Assembly: Loops, switches, functions.

Yipeng Huang

Rutgers University

March 11, 2021

Table of contents

Looking ahead

Class plan

1. Provide mid-semester course feedback at:
   `http://sirs.ctaar.rutgers.edu/blue`
2. Today, Thursday, 3/11: Assembly loops and switch statements.
3. Reading assignment for next two weeks: CS:APP Chapter 3.
4. Programming Assignment 3 on bits, bytes, integers, floats out. Due Monday March 22.

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

Jump instructions

Jumping

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

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)

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.

```
absdiff_if_else:
absdiff_goto:
    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

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.

```
1  unsigned long count_bits_for (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    for (
6      int shift=0; // init
7      shift<8*sizeof(unsigned long); // ↩
           test
8      shift++ // update
9    ) {
10     // body
11     tally += 0b1 & number>>shift;
12   }
13   return tally;
14 }
```

```
1  unsigned long count_bits_while (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    int shift=0; // init
6    while (
7      shift<8*sizeof(unsigned long) // ↩
           test
8    ) {
9      // body
10     tally += 0b1 & number>>shift;
11     shift++; // update
12   }
13   return tally;
14 }
```

Compiling while loops to do-while loops

```
1  unsigned long count_bits_while (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    int shift=0; // init
6    while (
7      shift<8*sizeof(unsigned long) // ↩
           test
8    ) {
9      // body
10     tally += 0b1 & number>>shift;
11     shift++; // update
12   }
13   return tally;
14 }
```

```
1  unsigned long count_bits_do_while (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    int shift=0; // init
6    do {
7      // body
8      tally += 0b1 & number>>shift;
9      shift++; // update
10   } while (shift<8*sizeof(unsigned long↩
         )); // test
11   return tally;
12 }
```

If initial iteration is guaranteed to run, then do one fewer test.

Compiling do-while loops to goto statements

```
1  unsigned long count_bits_do_while (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    int shift=0; // init
6    do {
7      // body
8      tally += 0b1 & number>>shift;
9      shift++; // update
10   } while (shift<8*sizeof(unsigned long↩
         )); // test
11   return tally;
12 }
```

```
1  unsigned long count_bits_goto (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    int shift=0; // init
6  LOOP:
7    // body
8    tally += 0b1 & number>>shift;
9    shift++; // update
10   if (shift<8*sizeof(unsigned long)) { ↩
         // test
11     goto LOOP;
12   }
13   return tally;
14 }
```

Loops get compiled into goto statements which are readily translated to assembly.

Compiling goto statements to assembly conditional jump instructions

```
1  unsigned long count_bits_goto (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    int shift=0; // init
6  LOOP:
7    // body
8    tally += 0b1 & number>>shift;
9    shift++; // update
10   if (shift<8*sizeof(unsigned long)) { ↩
         // test
11     goto LOOP;
12   }
13   return tally;
14 }
```

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

```
count_bits_for:
count_bits_while:
count_bits_do_while:
count_bits_goto:
  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>>shi
  cmpl $64, %ecx  # shift<8*sizeof(unsigned lo
  jne .LOOP       # goto LOOP;
  ret             # return tally;
```

Table of contents