Machine Level Programming: Control

Computer Systems Organization (Spring 2016)
CSCI-UA 201, Section 2

Instructor: Joanna Klikowska

Slides adapted from
Randal E. Bryant and David R. O’Hallaron (CMU)
Mohamed Zahran (NYU)

Condition Codes

Processor State (x86-64, first look)

- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
  - Location of current code control point (%rip, …)
  - Status of recent tests (CF, ZF, SF, OF)

Condition Codes (Implicit Setting)

- Single bit registers
  - CF Carry Flag (for unsigned)
  - SF Sign Flag (for signed)
  - ZF Zero Flag OF Overflow Flag (for signed)

- Implicitly set (think of it as side effect) by arithmetic operations
  - Example: addq src, dest \rightarrow t = a + b
  - CF set if carry out from most significant bit (unsigned overflow)
  - ZF set if t == 0
  - SF set if t < 0 (as signed)
  - OF set if two’s-complement (signed) overflow
  - (a<0 \&\& b<0 \&\& t<0) || (a<0 \&\& b<0 \&\& t>0)

- Not set by leaq instruction

Condition Codes (Explicit Setting - cmpq)

- Explicit Setting by Compare Instruction
  - cmpq src2, src1
    - cmpq b, a \rightarrow like computing a-b without setting destination

  - CF set if carry out from most significant bit (used for unsigned comparisons)
  - ZF set if a == b
  - SF set if (a-b) < 0 (as signed)
  - OF set if two’s-complement (signed) overflow
  - (a<0 \&\& b<0 \&\& (a-b)<0) || (a<0 \&\& b<0 \&\& (a-b)>0)

Condition Codes (Explicit Setting - testq)

- Explicit Setting by Test Instruction
  - testq src2, src1
    - testq b, a \rightarrow like computing a\&\& b without setting destination

    - ZF set when a\&\& b == 0
    - SF set when a\&\& b < 0
Reading Condition Codes

- SetX family of instructions
  - Set low-order byte of destination to 0 or 1 based on combinations of condition codes
  - Does not alter remaining bytes

<table>
<thead>
<tr>
<th>SetX</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sete</td>
<td>ZF</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td>setne</td>
<td>~ZF</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td>sets</td>
<td>SF</td>
<td>Negative</td>
</tr>
<tr>
<td>setns</td>
<td>~SF</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>setg</td>
<td><del>(SF</del>OF)~ZF</td>
<td>Greater (Signed)</td>
</tr>
<tr>
<td>setge</td>
<td><del>(SF</del>OF)</td>
<td>Greater or Equal</td>
</tr>
<tr>
<td>setl</td>
<td>(SF~OF)</td>
<td>Less (Signed)</td>
</tr>
<tr>
<td>setle</td>
<td>(SF~OF)ZF</td>
<td>Less or Equal (Signed)</td>
</tr>
<tr>
<td>seta</td>
<td>(SF~OF)ZF</td>
<td>Above (unsigned)</td>
</tr>
<tr>
<td>setb</td>
<td>CF</td>
<td>Below (unsigned)</td>
</tr>
</tbody>
</table>

x86-64 Integer Registers

We can reference low-order byte.

Conditional Branches

Jumping (in the code)

- 3X family of instructions
  - Jump to different part of code depending on condition codes

<table>
<thead>
<tr>
<th>3X</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jae</td>
<td>1</td>
<td>Unconditional</td>
</tr>
<tr>
<td>jae</td>
<td>ZF</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td>jae</td>
<td>~ZF</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td>jns</td>
<td>SF</td>
<td>Negative</td>
</tr>
<tr>
<td>jns</td>
<td>~SF</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>jg</td>
<td><del>(SF</del>OF)~ZF</td>
<td>Greater (Signed)</td>
</tr>
<tr>
<td>jge</td>
<td><del>(SF</del>OF)</td>
<td>Greater or Equal</td>
</tr>
<tr>
<td>jl</td>
<td>(SF~OF)</td>
<td>Less (Signed)</td>
</tr>
<tr>
<td>jle</td>
<td>(SF~OF)ZF</td>
<td>Less or Equal (Signed)</td>
</tr>
<tr>
<td>ja</td>
<td><del>CF</del>6-ZF</td>
<td>Above (unsigned)</td>
</tr>
<tr>
<td>jbe</td>
<td>CF</td>
<td>Below (unsigned)</td>
</tr>
</tbody>
</table>

Re-Writing Code with goto Statements

- C allows goto statement
  - Jump to position designated by label

Why do that?
- Because the "goto" code is closer to the assembly instructions.
General Conditional Expression Translation

C code:

```c
val = Test ? Then Expr : Else Expr;
```

for example:

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

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

Conditional Move Example

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

Loops

Bad Cases of Conditional Move

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

- expensive computations:
  - both values get calculated
  - only makes sense when computations are very simple

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

- risky computations:
  - may have undesirable side effects (above it is dereferencing a pointer that may be 0)

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

- computations with side effects:
  - both values get calculated
  - must be side-effect free (unlike the example above)

Do...while... loop example

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

- Count number of 1’s in argument x
- Use conditional branch to either continue looping or to exit loop

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 are very disruptive to instruction flow through pipelines
    - Conditional moves do not require control transfer

```c
C Code:
val = Test
  ? Then Expr
  :Else Expr;
```
**do...while... loop compilation**

```c
Long point goto (unsigned long x) {
    long result = 0;
    loop:
        result += x & 0xl;
        x >>= 1;
    if(!x) goto loop;
    return result;
}
```

---

**General do...while... Translation**

```
do
Body
while (Test);
```

```
loop:
Body
if (Test)
go to loop
```

---

**General while Loop Translation (ver. 1)**

```c
while (Test)
Body
```

```
goto test:

loop:

Body

test:
if (Test)
go to loop;
done:
```

- "Jump-to-middleware" translation
- Used with -Og option to gcc

---

**General while Loop Translation (ver. 2)**

```c
while (Test)
Body
```

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

- "Do-while" conversion
- Used with -O1

---

**General while Loop Translation (ver. 1)**

```c
long point while (unsigned long x) {
    long result = 0;
    while (x) {
        result += x & 0xl;
        x >>= 1;
    }
    return result;
}
```

- Compare to do-while version of function
- Initial goto starts loop at test

---

**General while Loop Translation (ver. 1)**

```c
long point while (unsigned long x) {
    long result = 0;
    while (x) {
        result += x & 0xl;
        x >>= 1;
    }
    return result;
}
```

- Compare to do-while version of function
- Initial conditional guards entrance to loop
for Loop Form

General form:

```plaintext
for (Init; Test; Update)  
   Body
```

```c
#define WEIZE 8*sizeof(int)
long pcount_for(unsigned long x)  
{  
    sint_t i;  
    long result = 0;  
    for (i = 0; i < WEIZE; i++)  
        unsigned bit =  
            (x >> i) & 0x1;  
        result += bit;  
    return result;
}
```

for Loop ⇒ while loop

```c
for (Init; Test; Update)  
   Body
```

```c
for (i = 0; i < WEIZE; i++)  
{  
    unsigned bit =  
        (x >> i) & 0x1;  
    result += bit;
}
```

Initial test can be optimized away