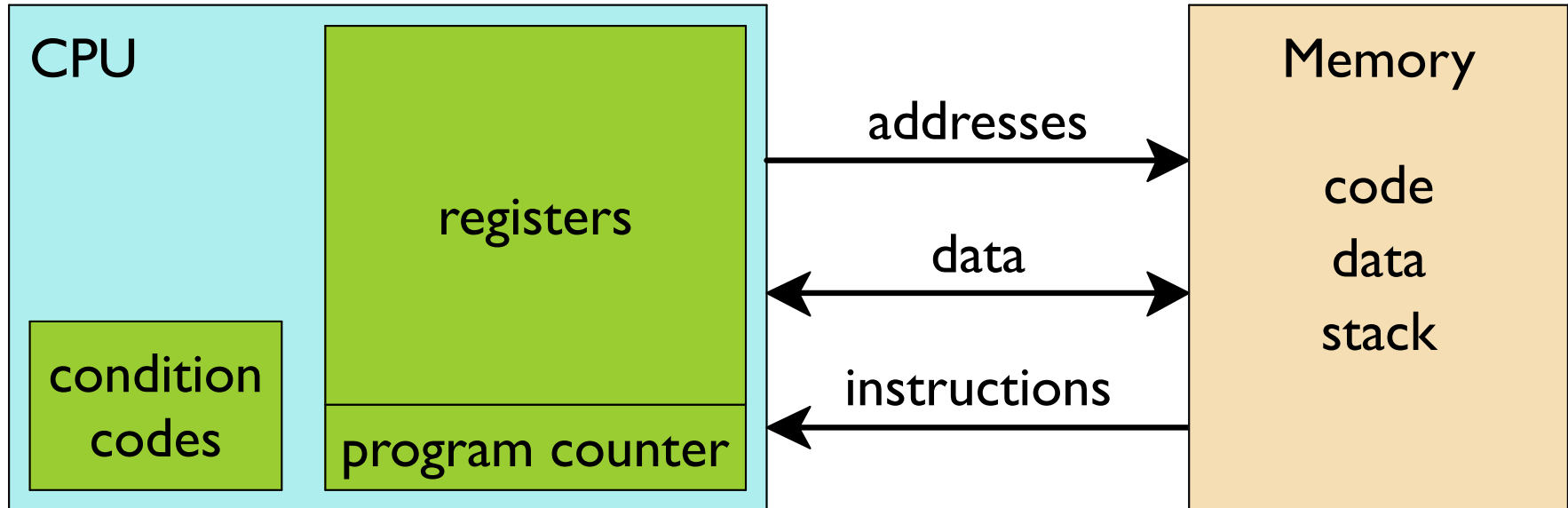
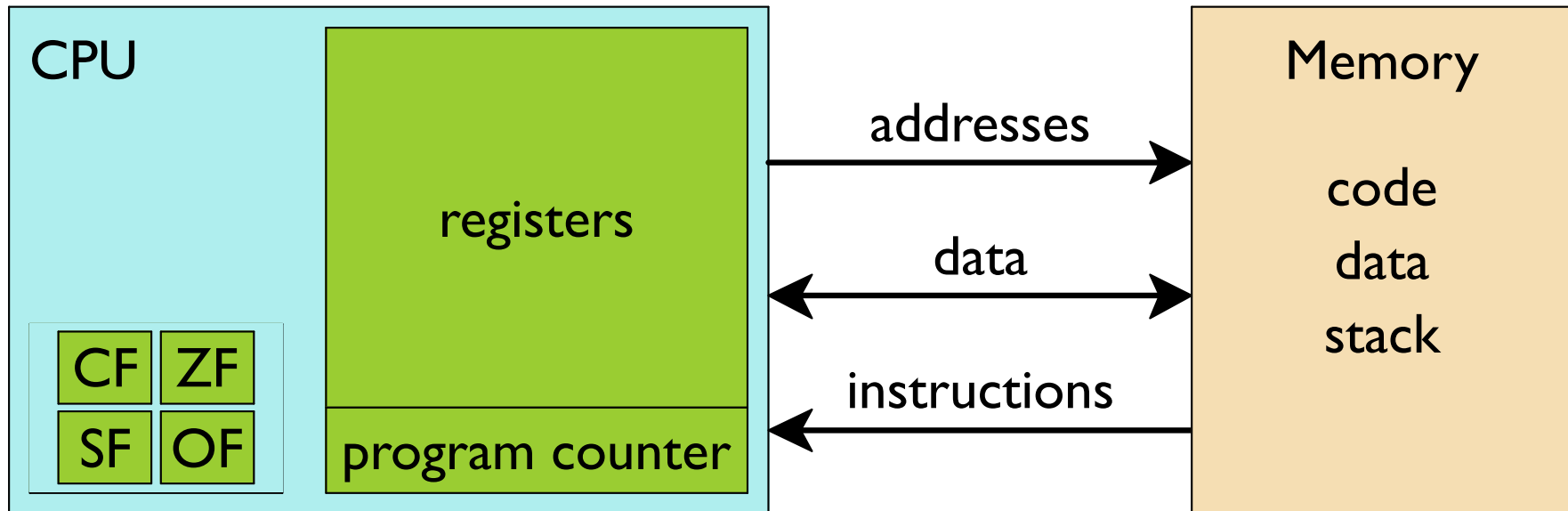


Condition Codes



Condition Codes



Set by most recent arithmetic (not counting `leax`):

- **CF**: carry — carry out of most-significant bit
- **ZF**: zero — produced zero
- **SF**: sign — produced negative
- **OF**: overflow — two's complement overflow

Condition Code Examples

CPU

register	value	unsigned	signed
<code>%eax</code>	<code>0x00000001</code>	1	1
<code>%ebx</code>	<code>0x00000002</code>	2	2
<code>%ecx</code>	<code>0x7FFFFFFF</code>	2147483647	2147483647
<code>%edx</code>	<code>0x80000001</code>	2147483649	-2147483647

```
subl %eax, %eax
```

