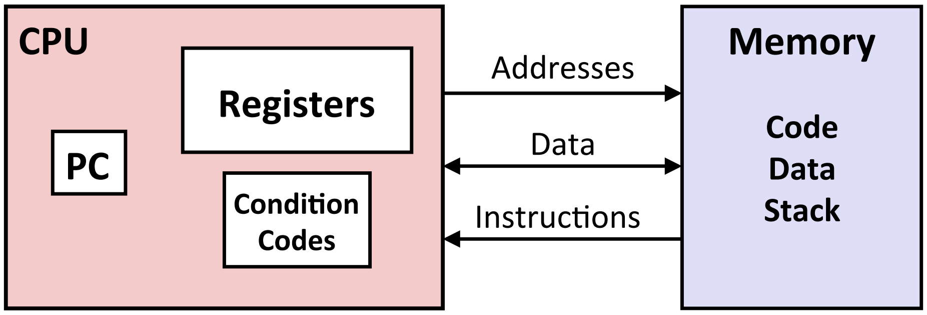
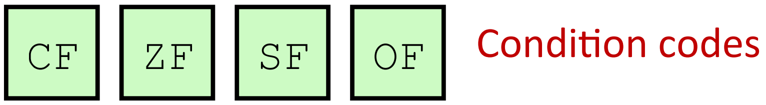
Recap: Assembly View of the Machine



Condition Codes



CF: **Carry flag** (unsigned overflow)

ZF: **Zero flag** (zero result)

SF: **Sign flag** (negative result)

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

Reading Condition Codes

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

Recap: Single-Byte Virtual Registers

`%rax`	`%al`	`%r8`	`%r8b`
`%rbx`	`%bl`	`%r9`	`%r9b`
`%rcx`	`%cl`	`%r10`	`%r10b`
`%rdx`	`%dl`	`%r11`	`%r11b`
`%rsi`	`%sil`	`%r12`	`%r12b`
`%rdi`	`%dil`	`%r13`	`%r13b`
`%rsp`	`%spl`	`%r14`	`%r14b`
`%rbp`	`%bpl`	`%r15`	`%r15b`

Example: Greater Than

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

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

```
    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 = 10;

  LABEL:do {

       if (a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LABEL;
       }

       printf("value of a: %d\n", a);
       a++;

     } while (a < 20);

     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;
}
```

```
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

```
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
```

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

Conditional to Goto

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

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

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

```
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
```

Bitbombs!

Input in C with scanf

```
int things_read;

int i;      // declared but uninitialized
char c;

// read an int, store 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);
```

```
int i;      // declared but uninitialized

...

scanf("%d", i); // DANGER!!!
```

Do-While Loops

C Code

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

Goto Version

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

Do-While Loop Compilation

Goto Version

```
long pcount_goto
  (unsigned long x) {
  long result = 0;
 loop:
  result += x & 0x1;
  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 >>= 1
    jne     .L2          #  if (x) goto loop
    rep; ret
```

While Loops: Jump-to-Middle

While version

```
while (Test)
  Body
```

Goto Version

```
  goto test;
loop:
  Body
test:
  if (Test)
    goto loop;
done:
```

Jump-to-Middle Example

C Code

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

Jump to Middle

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

While Loops: Guarded Do

While version

```
while (Test)
  Body
```

Do-While Version

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

Goto Version

```
  if (!Test)
    goto done;
loop:
  Body
  if (Test)
    goto loop;
done:
```

Guarded Do Example

C Code

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

Do-While Version

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

For Loops

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

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

Switch Statements

```
void print_feedback(char letter_grade) {
  switch (letter_grade) {
    case 'A':
      printf("Excellent!\n");
      break;
    case 'B':
      printf("Well done!\n");
      break;
    case 'C':
      printf("You passed!\n");
      break;
    case 'D':
      printf("Better try again\n");
      break;
    case 'F':
      printf("Uh oh\n");
      break;
    default:
      printf("Invalid grade\n");
      break;
  }
}
```

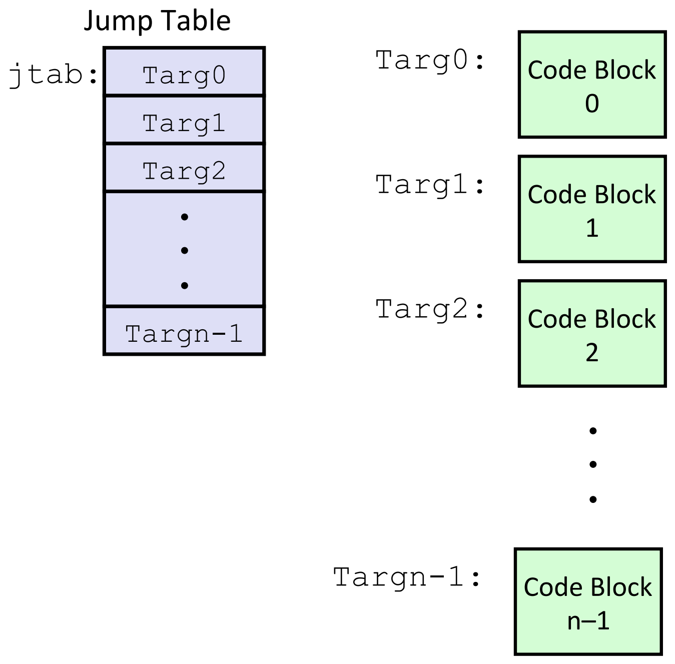
Switch Fall Through

