

Machine Programming Control

CSC 235 - Computer Organization

References

- Slides adapted from CMU

Outline

- Control: condition codes
- Conditional branches
- Loops
- Switch statements

Processor State (x86-64, Partial)

- Information about currently executing program
 - Register contents (temporary data)
 - Current code control point (contents of the instruction pointer `%rip`)
 - Status of recent operations (condition codes)

Condition Codes (Implicit Setting)

- Single bit registers that are set as a side effect of arithmetic operations
 - **CF** (carry flag): set if carry/borrow out from most significant bit
 - **ZF** (zero flag): set if Dest is zero
 - **SF** (sign flag): set if Dest less than zero
 - **OF** (overflow flag): set if two's complement overflow
- Note: condition codes are **not** set by the `leaq` instruction.

Condition Codes (Explicit Setting)

- Condition codes can be explicitly set with the compare instruction
 - Syntax: `cmpq src1, src2`
 - Semantics: computes `subq src1, src2` without setting destination
 - **CF**: set if carry/borrow out from most significant bit
 - **ZF**: set if `src1` is equal to `src2`
 - **SF**: set if `(src2 - src1)` is less than zero
 - **OF**: set if two's complement overflow

Condition Codes (Explicit Setting)

- Condition codes can be explicitly set with the test instruction
 - Syntax: `testq src1, src2`
 - Semantics: computes `andq src1, src2` without setting destination
 - **ZF**: set if `src1 & src2`)
 - **SF**: set if `(src2 & src1)` is less than zero
 - Useful to have one of the operands be a mask

Condition Codes (Explicit Reading)

- Condition codes can be explicitly read with the set instructions
 - Syntax: `setX Dest`
 - Semantics: set low-order byte of Dest to 0 or 1; does not alter remaining 7 bytes of Dest
 - Typically used with the `movzbq` instruction which zeros out all but the low-order byte
 - Example

```
cmpq    %rsi, %rdi # compare      setg
%al          # set low-order byte of %rax
movzbq %al, %rax  # zero rest of %rax
```


Condition Codes (Explicit Reading)

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

Example: set1 (signed <)

- Condition: $SF \wedge OF$

SF	OF	$SF \wedge OF$	Implication
0	0	0	No overflow, so SF implies no less than
1	0	1	No overflow, so SF implies greater than
0	1	1	Overflow, so SF implies negative overflow (<)
1	1	0	Overflow, so SF implies positive overflow (not <)

Jumping

- Jump instructions conditionally change the contents of the instruction pointer (`%rip`) based on implicitly reading the condition codes
 - Syntax: `jX label`
 - A label is a symbolic name for the address of an instruction
 - The syntax for a label is a string beginning with a letter, dot, or underscore followed by any number of digits, letters, dollar signs, and underscores where the last character must be a colon.
 - Used for control flow

Jumping

Instruction	Condition	Description
jmp	1	Unconditional
je	ZF	Equal/Zero
jne	\sim ZF	Not equal/Not Zero
js	SF	Negative
jns	\sim SF	Nonnegative
jg	\sim (SF ^ OF) & \sim ZF	Greater (signed)
jge	\sim (SF ^ OF)	Greater or Equal (signed)
j1	SF ^ OF	Less (signed)
jle	(SF ^ OF) ZF	Less or Equal (signed)
ja	\sim CF & \sim ZF	Above (unsigned)
jb	CF	Below (unsigned)

Conditional Branch Example

- C code

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

- Generate assembly with

```
gcc -Og -S absdiff.c
```

Conditional Branch Example

■ Assembly

```
# %rdi: argument x
# %rsi: argument y
# %rax: return value
absdiff:
    cmpq    %rsi, %rdi
    jle    .L2
    movq    %rdi, %rax
    subq    %rsi, %rax
    ret
.L2:
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

Conditional Branch Example

- C allows goto and labels

```
long absdiff (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 Moves

- Conditional move instructions
 - Syntax: `cmovX src, dest`
 - Semantics: execute the move instruction based on the condition codes
- Why?
 - Branches (jumps) are disruptive to instruction flow through pipelines
 - Conditional moves do not require control transfer

Conditional Move Example

- Generate assembly with: `gcc -O2 -S absdiff.c`
- Assembly version with conditional move

```
# %rdi: argument x
# %rsi: argument y
# %rax: return value
absdiff:
    movq    %rsi, %rax    # y
    movq    %rdi, %rdx    # x
    negq    %rax          # -y
    subq    %rsi, %rdx    # x - y
    subq    %rdi, %rax    # -y - x
    cmpq    %rsi, %rdi
    cmovg   %rdx, %rax
    ret
```

Bad Cases for Conditional Move

- Expensive computations, e.g. `val = test(x) ? f(x) : g(x);`
 - Both values get computed regardless of the function complexity
- Risky computations, e.g. `val = p ? *p : 0;`
 - Both values get computed, but may have undesirable effects
- Computations with side effects, e.g. `val = x > 0 ? x*=7 : x+=3;`
 - Both values get computed, but have side effects