Sets **ZF**
zero

Condition Code Examples

CPU

register	value	unsigned	signed
%eax	0x00000001	1	1
%ebx	0x00000002	2	2
%ecx	0x7FFFFFFF	2147483647	2147483647
%edx	0x80000001	2147483649	-2147483647

addl %eax, %ebx

Sets no flags

Condition Code Examples

CPU

register	value	unsigned	signed
%eax	0x00000001	1	1
%ebx	0x00000002	2	2
%ecx	0x7FFFFFFF	2147483647	2147483647
%edx	0x80000001	2147483649	-2147483647

subl %eax, %ebx **%ebx = %ebx - %eax**

Sets no flags

Condition Code Examples

CPU

register	value	unsigned	signed
%eax	0x00000001	1	1
%ebx	0x00000002	2	2
%ecx	0x7FFFFFFF	2147483647	2147483647
%edx	0x80000001	2147483649	-2147483647

subl %ebx, %eax **%eax = %eax - %ebx**

Sets **CF** **SF**
 carry sign

Condition Code Examples

CPU

register	value	unsigned	signed
<code>%eax</code>	<code>0x00000001</code>	1	1
<code>%ebx</code>	<code>0x00000002</code>	2	2
<code>%ecx</code>	<code>0x7FFFFFFF</code>	2147483647	2147483647
<code>%edx</code>	<code>0x80000001</code>	2147483649	-2147483647

```
addl %ecx, %edx
```

Sets **ZF** **CF**
zero carry

Condition Code Examples

CPU

register	value	unsigned	signed
<code>%eax</code>	<code>0x00000001</code>	1	1
<code>%ebx</code>	<code>0x00000002</code>	2	2
<code>%ecx</code>	<code>0x7FFFFFFF</code>	2147483647	2147483647
<code>%edx</code>	<code>0x80000001</code>	2147483649	-2147483647

`subl %ecx, %edx`

Sets **OF**
overflow

Condition Code Examples

CPU

register	value	unsigned	signed
<code>%eax</code>	<code>0x00000001</code>	1	1
<code>%ebx</code>	<code>0x00000002</code>	2	2
<code>%ecx</code>	<code>0x7FFFFFFF</code>	2147483647	2147483647
<code>%edx</code>	<code>0x80000001</code>	2147483649	-2147483647

```
subl %edx, %ecx
```

Sets **CF** **SF** **OF**
carry sign overflow

Condition Code Examples

CPU

register	value	unsigned	signed
%eax	0x00000001	1	1
%ebx	0x00000002	2	2
%ecx	0x7FFFFFFF	2147483647	2147483647
%edx	0x80000001	2147483649	-2147483647

subl *source, dest*

dest == *source* **ZF**

dest ≠ *source* **~ZF**

Condition Code Examples

CPU

register	value	unsigned	signed
%eax	0x00000001	1	1
%ebx	0x00000002	2	2
%ecx	0x7FFFFFFF	2147483647	2147483647
%edx	0x80000001	2147483649	-2147483647

subl *source, dest*

unsigned: $dest < source$ **CF**

$dest \leq source$ **CF | ZF**

signed: $dest < source$ **SF ^ OF**

$dest \leq source$ **(SF ^ OF) | ZF**

Condition Code Examples

CPU

register	value	unsigned	signed
<code>%eax</code>	<code>0x00000001</code>	1	1
<code>%ebx</code>	<code>0x00000002</code>	2	2
<code>%ecx</code>	<code>0x7FFFFFFF</code>	2147483647	2147483647
<code>%edx</code>	<code>0x80000001</code>	2147483649	-2147483647

```
subl %eax, %ebx
```

subl *source, dest*

Sets no flags

unsigned: *dest < source* **CF**

dest ≤ source **CF | ZF**

signed: *dest < source* **SF ^ OF**

dest ≤ source **(SF ^ OF) | ZF**

Condition Code Examples

CPU

register	value	unsigned	signed
%eax	0x00000001	1	1
%ebx	0x00000002	2	2
%ecx	0x7FFFFFFF	2147483647	2147483647
%edx	0x80000001	2147483649	-2147483647

```
subl %ebx, %eax
```

subl *source, dest*

Sets **CF** **SF**
carry sign

unsigned: *dest < source* **CF**

dest ≤ source **CF | ZF**

signed: *dest < source* **SF ^ OF**

dest ≤ source **(SF ^ OF) | ZF**

Condition Code Examples

CPU

register	value	unsigned	signed
<code>%eax</code>	<code>0x00000001</code>	1	1
<code>%ebx</code>	<code>0x00000002</code>	2	2
<code>%ecx</code>	<code>0x7FFFFFFF</code>	2147483647	2147483647
<code>%edx</code>	<code>0x80000001</code>	2147483649	-2147483647

```
subl %ecx, %edx
```

`subl source, dest`

unsigned: $dest < source$ **CF**

$dest \leq source$ **CF | ZF**

signed: $dest < source$ **SF ^ OF**

$dest \leq source$ **(SF ^ OF) | ZF**

Sets **OF**
overflow

Condition Code Examples

CPU

register	value	unsigned	signed
<code>%eax</code>	<code>0x00000001</code>	1	1
<code>%ebx</code>	<code>0x00000002</code>	2	2
<code>%ecx</code>	<code>0x7FFFFFFF</code>	2147483647	2147483647
<code>%edx</code>	<code>0x80000001</code>	2147483649	-2147483647

```
subl %edx, %ecx
```

`subl source, dest`

unsigned: $dest < source$ **CF**

$dest \leq source$ **CF | ZF**

signed: $dest < source$ **SF ^ OF**

$dest \leq source$ **(SF ^ OF) | ZF**

Sets **CF** **SF** **OF**
 carry sign overflow

Condition Codes and Comparisons

`setcc dest`

sets one byte in *dest* to 0 or 1

<code>sete</code>	equal / zero	ZF a.k.a. <code>setz</code>
<code>setne</code>	not equal / not zero	~ZF a.k.a. <code>setnz</code>
<code>sets</code>	negative	SF
<code>setns</code>	non-negative	~SF
<code>setg</code>	greater signed	~(SF^OF) & ~ZF
<code>setge</code>	greater or equal signed	~(SF^OF)
<code>setl</code>	less signed	(SF^OF)
<code>setle</code>	less or equal signed	(SF^OF) ZF
<code>seta</code>	above unsigned	~CF & ~ZF
<code>setb</code>	below unsigned	CF

