```
long switch_eg (unsigned
   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

- **Multiple case labels**
  - Here: 5, 6
- **Fall through cases**
  - Here: 2
- **Missing cases**
  - Here: 4
- **Lots to manage, we need a *jump table***

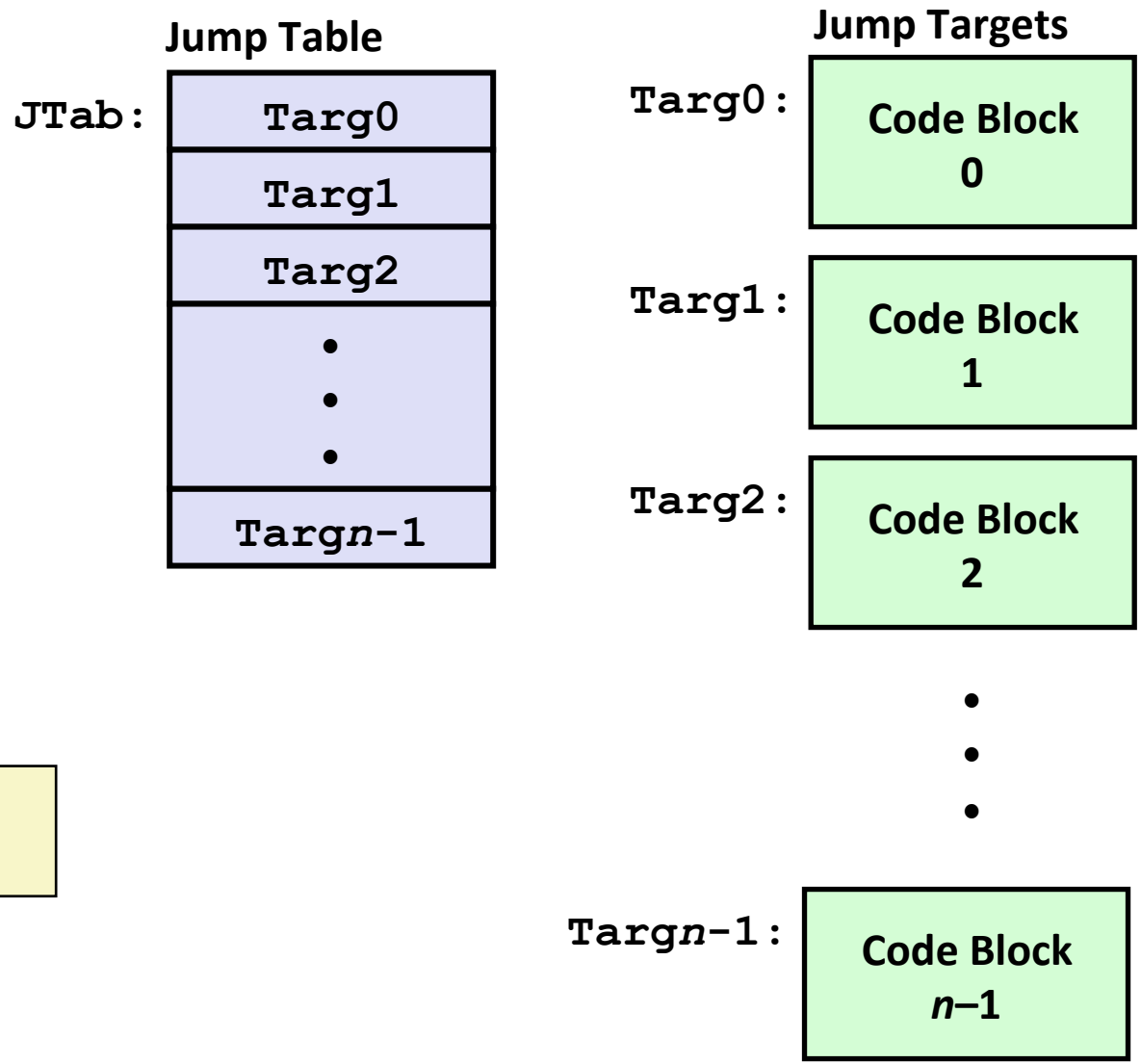
Jump Table Structure

Switch Form

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

Approximate Translation

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



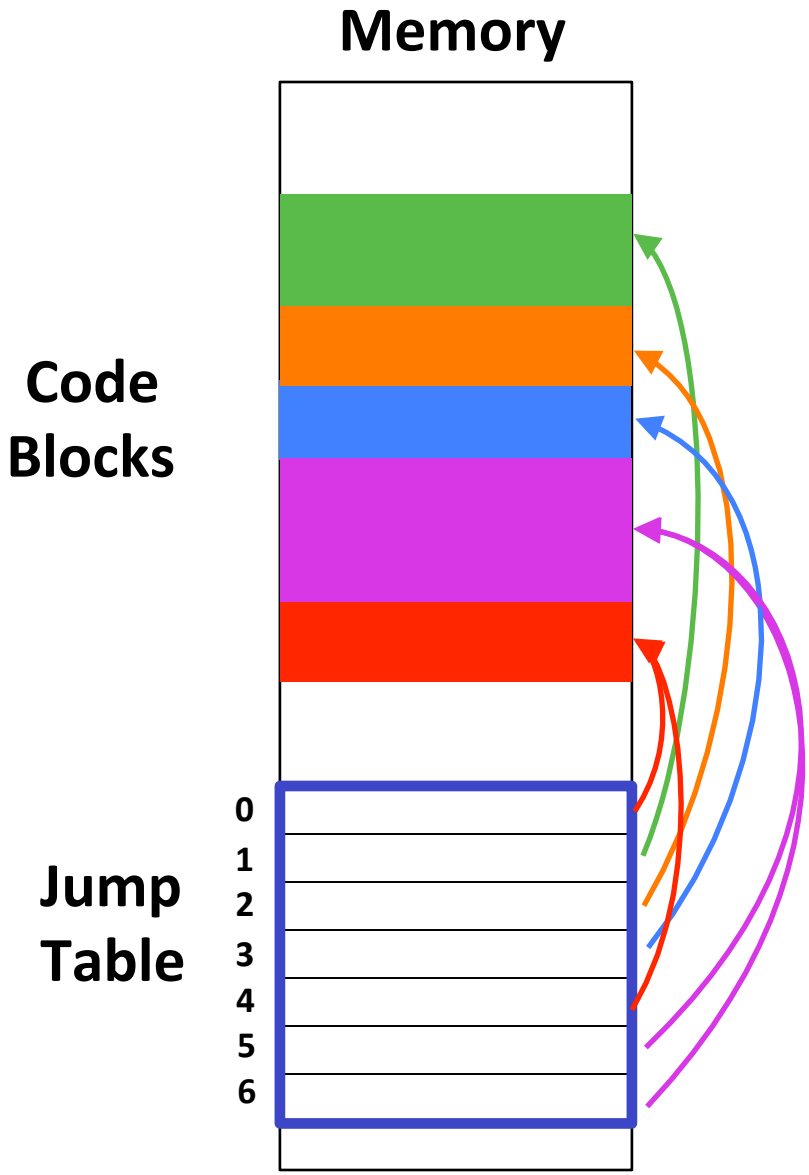
Jump Table Structure

C code:

