

LẬP TRÌNH HỆ THỐNG

ThS. Đỗ Thị Thu Hiền
(hiendtt@uit.edu.vn)



TRƯỜNG ĐH CÔNG NGHỆ THÔNG TIN - ĐHQG-HCM
KHOA MẠNG MÁY TÍNH & TRUYỀN THÔNG
FACULTY OF COMPUTER NETWORK AND COMMUNICATIONS

Tầng 8 - Tòa nhà E, trường ĐH Công nghệ Thông tin, ĐHQG-HCM
Điện thoại: (08)3 725 1993 (122)

Machine-level programming: Điều khiển luồng



Làm thế nào biểu diễn trong assembly?

Code C

```
if (x>y)
```

```
    result = x + y;
```

```
else
```

```
    result = x - y;
```

Assembly code

// lệnh if để kiểm tra điều kiện trong assembly??

→ addl %ebx, %eax

// else??

→ subl %ebx, %eax



```
for(i=0; i<8; i++)  
    result += i;
```

// Lệnh for??

và... ??



Ví dụ if/else trong assembly

Code C

```
int result;  
if (x < y)  
    result = y-x;  
else  
    result = x-y;  
return result;
```

Assembly code

x at %ebp+8, y at %ebp+12

1	movl	8(%ebp), %edx
2	movl	12(%ebp), %eax
3	cmpl	%eax, %edx
4	jge	.L2
5	subl	%edx, %eax
6	jmp	.L3
7	.L2:	
8	subl	%eax, %edx
9	movl	%edx, %eax
10	.L3:	