Using `setcc`

```
int gt (long x, long y) {  
    return x > y;  
}
```

```
.... # %rdi = x  
.... # %rsi = y  
subq %rsi, %rdi # Compare x to y  
setg %al        # Set when >  
movzbl %al, %eax  
ret            # Result in %eax
```

`movzxx source, dest`

move with zero extension

Comparisons

```
cmpx source, dest           dest - source
```

```
cmpq source, dest
```

is the same as

```
subq source, dest
```

without writing to *dest*

```
..... # %rdi = x
..... # %rsi = y
cmpq %rsi, %rdi   # Compare x to y
setg %al          # Set when >
movzbl %al, %eax
ret              # Result in %eax
```

Bitwise Comparisons

```
testx source, dest dest & source
```

```
testq source, dest
```

is the same as

```
andq source, dest
```

without writing to *dest*

```
..... # %rdi = x  
testq %rdi, %rdi # Compare by x & x  
setne %al # Set when x != 0  
movzbl %al, %eax  
ret # Result in %eax
```

Exercise: Comparisons

```
char ctest(int a, int b, int c)
  char t1 = a ___ b;
  char t2 = b ___ (___)a;
  char t3 = (___)c ___ (___)a;
  char t4 = (___)a ___ (___)c;
  char t5 = c ___ b;
  char t6 = a ___ 0;
  return t1+t2+t3+t4+t5+t6;
}
```

```
movl 8(%rbp),%ecx      # Get a
movl 12(%rbp),%esi     # Get b
cmpl %esi,%ecx        # Compare a-b
setl %al              # t1
cmpl %ecx,%esi        # Compare b-a
setb -1(%rbp)         # t2
cmpw %cx,16(%rbp)    # Compare c-a
setge -2(%rbp)        # t3
movb %cl,%dl
cmpb 16(%rbp),%dl     # Compare a-c
setne %bl            # t4
cmpl %esi,16(%rbp)   # Compare c-b
setg -3(%rbp)         # t5
testl %ecx,%ecx      # Test a&a
sete %dl             # t6
addb -1(%rbp),%al     # t1+=t2
addb -2(%rbp),%al     # t1+=t3
addb %bl,%al          # t1+=t4
addb -3(%rbp),%al     # t1+=t5
addb %dl,%al          # t1+=t6
movsbl %al,%eax      # Convert type
```

Exercise: Comparisons

```
char ctest(int a, int b, int c)
  char t1 = a < b;
  char t2 = b ___ (___)a;
  char t3 = (___)c ___ (___)a;
  char t4 = (___)a ___ (___)c;
  char t5 = c ___ b;
  char t6 = a ___ 0;
  return t1+t2+t3+t4+t5+t6;
}
```

```
movl 8(%rbp),%ecx      # Get a
movl 12(%rbp),%esi     # Get b
cmpl %esi,%ecx        # Compare a-b
setl %al              # t1
cmpl %ecx,%esi        # Compare b-a
setb -1(%rbp)         # t2
cmpw %cx,16(%rbp)    # Compare c-a
setge -2(%rbp)       # t3
movb %cl,%dl
cmpb 16(%rbp),%dl     # Compare a-c
setne %bl             # t4
cmpl %esi,16(%rbp)   # Compare c-b
setg -3(%rbp)        # t5
testl %ecx,%ecx      # Test a&a
sete %dl              # t6
addb -1(%rbp),%al     # t1+=t2
addb -2(%rbp),%al     # t1+=t3
addb %bl,%al          # t1+=t4
addb -3(%rbp),%al     # t1+=t5
addb %dl,%al          # t1+=t6
movsbl %al,%eax      # Convert type
```

Exercise: Comparisons

```
char ctest(int a, int b, int c)
  char t1 = a < b;
  char t2 = b < (unsigned)a;
  char t3 = (__)c __ (__ )a;
  char t4 = (__)a __ (__ )c;
  char t5 = c __ b;
  char t6 = a __ 0;
  return t1+t2+t3+t4+t5+t6;
}
```

```
movl 8(%rbp),%ecx      # Get a
movl 12(%rbp),%esi     # Get b
cmpl %esi,%ecx        # Compare a-b
setl %al              # t1
cmpl %ecx,%esi        # Compare b-a
setb -1(%rbp)         # t2
cmpw %cx,16(%rbp)    # Compare c-a
setge -2(%rbp)       # t3
movb %cl,%dl
cmpb 16(%rbp),%dl    # Compare a-c
setne %bl            # t4
cmpl %esi,16(%rbp)  # Compare c-b
setg -3(%rbp)       # t5
testl %ecx,%ecx     # Test a&a
sete %dl             # t6
addb -1(%rbp),%al   # t1+=t2
addb -2(%rbp),%al   # t1+=t3
addb %bl,%al        # t1+=t4
addb -3(%rbp),%al   # t1+=t5
addb %dl,%al        # t1+=t6
movsbl %al,%eax     # Convert type
```