```
switch(x) {
  case 1: <some code>
          break;
  case 2: <some code>
  case 3: <some code>
          break;
  case 5:
  case 6: <some code>
          break;
  default: <some code>
}
```

We can use the jump table when x <= 6:

```
if (x <= 6)
  target = JTab[x];
  goto *target;
else
  goto default;
```



Jump Table

Jump table

```
.section .rodata
   .align 4
.L62:
  .long   .L61   # x = 0
  .long   .L56   # x = 1
  .long   .L57   # x = 2
  .long   .L58   # x = 3
  .long   .L61   # x = 4
  .long   .L60   # x = 5
  .long   .L60   # x = 6
```

```
    switch(x) {
    case 1:       // .L56
        w = y*z;
        break;
    case 2:       // .L57
        w = y/z;
        /* Fall Through */
    case 3:       // .L58
        w += z;
        break;
    case 5:
    case 6:       // .L60
        w -= z;
        break;
    default:      // .L61
        w = 2;
    }
```

Switch Statement Example (IA32)

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

Setup:

```
switch_eg:
    pushl %ebp              # Setup
    movl  %esp, %ebp        # Setup
    pushl %ebx              # Setup
    movl  $1, %ebx          # w = 1
    movl  8(%ebp), %edx     # edx = x
    movl  16(%ebp), %ecx    # ecx = z
    cmpl  $6, %edx
    ja    .L61
    jmp   *.L62(,%edx,4)
```

Jump table

```
.section .rodata
   .align 4
.L62:
  .long   .L61   # x = 0
  .long   .L56   # x = 1
  .long   .L57   # x = 2
  .long   .L58   # x = 3
  .long   .L61   # x = 4
  .long   .L60   # x = 5
  .long   .L60   # x = 6
```

Translation?

Switch Statement Example (IA32)

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

Setup:

```
switch_eg:
    pushl %ebp              # Setup
    movl  %esp, %ebp        # Setup
    pushl %ebx              # Setup
    movl  $1, %ebx          # w = 1
    movl  8(%ebp), %edx     # edx = x
    movl  16(%ebp), %ecx    # ecx = z
    cmpl  $6, %edx          # x:6
    ja    .L61              # if > goto default
    jmp   *.L62(,%edx,4)    # goto JTab[x]
```

Indirect jump 

Jump table

```
.section .rodata
   .align 4
.L62:
  .long   .L61   # x = 0
  .long   .L56   # x = 1
  .long   .L57   # x = 2
  .long   .L58   # x = 3
  .long   .L61   # x = 4
  .long   .L60   # x = 5
  .long   .L60   # x = 6
```

Assembly Setup Explanation

- **Table Structure**
  - Each target requires 4 bytes
  - Base address at **`.L62`**

Jump table

```
.section .rodata
   .align 4
.L62:
  .long   .L61   # x = 0
  .long   .L56   # x = 1
  .long   .L57   # x = 2
  .long   .L58   # x = 3
  .long   .L61   # x = 4
  .long   .L60   # x = 5
  .long   .L60   # x = 6
```

- **Jumping: different address modes for target**

  Direct: **`jmp .L61`**
  - Jump target is denoted by label **`.L61`**

  Indirect: **`jmp *.L62(,%edx,4)`**
  - Start of jump table: **`.L62`**
  - Must scale by factor of 4 (labels are 32-bits = 4 bytes on IA32)
  - Fetch target from effective address **`.L62 + edx*4`**
    - **`target = JTab[x]; goto *target;`** (only for $0 \le$ **`x`** ≤ 6)

Code Blocks (Partial)

```
switch(x) {
  . . .
case 2:       // .L57
    w = y/z;
    /* Fall Through */
case 3:       // .L58
    w += z;
    break;
  . . .
default:      // .L61
    w = 2;
}
```

```
.L61:  // Default case
  movl  $2, %ebx     # w = 2
  movl  %ebx, %eax   # Return w
  popl  %ebx
  leave
  ret
.L57:  // Case 2:
  movl  12(%ebp), %eax  # y
  cltd                  # Div prep
  idivl %ecx            # y/z
  movl  %eax, %ebx # w = y/z
# Fall through
.L58:  // Case 3:
  addl  %ecx, %ebx # w+= z
  movl  %ebx, %eax # Return w
  popl  %ebx
  leave
  ret
```

Code Blocks (Rest)

```
switch(x) {
case 1:       // .L56
    w = y*z;
    break;
  . . .
case 5:
case 6:       // .L60
    w -= z;
    break;
  . . .
}
```

```
.L60: // Cases 5&6:
   subl  %ecx, %ebx  # w -= z
   movl  %ebx, %eax  # Return w
   popl  %ebx
   leave
   ret
.L56: // Case 1:
   movl  12(%ebp), %ebx # w = y
   imull %ecx, %ebx     # w*= z
   movl  %ebx, %eax  # Return w
   popl  %ebx
   leave
   ret
```

IA32 Object Code

- **Setup**
  - Label **`.L61`** becomes address **`0x08048630`**
  - Label **`.L62`** becomes address **`0x080488dc`**

Assembly Code

```
switch_eg:
  . . .
   ja     .L61             # if > goto default
   jmp    *.L62(,%edx,4)   # goto JTab[x]
```

Disassembled Object Code

```
08048610 <switch_eg>:
 . . .
08048622:  77 0c                      ja     8048630
08048624:  ff 24 95 dc 88 04 08       jmp    *0x80488dc(,%edx,4)
```

IA32 Object Code (cont.)

- **Jump Table**
  - Doesn't show up in disassembled code
  - Can inspect using GDB