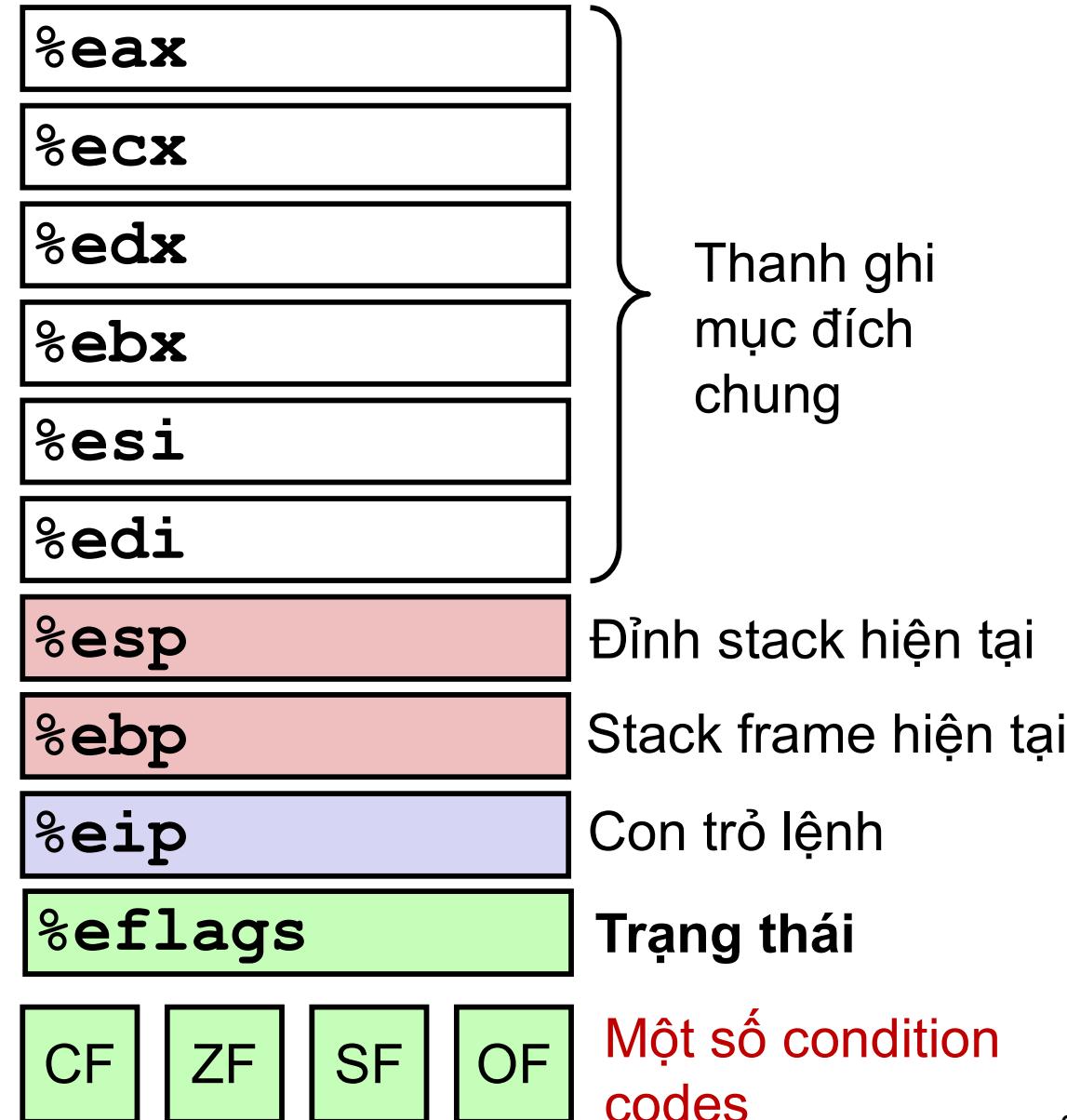
Nội dung

- Điều khiển luồng: Condition codes
- Rẽ nhánh có điều kiện
- Vòng lặp

Trạng thái bộ xử lý (IA32)

■ Các thông tin về chương trình hiện đang thực thi

- Dữ liệu tạm thời (`%eax`, ...)
- Vị trí của stack trong lúc chạy (`%esp`, `%ebp`)
- Vị trí kiểm soát câu lệnh được thực thi (`%eip`, ...)
- Trạng thái của một số test gần nhất (CF, ZF, SF, OF)



Trạng thái bộ xử lý (x86-64)?

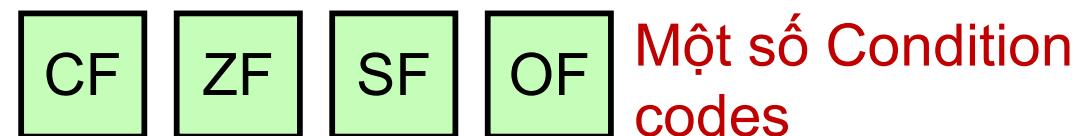
■ Các thông tin về chương trình hiện đang thực thi

- Dữ liệu tạm thời (`%rax`, ...)
- Vị trí của stack trong lúc chạy (`%rsp`)
- Vị trí kiểm soát câu lệnh được thực thi (`%rip`, ...)
- Trạng thái của một số test gần nhất (CF, ZF, SF, OF)

Đỉnh stack hiện tại

Registers

<code>%rax</code>	<code>%r8</code>
<code>%rbx</code>	<code>%r9</code>
<code>%rcx</code>	<code>%r10</code>
<code>%rdx</code>	<code>%r11</code>
<code>%rsi</code>	<code>%r12</code>
<code>%rdi</code>	<code>%r13</code>
<code>%rsp</code>	<code>%r14</code>
<code>%rbp</code>	<code>%r15</code>
<code>%rip</code>	Con trả lệnh
<code>%rflags</code>	Trạng thái



Condition Codes

■ Các “thanh ghi” 1-bit (0 hoặc 1)

- **CF** - Carry Flag (for unsigned) Được bật khi xảy ra tràn số không dấu
- **SF** - Sign Flag (for signed) Được bật khi kết quả là số âm
- **ZF** - Zero Flag Được bật khi kết quả là số 0
- **OF** Overflow Flag (for signed) Được bật khi xảy ra tràn số có dấu

■ Chứa trong thanh ghi `%eflag` / `%rfflag`

■ Gán giá trị cho các condition codes

- Gán ngầm: qua các phép tính toán học
- Gán tường minh: các lệnh so sánh, test

■ Condition codes có thể được dùng để:

- Thực thi các đoạn lệnh dựa trên các điều kiện
- Gán giá trị dựa trên điều kiện
- Chuyển dữ liệu dựa trên các điều kiện