Exercise: Comparisons

```
char ctest(int a, int b, int c)
  char t1 = a < b;
  char t2 = b < (unsigned)a;
  char t3 = (short)c >= (short)a;
  char t4 = (__)a __ (__)c;
  char t5 = c __ b;
  char t6 = a __ 0;
  return t1+t2+t3+t4+t5+t6;
}
```

```
movl 8(%rbp),%ecx      # Get a
movl 12(%rbp),%esi     # Get b
cmpl %esi,%ecx        # Compare a-b
setl %al              # t1
cmpl %ecx,%esi        # Compare b-a
setb -1(%rbp)         # t2
cmpw %cx,16(%rbp)    # Compare c-a
setge -2(%rbp)       # t3
movb %cl,%dl
cmpb 16(%rbp),%dl    # Compare a-c
setne %bl            # t4
cmpl %esi,16(%rbp)  # Compare c-b
setg -3(%rbp)       # t5
testl %ecx,%ecx     # Test a&a
sete %dl            # t6
addb -1(%rbp),%al   # t1+=t2
addb -2(%rbp),%al   # t1+=t3
addb %bl,%al        # t1+=t4
addb -3(%rbp),%al   # t1+=t5
addb %dl,%al        # t1+=t6
movsbl %al,%eax     # Convert type
```

Exercise: Comparisons

```
char ctest(int a, int b, int c)
  char t1 = a < b;
  char t2 = b < (unsigned)a;
  char t3 = (short)c >= (short)a;
  char t4 = (char)a != (char)c;
  char t5 = c ___ b;
  char t6 = a ___ 0;
  return t1+t2+t3+t4+t5+t6;
}
```

```
movl 8(%rbp),%ecx      # Get a
movl 12(%rbp),%esi     # Get b
cmpl %esi,%ecx        # Compare a-b
setl %al              # t1
cmpl %ecx,%esi        # Compare b-a
setb -1(%rbp)         # t2
cmpw %cx,16(%rbp)    # Compare c-a
setge -2(%rbp)       # t3
movb %cl,%dl
cmpb 16(%rbp),%dl    # Compare a-c
setne %bl            # t4
cmpl %esi,16(%rbp)  # Compare c-b
setg -3(%rbp)       # t5
testl %ecx,%ecx     # Test a&a
sete %dl            # t6
addb -1(%rbp),%al   # t1+=t2
addb -2(%rbp),%al   # t1+=t3
addb %bl,%al        # t1+=t4
addb -3(%rbp),%al   # t1+=t5
addb %dl,%al        # t1+=t6
movsbl %al,%eax     # Convert type
```


Exercise: Comparisons

```
char ctest(int a, int b, int c)
  char t1 = a < b;
  char t2 = b < (unsigned)a;
  char t3 = (short)c >= (short)a;
  char t4 = (char)a != (char)c;
  char t5 = c > b;
  char t6 = a ___ 0;
  return t1+t2+t3+t4+t5+t6;
}
```

```
movl 8(%rbp),%ecx      # Get a
movl 12(%rbp),%esi     # Get b
cmpl %esi,%ecx        # Compare a-b
setl %al              # t1
cmpl %ecx,%esi        # Compare b-a
setb -1(%rbp)         # t2
cmpw %cx,16(%rbp)    # Compare c-a
setge -2(%rbp)       # t3
movb %cl,%dl
cmpb 16(%rbp),%dl    # Compare a-c
setne %bl            # t4
cmpl %esi,16(%rbp)  # Compare c-b
setg -3(%rbp)       # t5
testl %ecx,%ecx     # Test a&a
sete %dl            # t6
addb -1(%rbp),%al   # t1+=t2
addb -2(%rbp),%al   # t1+=t3
addb %bl,%al        # t1+=t4
addb -3(%rbp),%al   # t1+=t5
addb %dl,%al        # t1+=t6
movsbl %al,%eax     # Convert type
```

Exercise: Comparisons

```
char ctest(int a, int b, int c)
  char t1 = a < b;
  char t2 = b < (unsigned)a;
  char t3 = (short)c >= (short)a;
  char t4 = (char)a != (char)c;
  char t5 = c > b;
  char t6 = a == 0;
  return t1+t2+t3+t4+t5+t6;
}
```

```
movl 8(%rbp),%ecx      # Get a
movl 12(%rbp),%esi     # Get b
cmpl %esi,%ecx        # Compare a-b
setl %al              # t1
cmpl %ecx,%esi        # Compare b-a
setb -1(%rbp)         # t2
cmpw %cx,16(%rbp)    # Compare c-a
setge -2(%rbp)       # t3
movb %cl,%dl
cmpb 16(%rbp),%dl     # Compare a-c
setne %bl            # t4
cmpl %esi,16(%rbp)   # Compare c-b
setg -3(%rbp)        # t5
testl %ecx,%ecx      # Test a&a
sete %dl             # t6
addb -1(%rbp),%al    # t1+=t2
addb -2(%rbp),%al    # t1+=t3
addb %bl,%al         # t1+=t4
addb -3(%rbp),%al    # t1+=t5
addb %dl,%al         # t1+=t6
movsbl %al,%eax      # Convert type
```

Conditional Assignment

```
char x = (a < b);
```

use `setl`

```
long x = ((a < b) ? 17 : 42);
```

Use `cmovlq`, which is like `movq`, but only if the condition codes imply “less than”

```
.... # %rdi = a
.... # %rsi = b
.... # %rax as x
movq $42, %rax # Guess 42
cmpq %rsi, %rdi # Compare a to b
cmovlq $17, %rax # Maybe correct guess
```

Conditional Move

`cmovccx source, dest`

conditionally copies *source* to *dest*

<code>cmovex</code>	equal / zero	ZF a.k.a. <code>cmovzx</code>
<code>cmovnex</code>	not equal / not zero	~ZF a.k.a. <code>cmovnzx</code>
<code>cmovsx</code>	negative	SF
<code>cmovnsx</code>	non-negative	~SF
<code>cmovgex</code>	greater signed	~(SF^OF) & ~ZF
<code>cmovgex</code>	greater or equal signed	~(SF^OF)
<code>cmovlxx</code>	less signed	(SF^OF)
<code>cmovlex</code>	less or equal signed	(SF^OF) ZF
<code>cmovax</code>	above unsigned	~CF & ~ZF
<code>cmovbxx</code>	below unsigned	CF

Comparisons and Conditionals

```
if (a < b) {  
    ....  
}
```

```
0x200: cmovgeq $0x203, %rip  
0x201: ....  
0x202: ....  
0x203: ....
```

?

```
if (a < b) {  
    ....  
} else {  
    ....  
}
```

```
0x300: cmovgeq $0x303, %rip  
0x301: ....  
0x302: movq $0x305, %rip  
0x303: ....  
0x304: ....  
0x305: ....
```

?