Condition Code Example

- Condition codes set from a sequence of instructions

Instruction	%rax	SF	CF	OF	ZF
xorq %rax, %rax	0x0000 0000 0000 0000	0	0	0	1
subq \$1, %rax	0xFFFF FFFF FFFF FFFF	1	1	0	0
cmpq \$2 %rax	0xFFFF FFFF FFFF FFFF	1	0	0	0
setl %al	0xFFFF FFFF FFFF FF01	1	0	0	0
movzbq %al, %rax	0x0000 0000 0000 0001	1	0	0	0

- Note: the setl and movzbq do not modify condition codes

Do While Loop Example

- C code to count number of 1s in argument x

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

Do While Loop Example

- C code with goto

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

Do While Loop Compilation

■ Assembly

```
    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
    ret
```

General Do While Translation

- C Code

```
do {  
    /* body */  
} while (/* test */)
```

- Goto version

```
loop:  
    /* body */  
    if (/* test */)   
        goto loop
```

General While Translation #1

- “Jump to middle” translation (used with `-Og` flag)
- While version

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

- Goto version

```
    goto test;  
loop:  
    /* body */  
test:  
    if (/* test */)  
        goto loop;  
done:
```


While Loop Example #1

- C code

```
long pcount_while (unsigned long x) {
    long result = 0;
    while (x) {
        result += x & 0x01;
        x >>= 1;
    }
    return result
}
```

While Loop Example #1

- Jump to middle

```
long pcount_while (unsigned long x) {  
    long result = 0;  
    goto test;  
loop:  
    result += x & 0x01;  
    x >>= 1;  
test:  
    if (x) goto loop;  
    return result;  
}
```

General While Translation #2

- “Do while” conversion (used with -O1 flag)
- While version

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

- Do while version

```
if (! /* test */) {  
    goto done;  
}  
do {  
    /* body */  
} while (/* test */)   
done:
```

While Loop Example #2

- Do while version

```
long pcount_while (unsigned long x) {  
    long result = 0;  
    if (!x) goto done;  
loop:  
    result += x & 0x01;  
    x >>= 1;  
    if (x) goto loop;  
done:  
    return result;  
}
```

For Loop

- C code

```
#define WSIZE 8*sizeof(int)
long popcount_for (unsigned long x) {
    size_t i;
    long result = 0;
    for (i = 0; i < WSIZE; i++) {
        unsigned bit = (x >> i) & 0x01;
        result += bit;
    }
    return result;
}
```

For Loop Translation

- Convert to while loop
- For version

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

- While version

```
/* init */;  
while (/* test */) {  
    /* body */  
    /* update */  
}
```

For While Conversion

- Code

```
long pcount_for_while(unsigned long x) {
    size_t i;
    long result = 0;
    i = 0;
    while (i < WSIZE) {
        unsigned bit = (x >> i) & 0x01;
        result += bit;
        i++;
    }
    return result;
}
```

For Loop Do-While Conversion

■ Code

```
long pcount_for_goto_dw(unsigned long x) {
    size_t i;
    long result = 0;
    i = 0;
loop:
    {
        unsigned bit = (x >> i) & 0x01;
        result += bit;
    }
    i++;
    if (i < WSIZE) {
        goto loop;
    }
done:
    return result;
}
```


Switch Statements Example

```
long my_switch(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 Statements

■ Switch Form

```
switch (x) {  
    case val_0:  
        /* block 0 */  
    case val_1:  
        /* block 1 */  
  
    ...  
  
    case val_n_minus_1:  
        /* block n minus 1 */  
}
```

Jump Table Structure

- Jump Targets

```
jtab: target_0
      target_1
      ...
      target_n_minus_1
```

- Jump Targets

```
target_0: /* code block 0 */
```

```
target_1: /* code block 1 */
```

```
target_n_minus_1: /* code block n minus 1 */
```

Switch Statement Example

- Assembly setup

```
my_switch:
```

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

Switch Statement Example

- 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
```

- Each target requires 8 bytes

Switch Statement Example

- Code block (`x == 1`)
- Assembly code

```
.L3:                # Case 1
    movq    %rsi, %rax # y
    imulq   %rdx, %rax # y*z
    ret
```

Switch Statement Example

- Code block (`x == 2 || x == 3`)
- Assembly code

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

Switch Statement Example

- Code block (`x == 5 || x == 6`)
- Assembly code

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


Summarizing

- C Control
 - if-then-else
 - do-while
 - while, for
 - switch
- Assembler Control
 - Conditional jump
 - Conditional move
 - Indirect jump (via jump tables)
 - Compiler generates code sequence to implement more complex control

Summary

- Control: condition codes
- Conditional branches
- Loops
- Switch statements