Gán giá trị Condition Codes (1)

Gán ngầm qua phép tính toán học

■ Các “thanh ghi” 1-bit

- | | | | |
|------|---------------------------|----|----------------------------|
| ▪ CF | Carry Flag (for unsigned) | SF | Sign Flag (for signed) |
| ▪ ZF | Zero Flag | OF | Overflow Flag (for signed) |

■ Được gán ngầm bằng các phép tính toán học

- Có thể được xem là tác dụng phụ (side affect) của các phép toán này

Ví dụ: **addl** Src, Dest $\leftrightarrow t = a+b$

CF được gán nếu có nhón bit ở most significant bit (tràn số không dấu)

ZF được gán nếu $t == 0$

SF được gán nếu $t < 0$ (có dấu)

OF được gán nếu tràn số bù 2 (có dấu)

$(a>0 \ \&\& \ b>0 \ \&\& \ t<0) \ || \ (a<0 \ \&\& \ b<0 \ \&\& \ t>=0)$

■ Không được gán giá trị bằng lệnh **leal** !

Gán giá trị Condition Codes (2)

Gán tương minh qua phép so sánh

■ Giá trị được gán tương minh bằng các lệnh So sánh

- **cmpl Src2, Src1**
- **cmpl b, a** tương tự như tính $a - b$ mà không cần lưu lại kết quả tính
- CF được gán nếu có nhón bit ở most significant bit (dùng cho so sánh số không dấu)
- ZF được gán nếu $a == b$
- SF được gán nếu $(a-b) < 0$ (phép trừ có dấu - âm)
- OF được gán nếu tràn số bù 2 (có dấu)
 $(a>0 \&\& b<0 \&\& (a-b)<0) \parallel (a<0 \&\& b>0 \&\& (a-b)>0)$

Gán giá trị Condition Codes (3)

Gán tương minh qua lệnh test

■ Gán tương minh bằng lệnh test

- **testl Src2, Src1**
 - **testl b, a** tương tự tính $a \& b$ mà không lưu lại kết quả tính
- Gán giá trị các condition codes dựa trên giá trị của **Src1 & Src2**
- Hữu ích khi có 1 toán hạng đóng vai trò là mask
- ZF được gán khi $a \& b == 0$
- SF được gán khi $a \& b < 0$



Sử dụng Condition Codes

Điều khiển luồng dựa trên điều kiện

■ Gán giá trị dựa trên điều kiện

- set~~X~~

■ Chuyển dữ liệu dựa trên điều kiện

- Conditional move

■ Rẽ nhánh có điều kiện

- Instruction rẽ nhánh: j~~X~~

- If/else

- Vòng lặp (loop)

Nội dung

- Điều khiển luồng: Condition codes
- Rẽ nhánh có điều kiện
- Vòng lặp

Các câu lệnh jump

■ Các lệnh rẽ nhánh: jX

- **jX** label
- Nhảy đến đoạn mã khác (được gán nhãn label) để thực thi dựa trên các condition codes.

jX	Điều kiện	Mô tả
jmp	1	Nhảy không điều kiện
je	ZF	Equal / Zero
jne	$\sim ZF$	Not Equal / Not Zero
js	SF	Negative
jns	$\sim SF$	Nonnegative
jg	$\sim (SF \wedge OF) \ \& \ \sim ZF$	Greater (Signed)
jge	$\sim (SF \wedge OF)$	Greater or Equal (Signed)
jl	$(SF \wedge OF)$	Less (Signed)
jle	$(SF \wedge OF) \mid ZF$	Less or Equal (Signed)
ja	$\sim CF \ \& \ \sim ZF$	Above (unsigned)
jb	CF	Below (unsigned)

Các câu lệnh jump kết hợp với so sánh

- Các lệnh jump thường kết hợp với các lệnh so sánh/test
 - Kết quả của lệnh so sánh/test quyết định có thực hiện jump hay không.

`cmpl src2, src1`

`jX label`

jX	Điều kiện nhảy
je	src1 == src2
jne	src1 != src2
jg	src1 > src2
jge	src1 ≥ src2
jl	src1 < src2
jle	src1 ≤ src2

Sử dụng lệnh jX nào?