?

but we can't use `%rip` as a destination

Conditional Jumps

`jcc source`

conditionally sets the program counter to the value of *source*

<code>je</code>	equal / zero	ZF a.k.a. <code>jz</code>
<code>jne</code>	not equal / not zero	~ZF a.k.a. <code>jnz</code>
<code>js</code>	negative	SF
<code>jns</code>	non-negative	~SF
<code>jg</code>	greater signed	~(SF^OF) & ~ZF
<code>jge</code>	greater or equal signed	~(SF^OF)
<code>jl</code>	less signed	(SF^OF)
<code>jle</code>	less or equal signed	(SF^OF) ZF
<code>ja</code>	above unsigned	~CF & ~ZF
<code>jb</code>	below unsigned	CF

Unconditional Jump

```
jmp source
```

always sets the program counter to the value of *source*

Comparisons and Conditionals

```
if (a < b) {  
    ....  
}
```

```
0x200: jge $0x203  
0x201: ....  
0x202: ....  
0x203: ....
```

```
    if (a >= b) goto skip;  
    ....  
skip:  
    ....
```

```
if (a < b) {  
    ....  
} else {  
    ....  
}
```

```
0x300: jge $0x303  
0x301: ....  
0x302: jmp $0x305  
0x303: ....  
0x304: ....  
0x305: ....
```

```
    if (a >= b) goto else;  
    .... goto skip;  
else:  
    ....  
skip:  
    ....
```


Test and Jump Alternatives

```
if (a < b) {  
    then  
} else {  
    else  
}  
.....
```

```
if (a >= b) goto else;  
then goto skip;  
else:  
    else  
skip:  
    .....
```

```
if (a < b) goto then;  
else goto skip;  
then:  
    then  
skip:  
    .....
```

Conditional Example

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

[Copy](#)

```
0:    cmpq    %rsi, %rdi  
3:    jle    0xc  
5:    movq    %rdi, %rax  
8:    subq    %rsi, %rax  
b:    retq  
c:    movq    %rsi, %rax  
f:    subq    %rdi, %rax  
12:   retq
```

Conditional Example

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

[Copy](#)

```
0:  cmpq  %rsi, %rdi  
3:  jle  0xc  
5:  movq  %rdi, %rax  
8:  subq  %rsi, %rax  
b:  retq  
c:  movq  %rsi, %rax  
f:  subq  %rdi, %rax  
12: retq
```

absdiff:

```
    cmpq  %rsi, %rdi  
    jle  .L4  
    movq  %rdi, %rax  
    subq  %rsi, %rax  
    ret
```

.L4:

```
    movq  %rsi, %rax  
    subq  %rdi, %rax  
    ret
```

More C Control Forms

- `do while` loops
- `while` loops
- `for` loops

do-while Loops

```
do {  
    body  
} while (test);
```

```
loop:  
    body  
    if (test)  
        goto loop;
```

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

register	variable	initial value
%ecx	i	0
%esi	n	n
%ebx	val	0
%edx	nval	1
%eax	t	

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

```
.L6:
    leal (%edx,%ebx),%eax
    movl %edx,%ebx
    movl %eax,%edx
    incl %ecx
    cmpl %esi,%ecx
    jl .L6
    movl %ebx,%eax
```


do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

register	variable	initial value
%ecx	i	0
%esi	n	n
%ebx	val	0
%edx	nval	1
%eax	t	

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

.L6:

```
leal (%edx,%ebx),%eax
movl %edx,%ebx
movl %eax,%edx
incl %ecx
cmpl %esi,%ecx
jl .L6
movl %ebx,%eax
```

t = ...

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

register	variable	initial value
%ecx	i	0
%esi	n	n
%ebx	val	0
%edx	nval	1
%eax	t	

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

.L6:

```
leal (%edx,%ebx),%eax
movl %edx,%ebx
movl %eax,%edx
incl %ecx
cmpl %esi,%ecx
jl .L6
movl %ebx,%eax
```

t = ...
val = nval

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

register	variable	initial value
%ecx	i	0
%esi	n	n
%ebx	val	0
%edx	nval	1
%eax	t	

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

.L6:

```
leal (%edx,%ebx),%eax
movl %edx,%ebx
movl %eax,%edx
incl %ecx
cmpl %esi,%ecx
jl .L6
movl %ebx,%eax
```

t = ...
val = nval
nval = t

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

register	variable	initial value
%ecx	i	0
%esi	n	n
%ebx	val	0
%edx	nval	1
%eax	t	

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

.L6:

```
leal (%edx,%ebx),%eax
movl %edx,%ebx
movl %eax,%edx
incl %ecx
cmpl %esi,%ecx
jl .L6
movl %ebx,%eax
```

t = ...
val = nval
nval = t
i++

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

register	variable	initial value
%ecx	i	0
%esi	n	n
%ebx	val	0
%edx	nval	1
%eax	t	

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

.L6:

```
leal (%edx,%ebx),%eax
movl %edx,%ebx
movl %eax,%edx
incl %ecx
cmpl %esi,%ecx
jl .L6
movl %ebx,%eax
```

```
t = ...
val = nval
nval = t
i++
i - n
```

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

register	variable	initial value
%ecx	i	0
%esi	n	n
%ebx	val	0
%edx	nval	1
%eax	t	

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

.L6:

```
leal (%edx,%ebx),%eax
movl %edx,%ebx
movl %eax,%edx
incl %ecx
cmpl %esi,%ecx
jl .L6
movl %ebx,%eax
```

```
t = ...
val = nval
nval = t
i++
i - n
if (i < n)
```

do-while Example

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

register	variable	initial value
%ecx	i	0
%esi	n	n
%ebx	val	0
%edx	nval	1
%eax	t	

```
int fib_dw(int n) {
    int i = 0;
    int val = 0;
    int nval = 1;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;

    return val;
}
```