```
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;
        /* Fall Through */
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Jump Tables

Switch Form

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



Translation (Extended C)

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

Switch Example

```
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;
        /* Fall Through */
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Jump table

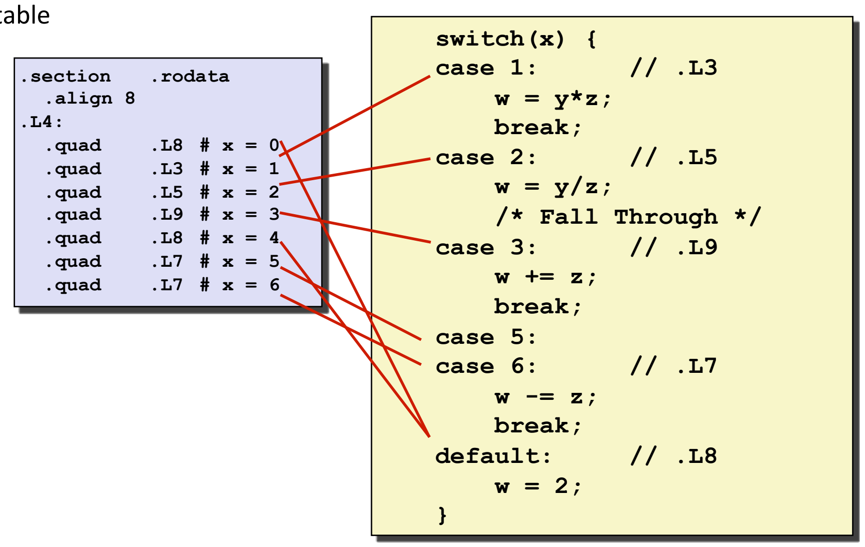
```
.section    .rodata
  .align 8
.L4:
  .quad     .L8  # x = 0
  .quad     .L3  # x = 1
  .quad     .L5  # x = 2
  .quad     .L9  # x = 3
  .quad     .L8  # x = 4
  .quad     .L7  # x = 5
  .quad     .L7  # x = 6
```

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

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

Example Jump Table

Jump table



```
.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(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

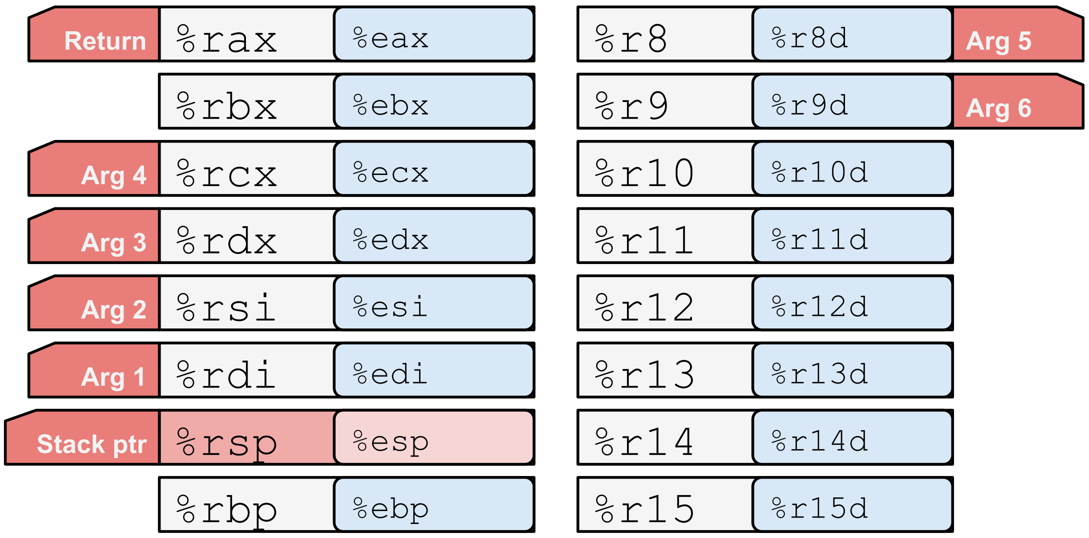
```c
long w = 1;
switch(x) {
case 1:       // .L3
    w = y*z;
    break;
case 2:       // .L5
    w = y/z;
    /* Fall Through */
case 3:       // .L9
    w += z;
    break;
case 5:
case 6:       // .L7
    w -= z;
    break;
default:      // .L8
    w = 2;
}
return w;
```

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

```
switch_eg:
    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
```

Procedure Call Registers


Return
%rax
%eax
%rbx
%ebx
Arg 4
%rcx
%ecx
Arg 3
%rdx
%edx
Arg 2
%rsi
%esi
Arg 1
%rdi
%edi
Stack ptr
%rsp
%esp
%rbp
%ebp
%r8
%r8d
Arg 5
%r9
%r9d
Arg 6
%r10
%r10d
%r11
%r11d
%r12
%r12d
%r13
%r13d
%r14
%r14d
%r15
%r15d