- Cho các giá trị: $\%eax = x$ $\%ebx = y$ $\%ecx = z$
- Một đoạn mã gán nhãn .L1

Điều kiện nhảy đến .L1	Tổ hợp lệnh cmpl/test và jX	
$x == y$	<code>cmpl %eax, %ebx je .L1</code>	<code>cmpl %ebx, %eax je .L1</code>
$y != z$	<code>cmpl %ebx, %ecx jne .L1</code>	<code>cmpl %ecx, %ebx jne .L1</code>
$z > x$	<code>cmpl %eax, %ecx jg .L1</code>	<code>cmpl %ecx, %eax jl .L1</code>
$x < 0$	<code>cmpl \$0, %eax jl .L1</code>	<code>cmpl \$0, %eax js .L1</code>
$y == 0$	<code>cmpl \$0, %ebx je .L1</code>	<code>test %ebx, %ebx jz .L1</code>
z	<code>cmpl \$0, %ecx jne .L1</code>	<code>test %ecx, %ecx jnz .L1</code>
$true$	<code>jmp .L1</code>	

Rẽ nhánh có điều kiện – Ví dụ

```
int absdiff(int x, int y)
{
    int result;
    if (x < y)
        result = y-x;
    else
        result = x-y;
    return result;
}
```

x at %ebp+8, y at %ebp+12

1	movl	8(%ebp), %edx	/x
2	movl	12(%ebp), %eax	/y
3	cmpl	%eax, %edx	
4	jge	.L2	
5	subl	%edx, %eax	
6	jmp	.L3	
7	.L2: # x >= y		
8	subl	%eax, %edx	
9	movl	%edx, %eax	
10	.L3:		

Sử dụng điều kiện nhảy là điều kiện **false** của if

Rẽ nhánh có điều kiện – Ví dụ (tt)

```
int absdiff(int x, int y)
{
    int result;
    if (x < y)
        result = y-x;
    else
        result = x-y;
    return result;
}
```

x at %ebp+8, y at %ebp+12

1.	movl	8(%ebp), %edx	/x
2.	movl	12(%ebp), %eax	/y
3.	cmpl	%eax, %edx	
4.	j1	.L2	
5.	subl	%eax, %edx	
6.	movl	%edx, %eax	
7.	jmp	.L3	
8.	.L2:		
9.	subl	%edx, %eax	
10..L3			

Sử dụng điều kiện nhảy là điều kiện **true** của if

Chuyển mã rẽ nhánh có điều kiện

Từ C sang assembly: Dạng Goto

- C hỗ trợ **goto** statement → bản chất giống lệnh **jmp**
- Nhảy đến vị trí xác định bởi **label**

```
int absdiff(int x, int y)
{
    int result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```
int absdiff_j(int x, int y)
{
    int result;
    int ntest = x <= y;
    if (ntest) goto Else;
    result = x-y;
    goto Done;
Else:
    result = y-x;
Done:
    return result;
}
```

Chuyển mã rẽ nhánh có điều kiện

Từ C sang assembly: Phương pháp chung

C code

```
if (test-expr)
    then-statement;
else
    else-statement;
```

Dạng Goto (*thực hiện tính toán và luồng tương tự mã assembly*)

```
nt = !test-expr;
if (nt)
    goto False;
then-statement;
goto Done;
False:
else-statement;
Done:
```

Assembly code

```
...
<instructions to check nt>
jX False
<instructions of then-statement>
jmp Done
False:
<instruction of else-statement>
Done:
...
```

Chuyển mã rẽ nhánh có điều kiện

Từ C sang assembly: Phương pháp chung

C code

```
if (a>b)
    result = a^b;
else
    result = a&b;
```

Dạng Goto (*thực hiện tính toán và luồng tương tự mã assembly*)

```
nt = a <= b;
if (nt)
    goto False;
result = a^b;
goto Done;
False:
    result = a&b;
Done:
```

Assembly code

```
%eax = a
%ebx = b
```

```
...
cmpl %ebx, %eax
jle False
xorl %ebx, %eax
jmp Done
False:
    andl %ebx, %eax
```

```
Done:
```

```
// return value in %eax
```