.L6:

```
leal (%edx,%ebx),%eax
movl %edx,%ebx
movl %eax,%edx
incl %ecx
cmpl %esi,%ecx
jl .L6
movl %ebx,%eax
```

```
t = ...
val = nval
nval = t
i++
i - n
if (i < n)
return val
```

while Loops

```
while (test) {  
    body  
}
```

```
loop:  
    if (!test)  
        goto done;  
    body  
    goto loop;  
done:
```

```
    if (!test)  
        goto done;  
loop:  
    body  
    if (test)  
        goto loop;  
done:
```


while Example

```
int fib_w(int n) {  
    int i = 1;  
    int val = 1;  
    int nval = 1;  
  
    while (i < n) {  
        int t = val + nval;  
        val = nval;  
        nval = t;  
        i++;  
    }  
  
    return val;  
}
```

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
    return val;
}
```

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

val - n

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

val - n
if (val >= n)

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

```
val - n
if (val >= n)
nmi = n-1
```

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

```
val - n
if (val >= n)
nmi = n-1

t = ...
```

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

```
val = n
if (val >= n)
nmi = n-1

t = ...
val = nval
```


while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

```
val - n
if (val >= n)
nmi = n-1

t = ...
val = nval
nval = t
```

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

```
val = n
if (val >= n)
nmi = n-1

t = ...
val = nval
nval = t
nmi--
```

while Example

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

register	variable	initial value
%edx	nmi	n-i
%ebx	val	1
%ecx	nval	1
%eax	n / t	

```
int fib_dw(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;
    if (i >= n) goto done;
loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n) goto loop;
done:
```

```
    cmpl %eax,%ebx
    jge .L9
    leal -1(%eax),%edx
.L10:
    leal (%ecx,%ebx),%eax
    movl %ecx,%ebx
    movl %eax,%ecx
    decl %edx
    jnz .L10
.L9:
```

```
val = n
if (val >= n)
nmi = n-1

t = ...
val = nval
nval = t
nmi--
if (nmi != 0)
```

for Loops

```
for (init; test; update) {  
    body  
}
```

```
    init  
loop:  
    if (!test)  
        goto done;  
    body  
    update  
    goto loop;  
done:
```

for Example

```
int fib_f(int n) {
    int i;
    int val = 1;
    int nval = 1;

    for (i = 1; i < n; i++) {
        int t = val + nval;
        val = nval;
        nval = t;
    }

    return val;
}
```

for Example

```
int fib_f(int n) {
    int i;
    int val = 1;
    int nval = 1;

    for (i = 1; i < n; i++) {
        int t = val + nval;
        val = nval;
        nval = t;
    }

    return val;
}
```

```
int fib_w(int n) {
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

Same machine code

Another Loop Exercise

```
int loop_while(int a, int b) {
    int i = 0;
    int result = a;

    while (i < 256) {
        result += a;
        a -= b;
        i += b;
    }

    return result;
}
```

```
    movl 8(%rbp),%eax # Get a
    movl 12(%rbp),%ebx # Get b
    xorl %ecx,%ecx
    movl %eax,%edx
.L5:
    addl %eax,%edx
    subl %ebx,%eax
    addl %ebx,%ecx
    cmpl $255,%ecx
    jle .L5
    movl %edx,%eax
```

register	variable	initial value
%eax		
%ebx		
%ecx		
%edx		

Another Loop Exercise

```
int loop_while(int a, int b) {
    int i = 0;
    int result = a;

    while (i < 256) {
        result += a;
        a -= b;
        i += b;
    }

    return result;
}
```

```
movl 8(%rbp),%eax # Get a
movl 12(%rbp),%ebx # Get b
xorl %ecx,%ecx
movl %eax,%edx
.L5:
addl %eax,%edx
subl %ebx,%eax
addl %ebx,%ecx
cmpl $255,%ecx
jle .L5
movl %edx,%eax
```

register	variable	initial value
%eax	a	a
%ebx		
%ecx		
%edx		

Another Loop Exercise

```
int loop_while(int a, int b) {
    int i = 0;
    int result = a;

    while (i < 256) {
        result += a;
        a -= b;
        i += b;
    }

    return result;
}
```

```
movl 8(%rbp),%eax # Get a
movl 12(%rbp),%ebx # Get b
xorl %ecx,%ecx
movl %eax,%edx
.L5:
addl %eax,%edx
subl %ebx,%eax
addl %ebx,%ecx
cmpl $255,%ecx
jle .L5
movl %edx,%eax
```

register	variable	initial value
<code>%eax</code>	<code>a</code>	<code>a</code>
<code>%ebx</code>	<code>b</code>	<code>b</code>
<code>%ecx</code>		
<code>%edx</code>		

Another Loop Exercise

```
int loop_while(int a, int b) {
    int i = 0;
    int result = a;

    while (i < 256) {
        result += a;
        a -= b;
        i += b;
    }

    return result;
}
```

```
movl 8(%rbp),%eax # Get a
movl 12(%rbp),%ebx # Get b
xorl %ecx,%ecx
movl %eax,%edx
.L5:
addl %eax,%edx
subl %ebx,%eax
addl %ebx,%ecx
cmpl $255,%ecx
jle .L5
movl %edx,%eax
```

register	variable	initial value
%eax	a	a
%ebx	b	b
%ecx	i	0
%edx		

Another Loop Exercise

```
int loop_while(int a, int b) {  
    int i = 0;  
    int result = a;  
  
    while (i < 256) {  
        result += a;  
        a -= b;  
        i += b;  
    }  
  
    return result;  
}
```

```
movl 8(%rbp),%eax # Get a  
movl 12(%rbp),%ebx # Get b  
xorl %ecx,%ecx  
movl %eax,%edx  
.L5:  
addl %eax,%edx  
subl %ebx,%eax  
addl %ebx,%ecx  
cmpl $255,%ecx  
jle .L5  
movl %edx,%eax
```

register	variable	initial value
%eax	a	a
%ebx	b	b
%ecx	i	0
%edx	result	a

Another Loop Exercise

```
int loop_while(int a, int b) {
    int i = 0;
    int result = a;

    while (i < 256) {
        result += a;
        a -= b;
        i += b;
    }

    return result;
}
```

```
movl 8(%rbp),%eax # Get a
movl 12(%rbp),%ebx # Get b
xorl %ecx,%ecx
movl %eax,%edx
.L5:
addl %eax,%edx
subl %ebx,%eax
addl %ebx,%ecx
cmpl $255,%ecx
jle .L5
movl %edx,%eax
```