```
gdb asm-cntl
(gdb) x/7xw 0x080488dc
```

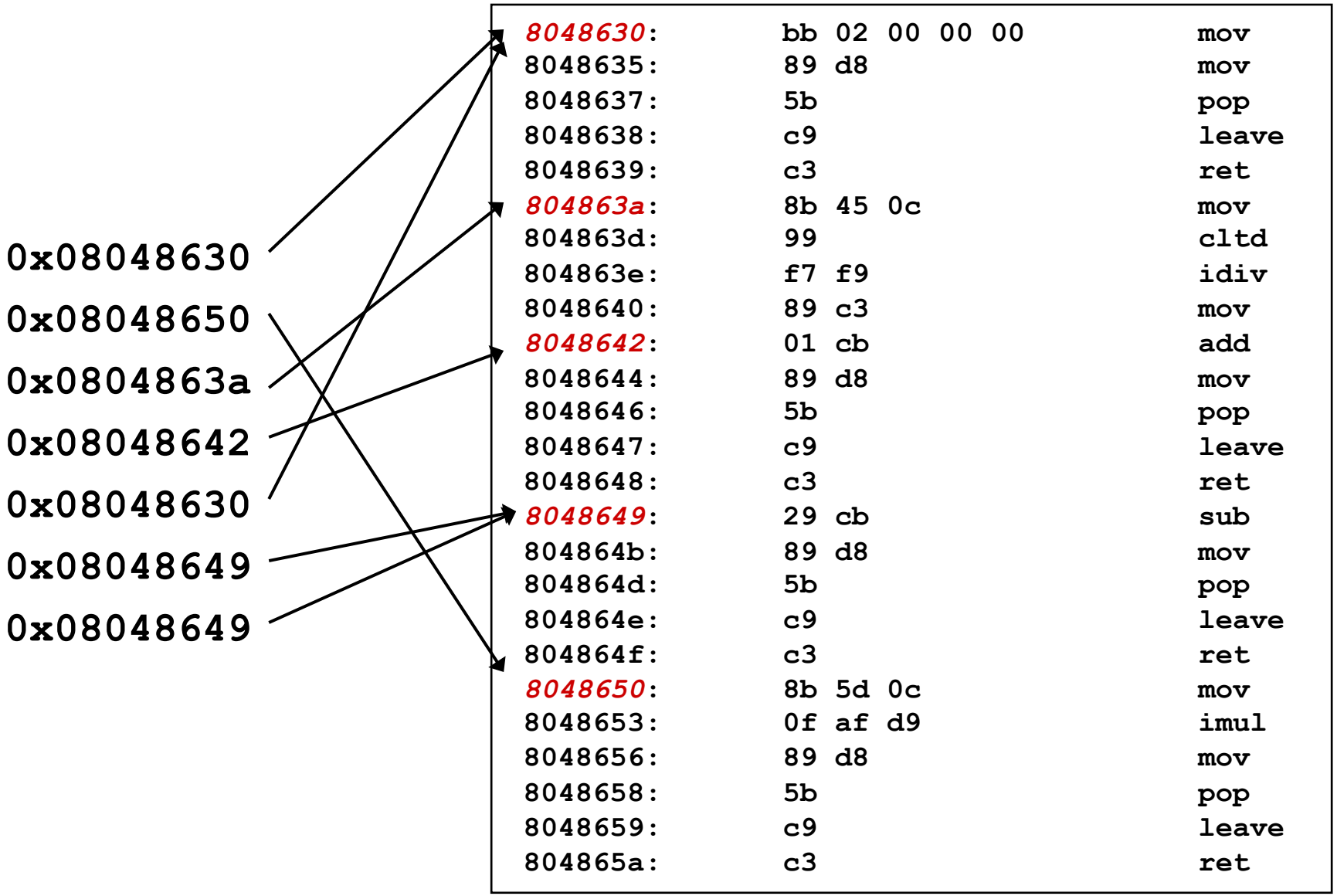
    - E*x*amine 7 he*x*adecimal format "*w*ords" (4-bytes each)
    - Use command "**`help x`**" to get format documentation

```
0x080488dc:
  0x08048630
  0x08048650
  0x0804863a
  0x08048642
  0x08048630
  0x08048649
  0x08048649
```

Disassembled Targets

```
8048630:      bb 02 00 00 00          mov    $0x2,%ebx
8048635:      89 d8                   mov    %ebx,%eax
8048637:      5b                      pop    %ebx
8048638:      c9                      leave
8048639:      c3                      ret
804863a:      8b 45 0c                mov    0xc(%ebp),%eax
804863d:      99                      cltd
804863e:      f7 f9                   idiv   %ecx
8048640:      89 c3                   mov    %eax,%ebx
8048642:      01 cb                   add    %ecx,%ebx
8048644:      89 d8                   mov    %ebx,%eax
8048646:      5b                      pop    %ebx
8048647:      c9                      leave
8048648:      c3                      ret
8048649:      29 cb                   sub    %ecx,%ebx
804864b:      89 d8                   mov    %ebx,%eax
804864d:      5b                      pop    %ebx
804864e:      c9                      leave
804864f:      c3                      ret
8048650:      8b 5d 0c                mov    0xc(%ebp),%ebx
8048653:      0f af d9                imul   %ecx,%ebx
8048656:      89 d8                   mov    %ebx,%eax
8048658:      5b                      pop    %ebx
8048659:      c9                      leave
804865a:      c3                      ret
```

Matching Disassembled Targets



Question

- **Would you implement this with a jump table?**

```
switch(x) {
  case 0:      <some code>
               break;
  case 10:     <some code>
               break;
  case 52000:  <some code>
               break;
  default:     <some code>
               break;
}
```

- **Probably not:**
  - Don't want a jump table with 52001 entries (too big)