang jika
JZ	Jump if Zero	ZF = 1
JNZ	Jump if Not Zero	ZF = 0
JO	Jump if Overflow	OF = 1
JNO	Jump if Not Overflow	OF = 0
JS	Jump if Signed	SF = 1
JNS	Jump if Not Signed	SF = 0
JC	Jump if Carry	CF = 1
JNC	Jump if Not Carry	CF = 0
JP	Jump if Parity	PF = 1
JNP	Jump if Not Parity	PF = 0

Contoh Jxx Generik

- Penjumlahan dua bilangan 8-bit, tampilkan pesan overflow jika hasil penjumlahan melebihi 8-bit dan tampilkan hasil jika tidak overflow

```
.segment data
    bil_1      db 255
    bil_2      db 1
    pesan_of   db "Hasil penjumlahan overflow!", 0

.segment text

    mov     al, [bil_1]
    add     al, [bil_2]

    jc      else_block    ; jika tidak overflow ke block else
                        ; Pada penjumlahan tidak bertanda overflow
                        ; ditandai dengan flag carry yang ter-set
                        ; sehingga kita gunakan jc (jump if carry)

    ; kode untuk block then
    movzx   eax, al        ; zero extend al ke eax
    call    print_int

    jmp     endif

else_block:
    ; kode untuk block else
    mov     eax, pesan_of
    call    print_string

endif:
```

Demo Jxx Generik

- Demo 5.1.3: Apakah hasil jumlah 8 bit overflow ?

