

Agenda

Branching in assembly

Preliminaries

cmp

test

jmp

goto

Conditional statements

if

cmov

Loops

Preliminaries: Making comparisons

Two types of comparisons between registers

- Subtraction (cmp)
- And (test)

Use flags to check the results of the test

- Is the result equal to zero? (ZF)
- Is the result negative? (SF)
- Did overflow occur? Used for signed comparisons (OF)
- Did arithmetic carry occur? Used for unsigned comparisons (CF)

x86_64 comparisons

Table 38. Conditional Control Instructions

Instruction	Translation
<code>cmp R1, R2</code>	Compares R2 with R1 (i.e., evaluates $R2 - R1$)
<code>test R1, R2</code>	Computes $R1 \& R2$

Table 39. Common Condition Code Flags.

Flag	Translation
ZF	Is equal to zero (1: yes; 0: no)
SF	Is negative (1: yes; 0: no)
OF	Overflow has occurred (1:yes; 0: no)
CF	Arithmetic carry has occurred (1: yes; 0:no)

NOTE: `test %rax, %rax` is often used to check whether a value is 0

Exercise: Making comparisons

What flags would be set by the following comparisons if

- `%rax` has the value 5
- `%edi` has the value -3
- `%esi` has the value 11

```
cmp $0, %rax
```

```
cmp %esi, %edi
```

```
test %rax, %esi
```

```
test %rax, %rax
```

Jump instructions

Idea: Instructions execute in sequence unless we **jump** or call a function.

Jump instructions set the program counter based on a condition. If the condition is false, we go to the next instruction. If it is true, we **goto** a different instruction.

Table 40. Direct Jump Instructions

Instruction	Description
<code>jmp L</code>	Jump to location specified by L
<code>jmp *addr</code>	Jump to specified address

Example: Jump instructions

jle 0x8048427 <main+28>

Table 41. Conditional Jump Instructions; Synonyms Shown in Parentheses

Signed Comparison	Unsigned Comparison	Description
je (jz)		jump if equal (==) or jump if zero
jne (jnz)		jump if not equal (!=)
js		jump if negative
jns		jump if non-negative
jg (jnl)	ja (jnbe)	jump if greater (>)
jge (jnl)	jae (jnb)	jump if greater than or equal (>=)
jl (jnge)	jb (jnae)	jump if less (<)
jle (jng)	jbe (jna)	jump if less than or equal (<=)

Goto statements

Idea: Goto statements jump to the instruction that matches a label

WARNING: Using goto statements in high-level code is a faux pas.

Example: Goto Example

```
int getSmallest(int x, int y) {  
    int smallest;  
    if ( x > y ) { //if (conditional)  
        smallest = y; //then statement  
    }  
    else {  
        smallest = x; //else statement  
    }  
    return smallest;  
}
```

```
int getSmallest(int x, int y) {  
    int smallest;  
  
    if (x <= y ) { //if (!conditional)  
        goto else_statement;  
    }  
    smallest = y; //then statement  
    goto done;  
  
else_statement:  
    smallest = x; //else statement  
  
done:  
    return smallest;  
}
```

Example: If statements

```
int getSmallest(int x, int y) {  
    int smallest;  
    if ( x > y ) {  
        smallest = y;  
    }  
    else {  
        smallest = x;  
    }  
    return smallest;  
}
```

```
(gdb) disas getSmallest  
Dump of assembler code for function getSmallest:  
0x40059a <+4>: mov  %edi,-0x14(%rbp)  
0x40059d <+7>: mov  %esi,-0x18(%rbp)  
0x4005a0 <+10>: mov  -0x14(%rbp),%eax  
0x4005a3 <+13>: cmp  -0x18(%rbp),%eax  
0x4005a6 <+16>: jle  0x4005b0 <getSmallest+26>  
0x4005a8 <+18>: mov  -0x18(%rbp),%eax  
0x4005ae <+24>: jmp  0x4005b9 <getSmallest+35>  
0x4005b0 <+26>: mov  -0x14(%rbp),%eax  
0x4005b9 <+35>: pop  %rbp  
0x4005ba <+36>: retq
```

Exercise: What does each instruction mean?

```
1: mov  %edi,-0x14(%rbp)
2: mov  %esi,-0x18(%rbp)
3: mov  -0x14(%rbp),%eax
4: cmp  -0x18(%rbp),%eax
5: jle  0x4005b0 <getSmallest+26>
6: mov  -0x18(%rbp),%eax
7: jmp  0x4005b9 <getSmallest+35>
8: mov  -0x14(%rbp),%eax
9: pop  %rbp
10: retq
```

Exercise: What does each instruction mean?

```
1: mov  %edi,-0x14(%rbp)
2: mov  %esi,-0x18(%rbp)
3: mov  -0x14(%rbp),%eax
4: cmp  -0x18(%rbp),%eax
5: jle  0x4005b0 <getSmallest+26>
6: mov  -0x18(%rbp),%eax
7: jmp  0x4005b9 <getSmallest+35>
8: mov  -0x14(%rbp),%eax
9: pop  %rbp
10: retq
```

