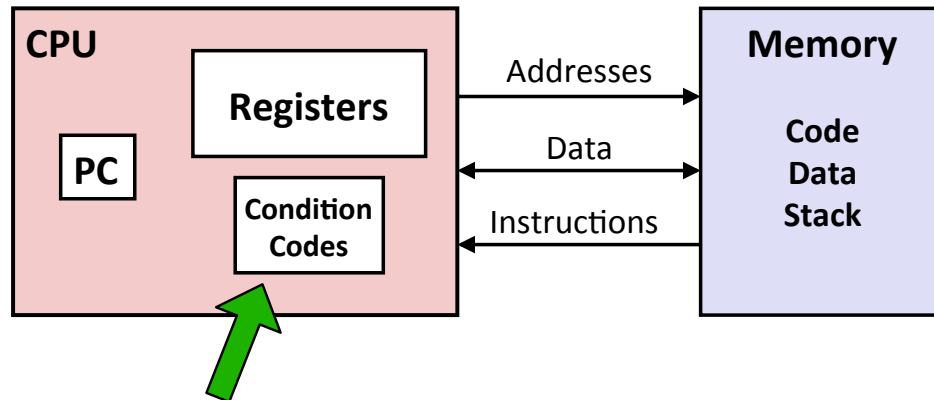
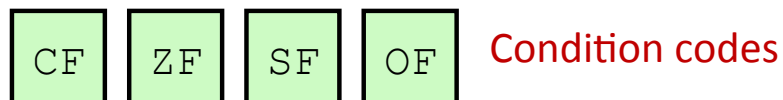


Machine Code: Control Flow



Condition Codes



CF: **Carry flag** (set if carry-out bit = 1)

ZF: **Zero flag** (set if result = 0)

SF: **Sign flag** (set if result top bit = 1)

OF: **Overflow flag** (set if signed overflow)

- (1) implicit settings (all arith except `leaq`)
- (2) `cmpq a,b` (set CCs based on `b - a`)
- (3) `testq a,b` (set CCs based on `a & b`)

Reading Condition Codes

SetX	Condition	Description
sete / setz	ZF	Equal / Zero
setne	\sim ZF	Not Equal / Not Zero
sets	SF	Negative
setns	\sim SF	Nonnegative
setg	\sim (SF \wedge OF) & \sim ZF	Greater (Signed)
setge	\sim (SF \wedge OF)	Greater or Equal (Signed)
setl	(SF \wedge OF)	Less (Signed)
setle	(SF \wedge OF) ZF	Less or Equal (Signed)
seta	\sim CF & \sim ZF	Above (unsigned)
setb	CF	Below (unsigned)

Example: Greater Than

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

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

y **x**
↓ ↓

```
cmpq   %rsi, %rdi   # Compare x:y  
setg   %al          # Set when >  
movzbl %al, %eax    # Zero rest of %rax  
ret
```

Goto in C

```
#include <stdio.h>

int main() {
    int a = 0;

    label → FOO:
    while (a < 20) {
        if (a == 15) {
            a++;
            goto FOO;
        }

        printf("%d\n", a);
        a++;
    }

    return 0;
}
```

Jumping

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF^OF) & ~ZF	Greater (Signed)
jge	~(SF^OF)	Greater or Equal (Signed)
jl	(SF^OF)	Less (Signed)
jle	(SF^OF) ZF	Less or Equal (Signed)
ja	~CF & ~ZF	Above (unsigned)
jb	CF	Below (unsigned)

Example: absdiff

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

y **x**
↓ ↓

```
absdiff:
    cmpq   %rsi, %rdi # x:y
    jle    .L4
    movq   %rdi, %rax
    subq   %rsi, %rax
    ret
.L4:      # x <= y
    movq   %rsi, %rax
    subq   %rdi, %rax
    ret
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

absdiff with Goto

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

y **x**
↓ ↓

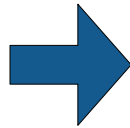
```
absdiff:
    cmpq   %rsi, %rdi # x:y
    jle    .L4
    movq   %rdi, %rax
    subq   %rsi, %rax
    ret
.L4:      # x <= y
    movq   %rsi, %rax
    subq   %rdi, %rax
    ret
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

Conditional to Goto

Conditional Version

```
if (test)
    then-cmd
else
    else-cmd
...
```



Goto Version

```
t = test;
if (!t) goto false;
then-cmd
goto done;
false:
    else-cmd
done:
    ...
```

BitBombs!



Parsing Input Strings with scanf

```
int things_read; // numbers of "objects" read by scanf

int i;          // declared but uninitialized
char c;

// read an int from user, store it at address &i
things_read = scanf("%d", &i);

// read an int and a char, store at addresses &i and &c
things_read = scanf("%d %c", &i, &c);

// sscanf variant: read from string instead of user input
things_read = sscanf(some_str, "%d %c", &i, &c);
```

Do-While Loops

C Code

```
long loop_dowhile
(unsigned long x) {
    long result = 0;
    do {
        result += x & 0x1;
        x = x >> 1;
    } while (x);
    return result;
}
```



Goto Version

```
long loop_goto
(unsigned long x) {
    long result = 0;
loop:
    result += x & 0x1;
    x = x >> 1;
    if (x) goto loop;
    return result;
}
```