Chuyển mã rẽ nhánh có điều kiện

Ví dụ 2.1: if/else - Từ C sang assembly

```
1 int func(int x, int y)
2 {
3     int result = 0;
4     if (x > 2)
5         result = x + y;
6     else
7         result = x - y;
8     return result;
9 }
```

```
// x at %ebp+8, y at %ebp+12
1.    movl    $0, -4(%ebp) //result
2.    cmpl    $2, 8(%ebp)
3.    jle     .L2
4.    movl    8(%ebp), %eax //x
5.    addl    12(%ebp), %eax //x+y
6.    movl    %eax, -4(%ebp)
7.    jmp     .L3
8. .L2:
9.    movl    8(%ebp), %eax //x
10.   subl   12(%ebp), %eax //x-y
11.   movl    %eax, -4(%ebp)
12. .L3:
```

Dạng Goto (thực hiện tính toán và luồng tương tự mã assembly)

```
1. int func(int x, int y)
2. {
3.     int result = 0;
4.     not_true = x <= 2 ;
5.     if (not_true)
6.         goto False;
7.     result = x + y;
8.     goto Done;
9. False:
10.    result = x - y;
11. Done:
12. }
```

Chuyển mã rẽ nhánh có điều kiện

Ví dụ 2.2: if/else - Từ C sang assembly

```
1 int func(int x, int y) {  
2     int sum = 0;  
3     if (x != 0)  
4         y--;  
5     sum = x + y;  
6     return sum;  
7 }
```

```
// x at %ebp+8, y at %ebp+12  
1.     movl $0,-4(%ebp) //sum  
2.     cmpl $0,8(%ebp)  
3.     je .False  
4.     subl $1,12(%ebp)  
5.     jmp .Done  
6. .False:  
7. .Done:  
8.     movl 8(%ebp),%eax  
9.     addl 12(%ebp),%eax  
10.    movl %eax,-4(%ebp)  
11.    movl -4(%ebp),%eax //return
```

Dạng Goto (*thực hiện tính toán và luồng tương tự mã assembly*)

```
1. int func(int x, int y)  
2. {  
3.     int sum = 0;  
4.     not_true = x == 0 ;  
5.     if (not_true)  
6.         goto False;  
7.     y--;  
8.     goto Done;  
9. False:  
10. Done:  
11.     sum = x + y;  
12. }
```

Chuyển mã rẽ nhánh có điều kiện

Ví dụ 3: Nested if - Từ C sang assembly

C code

```
1 int func(int x, int y)
2 {
3     int result = 0;
4     if (x)    if #1
5     {
6         if (y > 1) if #2
7             result = x + y;
8         else
9             result = x * y;
10    }
11 }
```

Goto code

```
1. int func(int x, int y)
2. {
3.     int result = 0;
4.     notif1 = x == 0      ;
5.     if (notif1)
6.         goto F1;
7.     notif2 = y <= 1;
8.     if (notif2)
9.         goto F2;
10.    result = x + y;
11.    goto Done;
12.    F2:
13.        result = x * y;
14.    F1:
15.    Done:
16. }
```

Chuyển mã rẽ nhánh có điều kiện

Ví dụ 3: Nested if - Từ C sang assembly

C code

```
1 int func(int x, int y)
2 {
3     int result = 0;
4     if (x)    if #1
5     {
6         if (y > 1) if #2
7             result = x + y;
8         else
9             result = x * y;
10    }
11 }
```

Assembly code

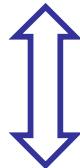
```
// x at %ebp+8, y at %ebp+12
1.     movl    $0, -4(%ebp)  #result
2.     cmpl    $0, 8(%ebp)
3.     je      .L2          # if 1
4.     cmpl    $1, 12(%ebp)
5.     jle     .L3          # if 2
6.     movl    8(%ebp), %eax
7.     addl    12(%ebp), %eax
8.     movl    %eax, -4(%ebp)
9.     jmp     .L4
10. .L3: //false của if #2
11.     movl    8(%ebp), %eax
12.     imull   12(%ebp), %eax
13.     movl    %eax, -4(%ebp)
14. .L4:
15. .L2: //false của if #1
...
```

Chuyển mã rẽ nhánh có điều kiện

Ví dụ 4: if/else - Từ C sang assembly

C code

```
1 int func(int x, int y)
2 {
3     int result = 0;
4     if (y && x != y)
5         result = x + y;
6     return result;
7 }
```



```
1 int func(int x, int y)
2 {
3     int result = 0;
4     if (y)
5         if (x != y)
6             result = x + y;
7     return result;
8 }
```

Viết Assembly code tương ứng?

