Assembly: Loops, switches, functions.

Yipeng Huang

Rutgers University

March 11, 2021
Table of contents

Announcements

Modifying control flow via conditional branch statements
   Jump instructions
   Conditional branch statements

Loop statements
   Compiling for loops to while loops
   Compiling while loops to do-while loops
   Compiling do-while loops to goto statements
   Compiling goto statements to assembly conditional jump instructions

Switch statements
Looking ahead

Class plan

1. Provide mid-semester course feedback at:
   http://sirs.ctaar.rutgers.edu/blue
3. Reading assignment for next two weeks: CS:APP Chapter 3.
Programming Assignment 3: doubleToBin

doubleToBin
  ▶ Reading in the double value.
  ▶ Using pointer casting to obtain a reference solution (which you should not print out).
  ▶ Using assertions to check your solution against the reference solution.
  ▶ Decoding the exp field to get the exponent E.
  ▶ Doing trial division from smallest possible value of the exp field.
Table of contents

Announcements

Modifying control flow via conditional branch statements
  Jump instructions
  Conditional branch statements

Loop statements
  Compiling for loops to while loops
  Compiling while loops to do-while loops
  Compiling do-while loops to goto statements
  Compiling goto statements to assembly conditional jump instructions

Switch statements
Jumping

- **jX Instructions**
  - Jump to different part of code depending on condition codes

<table>
<thead>
<tr>
<th>jX</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp</td>
<td>1</td>
<td>Unconditional</td>
</tr>
<tr>
<td>je</td>
<td>~ZF</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td>jne</td>
<td>~ZF</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td>js</td>
<td>SF</td>
<td>Negative</td>
</tr>
<tr>
<td>jns</td>
<td>~SF</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>jg</td>
<td>~ (SF^OF) &amp; ~ZF</td>
<td>Greater (Signed)</td>
</tr>
<tr>
<td>jge</td>
<td>~(SF^OF)</td>
<td>Greater or Equal (Signed)</td>
</tr>
<tr>
<td>jl</td>
<td>(SF^OF)</td>
<td>Less (Signed)</td>
</tr>
<tr>
<td>jle</td>
<td>(SF^OF)</td>
<td>ZF</td>
</tr>
<tr>
<td>ja</td>
<td>~CF&amp;~ZF</td>
<td>Above (unsigned)</td>
</tr>
<tr>
<td>jb</td>
<td>CF</td>
<td>Below (unsigned)</td>
</tr>
</tbody>
</table>

Figure: Jump instructions. Image credit CS:APP
Branch statements

```c
unsigned long absdiff_ternary ( unsigned long x, unsigned long y ){
    return x<y ? y-x : x-y;
}
```

```c
unsigned long absdiff_if_else ( unsigned long x, unsigned long y ){
    if (x<y) return y-x;
    else return x-y;
}
```

```c
unsigned long absdiff_goto ( unsigned long x, unsigned long y ){
    if (!(x<y)) goto Else;
    return y-x;
    Else:
    return x-y;
}
```

All C functions above translate (-fno-if-conversion) to assembly at right.

```assembly
absdiff_if_else:
    cmpq %rsi, %rdi
    jnb .ELSE
    movq %rsi, %rax
    subq %rdi, %rax
    ret
.ElSE:
    movq %rdi, %rax
    subq %rsi, %rax
    ret
```

Explanation

▷ `cmpq %rsi, %rdi`: Calculates `%rdi - %rsi (x - y)`, sets condition codes.
▷ `jnb .ELSE`: Sets program counter / instruction pointer in `%rip (.ELSE)` if CF flag not set indicating no unsigned overflow.
Table of contents

Announcements

Modifying control flow via conditional branch statements
  Jump instructions
  Conditional branch statements

Loop statements
  Compiling for loops to while loops
  Compiling while loops to do-while loops
  Compiling do-while loops to goto statements
  Compiling goto statements to assembly conditional jump instructions

Switch statements
Compiling for loops to while loops

C loop statements such as for loops, while loops, and do-while loops do not exist in assembly. They are instead constructed from conditional jump statements.

```c
unsigned long count_bits_for (  
    unsigned long number  
) {  
    unsigned long tally = 0;  
    for (  
        int shift=0; // init  
        shift<8*sizeof(unsigned long); // ← test  
        shift++ // update  
    ) {  
        // body  
        tally += 0b1 & number>>shift;  
    }  
    return tally;  
}
```

```c
unsigned long count_bits_while (  
    unsigned long number  
) {  
    unsigned long tally = 0;  
    int shift=0; // init  
    while (  
        shift<8*sizeof(unsigned long) // ← test  
    ) {  
        // body  
        tally += 0b1 & number>>shift;  
        shift++; // update  
    }  
    return tally;  
}
```
Compiling while loops to do-while loops

```c
unsigned long count_bits_while (unsigned long number) {
    unsigned long tally = 0;
    int shift=0; // init
    while (
        shift<8*sizeof(unsigned long) // ← test
    ) {
        // body
        tally += 0b1 & number>>shift;
        shift++; // update
    }
    return tally;
}
```

```c
unsigned long count_bits_do_while (unsigned long number) {
    unsigned long tally = 0;
    int shift=0; // init
    do {
        // body
        tally += 0b1 & number>>shift;
        shift++; // update
    } while (shift<8*sizeof(unsigned long←)); // test
    return tally;
}
```

If initial iteration is guaranteed to run, then do one fewer test.
Compiling do-while loops to goto statements

Loops get compiled into goto statements which are readily translated to assembly.
Compiling goto statements to assembly conditional jump instructions

```c
unsigned long count_bits_goto (
    unsigned long number
) {
    unsigned long tally = 0;
    int shift=0; // init
    LOOP:
    // body
    tally += 0b1 & number>>shift;
    shift++; // update
    if (shift<8*sizeof(unsigned long)) {
        goto LOOP;
    }
    return tally;
}
```

All C loop statements so far translate to assembly at right.

```
count_bits_for:
  xorl %ecx, %ecx # int shift=0; // init
  xorl %eax, %eax # unsigned long tally = 0;
  .LOOP:
  movq %rdi, %rdx # number
  shrq %cl, %rdx # number>>shift
  incl %ecx # shift++; // update
  andl $1, %edx. # 0b1 & number>>shift
  addq %rdx, %rax # tally += 0b1 & number>>shift;
  cmpl $64, %ecx # shift<8*sizeof(unsigned long)
  jne .LOOP # goto LOOP;
  ret # return tally;
```
Table of contents

Announcements

Modifying control flow via conditional branch statements
  Jump instructions
  Conditional branch statements

Loop statements
  Compiling for loops to while loops
  Compiling while loops to do-while loops
  Compiling do-while loops to goto statements
  Compiling goto statements to assembly conditional jump instructions

Switch statements