Compiler optimizes away initial
 $i < 255$, since i clearly starts as 0

register	variable	initial value
<code>%eax</code>	<code>a</code>	<code>a</code>
<code>%ebx</code>	<code>b</code>	<code>b</code>
<code>%ecx</code>	<code>i</code>	<code>0</code>
<code>%edx</code>	<code>result</code>	<code>a</code>


switch Statements

```
long switch_eg(long x, long y, long z) {
    long w = 1;

    switch (x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Multi-way branching on
an integer value

switch Statements

```
long switch_eg(long x, long y, long z) {  
    long w = 1;  
  
    switch (x) {  
    case 1:    
        w = y*z;  
        break;  
    case 2:  
        w = y/z;  
    case 3:  
        w += z;  
        break;  
    case 5:  
    case 6:  
        w -= z;  
        break;  
    default:  
        w = 2;  
    }  
    return w;  
}
```

Multi-way branching on
an integer value

switch Statements

```
long switch_eg(long x, long y, long z) {
    long w = 1;

    switch (x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Jump out of switch

Multi-way branching on
an integer value

switch Statements

```
long switch_eg(long x, long y, long z) {
    long w = 1;

    switch (x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Jump here if $x == 2$

Multi-way branching on
an integer value

switch Statements

```
long switch_eg(long x, long y, long z) {
    long w = 1;

    switch (x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Multi-way branching on
an integer value

No break, so **fall through** for
x == 2

switch Statements

```
long switch_eg(long x, long y, long z) {  
    long w = 1;  
  
    switch (x) {  
    case 1:  
        w = y*z;  
        break;  
    case 2:  
        w = y/z;  
    case 3:  
        w += z;  
        break;  
    case 5:  
    case 6:  
        w -= z;  
        break;  
    default:  
        w = 2;  
    }  
    return w;  
}
```

Multi-way branching on
an integer value

Jump here if $x == 5$ or $x == 6$

switch Statements

```
long switch_eg(long x, long y, long z) {  
    long w = 1;  
  
    switch (x) {  
    case 1:  
        w = y*z;  
        break;  
    case 2:  
        w = y/z;  
    case 3:  
        w += z;  
        break;  
    case 5:  
    case 6:  
        w -= z;  
        break;  
    default:  
        w = 2;  
    }  
    return w;  
}
```

Multi-way branching on
an integer value

Jump here if **x** is some value not
covered above:

x < 1 or **x == 4** or **x > 6**

Implementing switch

Small number of cases: **if** plus **goto**

```
switch (x) {  
  case 1:  
    one  
    break;  
  case 2:  
    two  
  case 3:  
    two-and-three  
}
```

```
if (x == 1)  
  goto one;  
else if (x == 2)  
  goto two;  
else if (x == 3)  
  goto three;  
else  
  goto done;  
  
one:  
  one  
  goto done;  
two:  
  two  
three:  
  two-and-three  
done:
```

Implementing switch

Many consecutive cases: **jump table**

```
switch (x) {  
case 1:  
    one  
    break;  
case 2:  
    ....  
case 99:  
    ninety-nine  
}
```

```
void *jump_table[] = { &&one, &&two, ...  
                      &&ninety_nine };  
  
if ((x >= 1) && (x <= 99))  
    goto *jump_table[x-1];  
else  
    goto done;  
  
one:  
    one  
    goto done;  
two:  
    ...  
ninety_nine:  
    ninety-nine  
done:
```

Jump Table Machine Code

```
int jump(int v, int a, int b, int c) {  
    void *jump_table[] = { &&one, &&two, &&three };  
  
    goto *jump_table[v-1];  
  
one:  
    v = a;  
    goto done;  
two:  
    v = b;  
    goto done;  
three:  
    v = c;  
done:  
    return v;  
}
```

[Copy](#)

Jump Table Machine Code

```
int jump(int v, int a, int b, int c) {  
    void *jump_table[] = { &&one, &&two, &&three };  
  
    goto *jump_table[v-1];  
}
```

```
one:  
    v = a;  
    goto done;  
two:  
    v = b;  
    goto done;  
three:  
    v = c;  
done:  
    return v;  
}  
  
400530: movq    $0x400552,-0x20(%rsp)  
400539: movq    $0x400555,-0x18(%rsp)  
400542: movq    $0x400558,-0x10(%rsp)  
40054b: movslq  %edi,%rdi  
40054e: jmpq    *-0x20(%rsp,%rdi,8)  
400552: mov     %esi,%eax  
400554: retq  
400555: mov     %edx,%eax  
400557: retq  
400558: mov     %ecx,%eax  
40055a: nopw   0x0(%rax,%rax,1)  
400560: retq
```

Compiler-Generated Jump Tables

```
long switch_eg(long x, long y, long z) {  
    long w = 1;  
  
    switch(x) {  
        ...  
    }  
  
    return w;  
}
```


Compiler-Generated Jump Tables

```
long switch_eg(long x, long y, long z) {  
    long w = 1;  
  
    switch(x) {  
        ...  
    }  
  
    return w;  
}
```

```
.section .rodata  
    .align 8  
.L4:  
    .quad .L8 # x == 0  
    .quad .L3 # x == 1  
    .quad .L5 # x == 2  
    .quad .L9 # x == 3  
    .quad .L8 # x == 4  
    .quad .L7 # x == 5  
    .quad .L7 # x == 6
```