Biết giá trị trả về sẽ lưu trong thanh ghi %eax

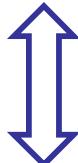
```
// x at %ebp+8, y at %ebp+12
1.    movl    $0, -4(%ebp) //result
2.    cmpl    $0, 12(%ebp)
3.    je     .L2
4.    movl    12(%ebp), %edx
5.    cmpl    8(%ebp), %edx
6.    je     .L2
7.    addl    8(%ebp), %edx
8.    movl    %edx, -4(%ebp)
9. .L2:
10.   movl   -4(%ebp), %eax
```

Chuyển mã rẽ nhánh có điều kiện

Ví dụ 5: if/else - Từ C sang assembly

C code

```
1. int arith(int a, int b, int c)
2. {
3.     int sum = 0;
4.     if(c < 0 || a == b)
5.         sum = (a & b)^c;
6.     return sum;
7. }
```



```
1. int arith(int a, int b, int c)
2. {
3.     int sum = 0;
4.     if (c < 0)
5.         sum = (a & b)^c;
6.     else if (a == b)
7.         sum = (a & b)^c;
8.     return sum;
9. }
```

Viết **Code Goto** và **Assembly** tương ứng?

Biết giá trị trả về sẽ lưu trong thanh ghi %eax

```
// a at %ebp+8, b at %ebp+12, c at %ebp+16
1.    movl    $0, -4(%ebp) //sum
2.    cmpl    $0, 16(%ebp)
3.    jge     .L2
4. .L1:
5.    movl    8(%ebp), %edx
6.    andl    12(%ebp), %edx
7.    xorl    16(%ebp), %edx
8.    movl    %edx, -4(%ebp)
9.    jmp    .L3
10. .L2: //so sánh tiếp a và b
11.    movl    8(%ebp), %edx
12.    cmpl    12(%ebp), %edx
13.    jne     .L3
14.    jmp    .L1
15. .L3:
16.    movl    -4(%ebp), %eax
```

Chuyển mã rẽ nhánh có điều kiện

Ví dụ 1: if/else - Từ assembly sang C

Assembly code

```
x at ebp+8, y at ebp+12, sum at eax
1.      movl 8(%ebp),%ecx    //x
2.      movl 12(%ebp),%ebx   //y
3.      cmpl $0,%ecx
4.      jle .L2
5.      leal (%ecx,%ebx),%eax
6.      jmp .L3
7. .L2:
8.      movl %ebx,%eax
9.      subl %ecx,%eax
10..L3:
```

```
int sum(int x, int y)
{
    if (x > 0)
        sum = x + y;
    else
        sum = y - x;
}
```

Dự đoán Code C?

- Điều kiện true của if:
 $x > 0$
- Đoạn code tương ứng với điều kiện true của if?
Dòng code 5:
 $sum = x + y;$
- Đoạn code tương ứng với false của if?
Dòng code 8-9:
 $sum = y;$
 $sum = sum - x;$
hay
 $sum = y - x;$

Chuyển mã rẽ nhánh có điều kiện

Ví dụ 2: if/else - Từ assembly sang C

Assembly code

```
x at ebp+8, y at ebp+12, sum at ebp-4
1.    movl 8(%ebp),%eax
2.    cmpl 12(%ebp),%eax
3.    jg   .L1
4.    addl 12(%ebp),%eax
5.    movl %eax,-4(%ebp)
6. .L1:
7.    incl -4(%ebp)
```

Dự đoán Code C?

```
int sum(int x, int y)
{
    if (x <= y)
        sum = x + y;
    sum++;
}
```