Example: Tracing

```
int getSmallest(int x, int y) {  
    int smallest;  
    if ( x > y ) { smallest = y; }  
    else { smallest = x; }  
    return smallest;  
}
```

```
0x40059a <+4>: mov  %edi,-0x14(%rbp)  
0x40059d <+7>: mov  %esi,-0x18(%rbp)  
0x4005a0 <+10>: mov  -0x14(%rbp),%eax  
0x4005a3 <+13>: cmp  -0x18(%rbp),%eax  
0x4005a6 <+16>: jle  0x4005b0 <getSmallest+26>  
0x4005a8 <+18>: mov  -0x18(%rbp),%eax  
0x4005ae <+24>: jmp  0x4005b9 <getSmallest+35>  
0x4005b0 <+26>: mov  -0x14(%rbp),%eax  
0x4005b9 <+35>: pop  %rbp  
0x4005ba <+36>: retq
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Converting if statements to goto form

C if statement

```
if (condition) {  
    then_statement;  
}  
else {  
    else_statement;  
}
```

Compiler's equivalent goto form

```
if (!condition) {  
    goto else;  
}  
then_statement;  
goto done;  
else:  
    else_statement;  
done:
```

Exercise: Rewrite in goto form

```
int main() {  
    printf("Do you like jokes?\n");  
    char c;  
    scanf(" %c", &c);  
  
    if (c == 'Y') {  
        printf("Me too!\n");  
    }  
    else if (c == 'N') {  
        printf("You're weird\n");  
    }  
    else {  
        printf("I don't understand\n");  
    }  
}
```

cmove instructions

A **cmove (conditional move)** is an efficient way to set a value based on a condition.

Example: `min = a < b ? a : b;`

```
min = a < b ? a : b;
```



```
if (a < b) {  
    min = a;  
}  
else {  
    min = b;  
}
```

Branching is more expensive than sequential instructions. With `cmove`, we evaluate both cases and then assign one of them

Example: cmov

```
int getSmallest_cmov(int x, int y) {  
    return x > y ? y : x;  
}
```

```
0x4005d7 <+0>: push  %rbp      #save %rbp  
0x4005d8 <+1>: mov   %rsp,%rbp   #update %rbp  
0x4005db <+4>: mov   %edi,-0x4(%rbp) #copy x to %rbp-0x4  
0x4005de <+7>: mov   %esi,-0x8(%rbp) #copy y to %rbp-0x8  
0x4005e1 <+10>: mov   -0x8(%rbp),%eax #copy y to %eax  
0x4005e4 <+13>: cmp   %eax,-0x4(%rbp) #compare x and y  
0x4005e7 <+16>: cmovle -0x4(%rbp),%eax #if (x <=y) copy x to %eax  
0x4005eb <+20>: pop   %rbp      #restore %rbp  
0x4005ec <+21>: retq          #return %eax
```

cmov functions

Table 46. The cmov Instructions.

Signed	Unsigned	Description
<code>cmovz</code> (<code>cmovz</code>)		move if equal (==)
<code>cmovne</code> (<code>cmovnz</code>)		move if not equal (!=)
<code>cmovs</code>		move if negative
<code>cmovns</code>		move if non-negative
<code>cmovg</code> (<code>cmovnl</code>)	<code>cmova</code> (<code>cmovnbe</code>)	move if greater (>)
<code>cmovge</code> (<code>cmovnl</code>)	<code>cmovae</code> (<code>cmovnb</code>)	move if greater than or equal (>=)
<code>cmovl</code> (<code>cmovnge</code>)	<code>cmovb</code> (<code>cmovnae</code>)	move if less (<)
<code>cmovle</code> (<code>cmovng</code>)	<code>cmovbe</code> (<code>cmovna</code>)	move if less than or equal (<=)

Loops

Implemented using jump and goto statements, like if statements

```
int sumUp(int n) {  
    //initialize total and i  
    int total = 0;  
    int i = 1;  
  
    while (i <= n) { //while i is less than or equal to n  
        total += i; //add i to total  
        i++;      //increment i by 1  
    }  
    return total;  
}
```

Dump of assembler code for function sumUp:

```
0x400526 <+0>: push  %rbp  
0x400527 <+1>: mov   %rsp,%rbp  
0x40052a <+4>: mov   %edi,-0x14(%rbp)  
0x40052d <+7>: mov   $0x0,-0x8(%rbp)  
0x400534 <+14>: mov   $0x1,-0x4(%rbp)  
0x40053b <+21>: jmp   0x400547 <sumUp+33>  
0x40053d <+23>: mov   -0x4(%rbp),%eax  
0x400540 <+26>: add   %eax,-0x8(%rbp)  
0x400543 <+29>: add   $0x1,-0x4(%rbp)  
0x400547 <+33>: mov   -0x4(%rbp),%eax  
0x40054a <+36>: cmp   -0x14(%rbp),%eax  
0x40054d <+39>: jle   0x40053d <sumUp+23>  
0x40054f <+41>: mov   -0x8(%rbp),%eax  
0x400552 <+44>: pop   %rbp  
0x400553 <+45>: retq
```

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	

Example: Tracing a loop

```
0x40052a <+4>: mov  %edi,-0x14(%rbp)
0x40052d <+7>: mov  $0x0,-0x8(%rbp)
0x400534 <+14>: mov  $0x1,-0x4(%rbp)
0x40053b <+21>: jmp  0x400547 <sumUp+33>
0x40053d <+23>: mov  -0x4(%rbp),%eax
0x400540 <+26>: add  %eax,-0x8(%rbp)
0x400543 <+29>: add  $0x1,-0x4(%rbp)
0x400547 <+33>: mov  -0x4(%rbp),%eax
0x40054a <+36>: cmp  -0x14(%rbp),%eax
0x40054d <+39>: jle  0x40053d <sumUp+23>
0x40054f <+41>: mov  -0x8(%rbp),%eax
```

Memory

Address	Value

Registers

Register	Value
%eax	
%edi	
%esi	
%rip	