Do-While Loop Compilation

Goto Version

```
long loop_goto
(unsigned long x) {
    long result = 0;
loop:
    result += x & 0x1;
    x = x >> 1;
    if (x) goto loop;
    return result;
}
```

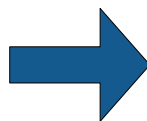
Register	Use(s)
%rdi	Argument x
%rax	result

```
    movl    $0, %eax    # result = 0
.L2:                                # loop:
    movq    %rdi, %rdx
    andl    $1, %edx    # t = x & 0x1
    addq    %rdx, %rax  # result += t
    shrq    %rdi        # x = x >> 1
    jne     .L2         # if (x) goto loop [also jnz]
    ret
```

While Loops: Jump to Middle

While Version

```
while (test)
    Body
...
```



Goto Version

```
goto middle;
loop:
    Body
middle:
    t = test;
    if (t) goto loop;
done:
    ...
```

While Loops: Guarded Do

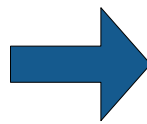
While Version

```
while (test)
  Body
...
```



Do-While Version

```
if (!test)
  goto done;
do
  Body
  while (test);
done:
...
```



Goto Version

```
t = test;
if (!t) goto done;
loop:
  Body
  t = test;
  if (t) goto loop;
done:
```

While Loop Example

C Code

```
long bitcount(unsigned long x) {
  long result = 0;
  while (x) {
    result += x & 0x1;
    x = x >> 1;
  }
  return result;
}
```

Jump to Middle

```
long bitcount_jtm
(unsigned long x) {
  long result = 0;
  goto middle;
loop:
  result += x & 0x1;
  x = x >> 1;
middle:
  if (x) goto loop;
  return result;
}
```

Guarded Do

```
long bitcount_gd
(unsigned long x) {
  long result = 0;
  if (!x) goto done;
loop:
  result += x & 0x1;
  x = x >> 1;
  if (x) goto loop;
done:
  return result;
}
```


Guarded Do Optimization

C Code

```
int x = 0;
while (x < 5) {
    print(x);
    x++;
}
```



Guarded Do

```
int x = 0;
if (x >= 5) goto done;
loop:
    print(x);
    x++;
    if (x < 5) goto loop;
done:
...
```

For Loops

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



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

Switch Statements

```
void print_digit(int digit) {  
    switch (digit) {  
        case 0:  
            printf("zero\n");  
            break;  
        case 1:  
            printf("one\n");  
            break;  
        case 2:  
            printf("two\n");  
            break;  
        case 3:  
            printf("three\n");  
            break;  
        ...  
        case 9:  
            printf("nine\n");  
            break;  
        default:  
            printf("not a digit\n");  
            break;  
    }  
}
```

Switch Fall Through

```
long switch_example  
(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;  
}
```

No break (fall through)

No case 4 (default)

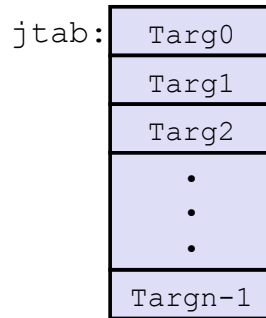
Fall through (case 5 same as 6)

Jump Tables

Switch Form

```
switch(x) {
  case val_0:
    Block 0
  case val_1:
    Block 1
    . . .
  case val_n-1:
    Block n-1
}
```

Jump Table



Targ0:

Code Block 0

Targ1:

Code Block 1

Targ2:

Code Block 2

⋮

Targn-1:

Code Block n-1

Translation (Extended C)

```
goto *jtab[x];
```

Switch Example

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

Jump Table

```
.L4: # address of JTab
.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
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rdx	Argument z
%rax	Return value

```
switch_example:
  movq   %rdx, %rcx
  cmpq   $6, %rdi    # x:6
  ja     .L8         # Use default
  jmp    *.L4(,%rdi,8) # goto *JTab[x]
```

↑
Indirect jump

Example Jump Table

Jump Table

```
.L4: # address of JTab
.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(x) {
case 1: // .L3
    w = y*z;
    break;
case 2: // .L5
    w = y/z;
    /* Fall Through */
case 3: // .L9
    w += z;
    break;
case 5:
case 6: // .L7
    w -= z;
    break;
default: // .L8
    w = 2;
}
```

Code Blocks

```
long w = 1;
switch(x) {
case 1: // .L3
    w = y*z;
    break;
case 2: // .L5
    w = y/z;
    /* Fall Through */
case 3: // .L9
    w += z;
    break;
case 5:
case 6: // .L7
    w -= z;
    break;
default: // .L8
    w = 2;
}
return w;
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rdx	Argument z
%rax	Return value

```
switch_example:
movq %rdx, %rcx
cmpq $6, %rdi # x:6
ja .L8 # Use default
jmp *.L4(,%rdi,8) # goto *JTab[x]
```

```
.L3: # Case 1
movq %rsi, %rax # y
imulq %rdx, %rax # y*z
ret
.L5: # Case 2
movq %rsi, %rax
cqto
idivq %rcx # y/z
jmp .L6 # goto merge
.L9: # Case 3
movl $1, %eax # w = 1
.L6: # merge:
addq %rcx, %rax # w += z
ret
.L7: # Case 5,6
movl $1, %eax # w = 1
subq %rdx, %rax # w -= z
ret
.L8: # Default:
movl $2, %eax # 2
ret
```