Compiler-Generated Jump Tables

```
long switch_eg(long x, long y, long z) {
    long w = 1;

    switch(x) {
        ...
    }

    return w;
}
```

```
.section .rodata
    .align 8
.L4:
    .quad .L8 # x == 0
    .quad .L3 # x == 1
    .quad .L5 # x == 2
    .quad .L9 # x == 3
    .quad .L8 # x == 4
    .quad .L7 # x == 5
    .quad .L7 # x == 6
```

```
switch_eg:
    movq    %rdx, %rcx
    cmpq    $6, %rdi      # Compare x to 6
    ja     .L8            # > 6 as unsigned ⇒ use default
    jmp     *.L4(,%rdi,8) # Goto *jump_table[x]
```

Compiler-Generated Jump Tables

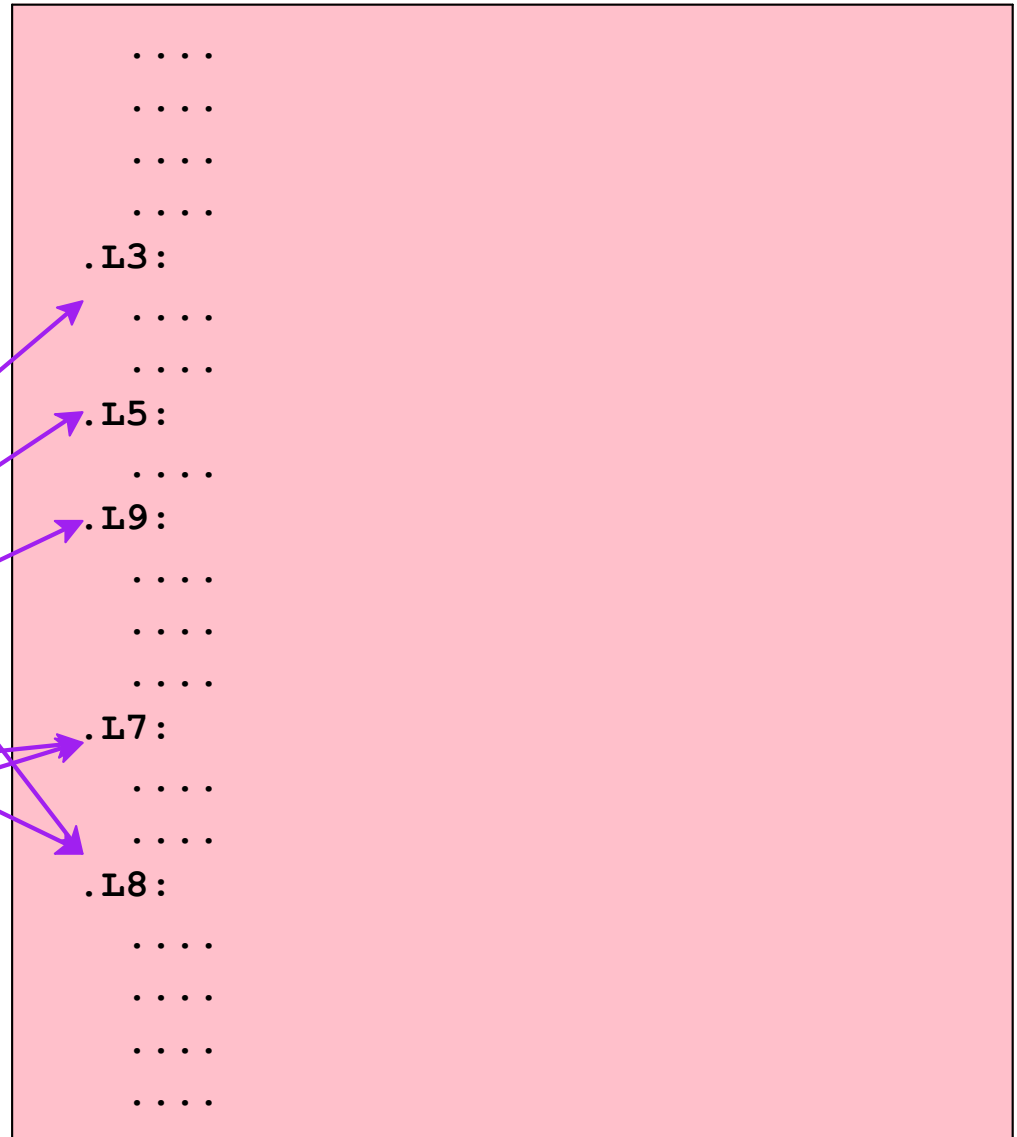
```
.section .rodata
.align 8
.L4:
.quad .L8 # x == 0
.quad .L3 # x == 1
.quad .L5 # x == 2
.quad .L9 # x == 3
.quad .L8 # x == 4
.quad .L7 # x == 5
.quad .L7 # x == 6
```

```
long switch_eg(long x, long y, long z) {
    long w = 1;

    switch (x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Compiler-Generated Jump Tables

```
.section .rodata
.align 8
.L4:
.quad .L8 # x == 0
.quad .L3 # x == 1
.quad .L5 # x == 2
.quad .L9 # x == 3
.quad .L8 # x == 4
.quad .L7 # x == 5
.quad .L7 # x == 6
```



Compiler-Generated Jump Tables

```
long w = 1;

switch (x) {
    ...
case 2:
    w = y/z;
case 3:
    w += z;
    break;
    ...
}

return w;
```

```
.L5:                                # Case x == 2
    movq    %rsi, %rax              # w = y
    cqto                                # Expands w
    idivq   %rcx                    # w = w / z
    jmp     .L6                      # Goto merge
.L9:                                # Case x == 3
    movl    $1, %eax                # w = 1
.L6:                                # merge:
    addq    %rcx, %rax              # w += z
    ret
```

register	variable
%rdi	x
%rsi	y
%rcx	z
%rax	w
%rdx	w (high bits for division)