Machine-Level Programming II: Control

Today

- Control: Condition codes
- Conditional branches
- Loops
- **Switch Statements**

```
void foo(long x, long n, long *dest)
{
 long val;

 switch (n) {

  case 0:
    val = x*13; break;

  case 2:
    val = x+10; /*Fall through*/

  case 3:
    val = x+11; break;

  case 4:
  case 6:
    val = x*x; break;

  default:
    val=0;
 }
 *dest = val;
}
```

- Can be translated to if-else
- Problem: not efficient
- Solution: use a jump table

```
void foo(long x, long n, long *dest)
{
 long val;

 switch (n) {

  case 0:
    val = x*13; break;

  case 2:
    val = x+10; /*Fall through*/

  case 3:
    val = x+11; break;

  case 4:
  case 6:
    val = x*x; break;

  default:
    val=0;
 }
 *dest = val;
}
```

```
void foo(long x, long n, long *dest)
 {
  long val;

  /* Table of code pointers */
  static void *jt[7] = {
    &&loc_A, &&loc_def, &&loc_B,
    &&loc_C, &&loc_D, &&loc_def,
    &&loc_D};

  unsigned long index = n;
  if (index > 6) goto loc_def;
  goto *jt[index];

  loc_A: /* Case 0 */
    val = x*13; goto done;

  loc_B: /* Case 2 */
    val = x+10; /* Fall through */

  loc_C: /* Case 3 */
    val = x+11; goto done;


  loc_D: /* Cases 4, 6 */
    val = x*x; goto done;

  loc_def: /* Default case */
    val=0;

  done: *dest = val;
 }
```

Jump Table Structure

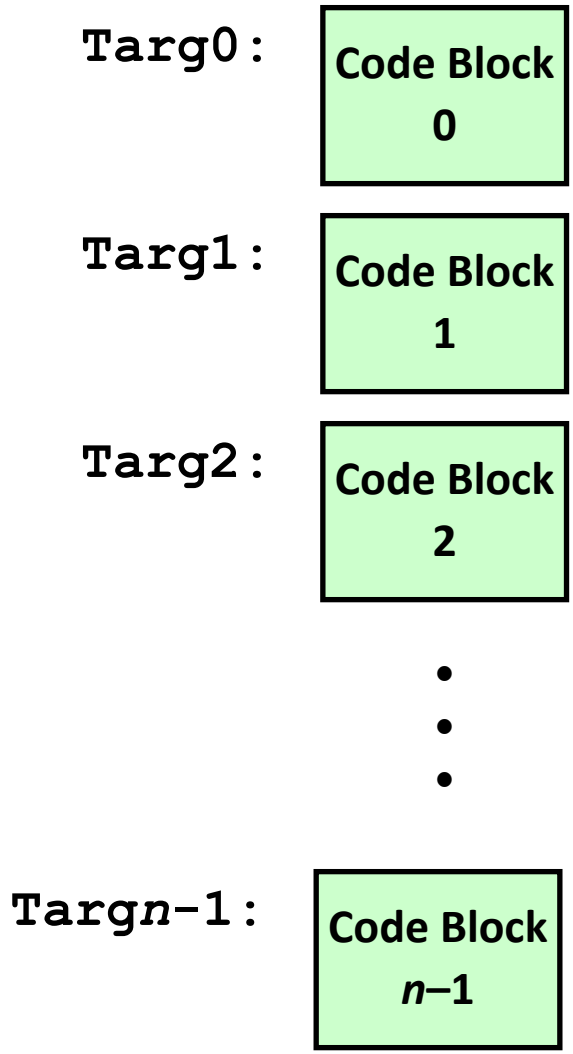
Switch Form

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

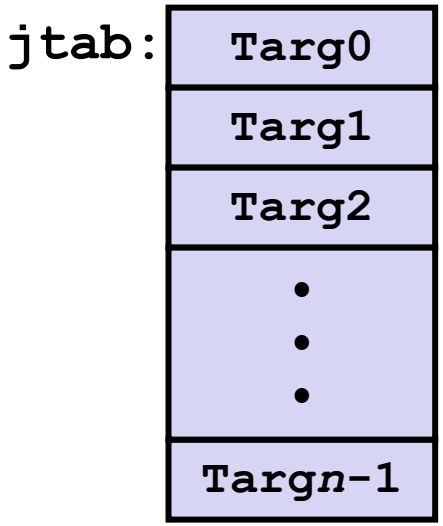
Translation (Extended C)

```
goto *JTab[x];
```

Jump Targets



Jump Table



Switch Statement 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;
}
```

Switch Statement Example

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

Setup:

```
switch_eg:
    movq    %rdx, %rcx
    cmpq    $6, %rdi    # x:6
    ja      .L8
    jmp     *.L4(,%rdi,8)
```

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

Switch Statement Example

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

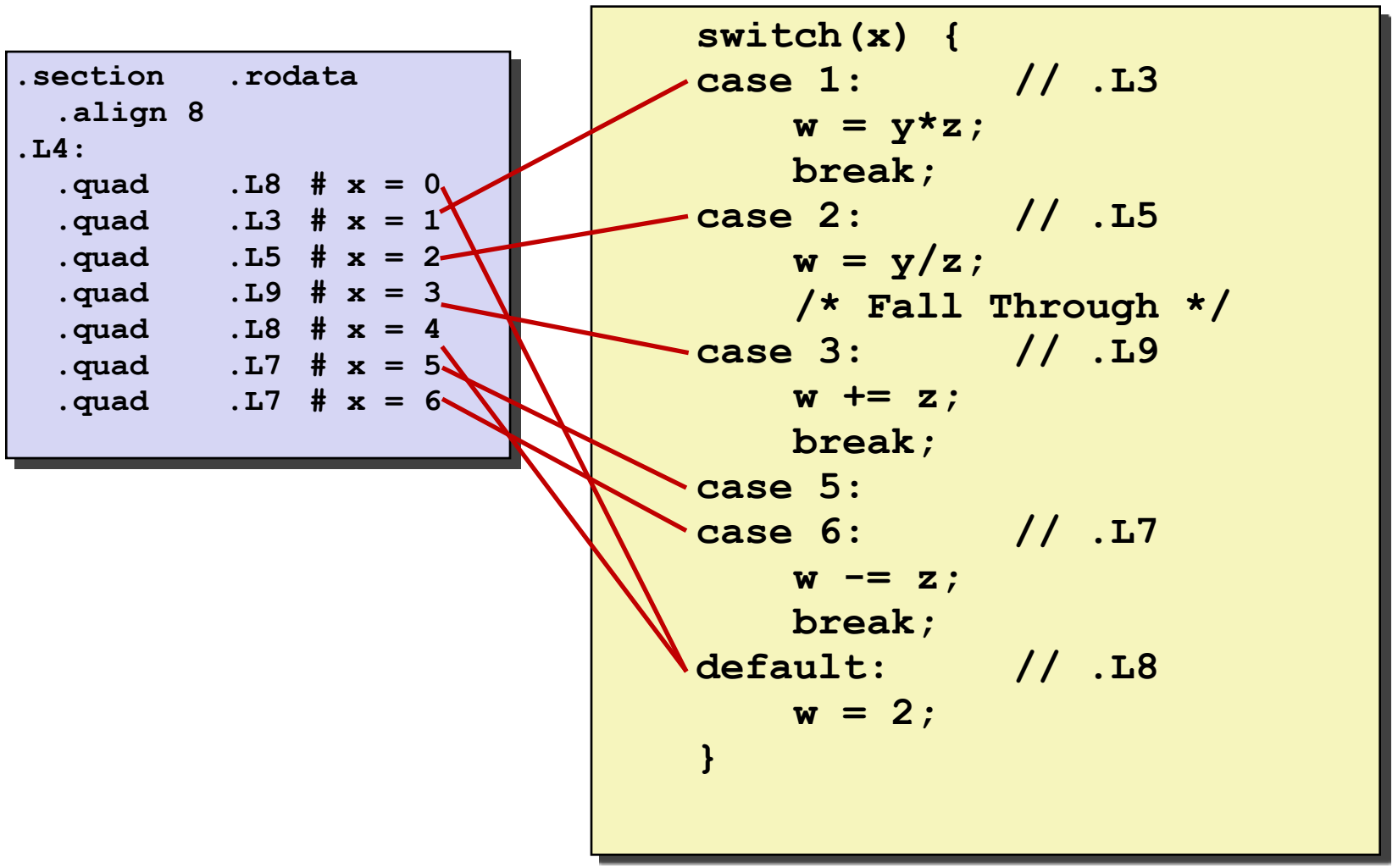
Setup:

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

Indirect jump

Jump Table

Jump table



Assembly Setup Explanation

Table Structure

- Each target requires 8 bytes
- Base address at **`.L4`**

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
```

Jumping

- **Direct:** **`jmp .L8`**
- Jump target is denoted by label **`.L8`**

- **Indirect:** **`jmp *.L4(,%rdi,8)`**
- Start of jump table: **`.L4`**
- Must scale by factor of 8 (addresses are 8 bytes)
- Fetch target from effective Address **`.L4 + x*8`**
  - Only for $0 \le \mathbf{x} \le 6$

Code Blocks (x == 1)

```
case 1:          // .L3
        w = y*z;
        break;
   . . .
```

```
.L3:
   movq     %rsi, %rax   # y
   imulq    %rdx, %rax   # y*z
   ret
```

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

Code Blocks (x == 2, x == 3)

```
long w = 1;
   . . .
switch(x) {
   . . .
case 2:
    w = y/z;
    /* Fall Through */
case 3:
    w += z;
    break;
   . . .
}
```

```
.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
```

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

Code Blocks (x == 5, x == 6, default)

```
  switch(x) {
    . . .
    case 5:  // .L7
    case 6:  // .L7
        w -= z;
        break;
    default: // .L8
        w = 2;
}
```

```
.L7:                    # Case 5,6
  movl   $1, %eax       #  w = 1
  subq   %rdx, %rax     #  w -= z
  ret
.L8:                    # Default:
  movl   $2, %eax       #  2
  ret
```

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