Machine Level Programming: Control

CSCI 237: Computer Organization
14th Lecture, Monday, Oct. 7

Kelly Shaw

2

2

Administrative Details

- Lab #3 checkpoint due Tuesday at 11pm
  - Any questions?
- Read CSAPP 3.7-3.8

3

3

Last Time: Machine-Level Programming: Control

- Arithmetic and Logic Instructions
- gdb commands for bomb portion of lab
- Intro to data-dependent control
  - Condition codes
  - Conditional branches
  - Conditional data
  - Loops
  - Switch Statements

4

4

Today: Machine-Level Programming: Control

- Intro to data-dependent control
  - Condition codes
  - Conditional branches
  - Conditional data
  - Loops
  - Switch Statements

5

5

Reading Condition Codes

- Three ways to "access" condition codes in assembly. We will go over them in detail:
  1. Operations that set a byte to 0/1 based on some combination of the condition codes
  2. Operations that "jump" to some part of program based on condition codes
  3. Operations that transfer data only if some condition codes are set

We are going to do a lot of conversion between C and assembly, and between assembly and C. The practice problems and examples in the textbook are really helpful!

6

6

Reading Condition Codes 2: `jmp` instrs

- Before we talk about instructions that alter control flow in assembly, let's review the control flow constructs in C
  - if (condition)
  - while (condition)
  - do { } while;
  - for (init; condition; post)
  - switch
  - goto!!
  - functions

Simplest and least familiar, but analogous to how we will write assembly. Let's review!

7

7

Expressing control flow constructs with goto

- C allows **`goto`** statement (typically considered bad programming style, but sometimes useful!)
  - Control jumps to a position designated by the target label
- We can convert many control flow constructs to equivalent C code with **`goto`** statements

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

```
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;
}
```

8

8

Jumping in ASM is the C goto analog

- **`jX`** *target*
  - Jump to different part of code depending on condition codes
  - target is a *Label* or **Operand*

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 <)

9

9

Conditional Branch Example (with jumps)

- Generation

```
> gcc –Og –S –fno-if-conversion absdiff.c
```

we'll come back to this.

```
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

10

10

Practice on Your Own

- Rewrite the following assembly code into C code:

```
fcn:
    movq    $4, %rsi
    cmpq    %rsi, %rdi
    sete    %al
    movzbl %al, %eax
    ret
```

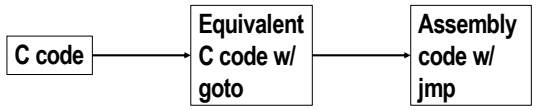
Note: Function return values are placed in the register %rax

11

11

jX Is a Powerful Tool

- We can convert many C control flow constructs to equivalent C code that contains goto statements
- We can convert C code with goto statements into ASM with **jX**
- We will spend much of this and next lecture doing this mapping



- But first, we will talk about conditional moves

12

12

General Conditional Expression Translation (Using Branches)

C Code

```
val = Test ? Then_Expr : Else_Expr;
```

```
val = x > y ? x-y : y-x;
```

Goto Version

```
  ntest = !Test;
  if (ntest) goto Else;
  val = Then_Expr;
  goto Done;
Else:
  val = Else_Expr;
Done:
  . . .
```

Conversion strategy:

- Create separate code regions for then & else expressions
- Execute appropriate one

13

13

General Conditional Expression Translation (Using Conditional Moves)

- Conditional Move Instructions
  - Instruction supports:
    if (Test) Dest ← Src
  - Supported in post-1995 x86 processors
  - GCC tries to use them
    - But, only when known to be safe
- Why?
  - Branches (jumps) are very disruptive to instruction flow through pipelines
  - Performance relies on good predictions
  - Conditional moves do not require control transfer!

C Code

```
val = Test
    ? Then_Expr
    : Else_Expr;
```

"Goto" Version

```
  result = Then_Expr;
  eval = Else_Expr;
  nt = !Test;
  if (nt) result = eval;
  return result;
```

14

14

Conditional Move Example

Generation

```
> gcc -O1 -S absdiff.c
```

Notice the compile flags!

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

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

```
absdiff:
    movq    %rdi, %rax  # x
    subq    %rsi, %rax  # result = x-y
    movq    %rsi, %rdx  # y
    subq    %rdi, %rdx  # eval = y-x
    cmpq    %rsi, %rdi  # x:y
    cmovle  %rdx, %rax  # if <=, result = eval
    ret
```

15

15

Bad Cases for Conditional Move

Expensive Computations

```
val = Test(x) ? Hard1(x) : Hard2(x);
```

- Both values get computed
- Only makes sense when computations are very simple

Bad Performance

Risky Computations

```
val = p ? *p : 0;
```

- Both values get computed
- May have undesirable effects

Unsafe

Computations with side effects

```
val = x > 0 ? x*=7 : x+=3;
```

- Both values get computed
- Must be side-effect free

Illegal

16

16

Today: Machine-Level Programming: Control

- Intro to data-dependent control
  - Condition codes
  - Conditional branches
  - Conditional data
  - Loops
  - Switch Statements

17

17

"Do-While" Loop Example

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;
}
```

- Count number of 1's in argument **x** (bitcount or "popcount")
- Use conditional branch to either continue looping or to exit loop

18

18

"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
```

19

19