CS354: Machine Organization and Programming

Lecture 10
Friday the September 26th 2015

Section 2
Instructor: Leo Arulraj

Class Announcements

1. Sample Ques and Midterm location posted in Exams link from Course website.

Oct 6th Tues 5:30 PM to 7:00 PM at Van Vleck Room B130(Section 2)

2. Anyone looking for a partner for P1 and beyond please come leave your name, email with me after class. I have a couple of students who are also looking for partners.

Lecture Overview

- Logical and shift instructions
- Condition codes
- Set instructions
- Jump instructions
- Conditional move instructions
- How to write in x86 assembly: do while loops, while loops, for loops, switch statements

Logical and Shift Instructions

`not`	D	D gets ~D (complement)
`and`	S, D	D gets D & S (bitwise logical AND)
`or`	S, D	D gets D | S (bitwise logical OR)
`xor`	S, D	D gets D ^ S (bitwise logical XOR)
`sal` `shl`	k, D	D gets D logically left shifted by k bits
`sar`	k, D	D gets D arithmetically right shifted by k bits
`shr`	k, D	D gets D logically right shifted by k bits

Examples

Assume x at %ebp+8, y at %ebp+12, z at %ebp+16

1 movl 12(%ebp), %eax	y
2 xorl 8(%ebp), %eax	t1 = x ^ y
3 sarl $3, %eax	t2 = t1 >> 3
4 notl %eax	t3 = ~t2
5 subl 16(%ebp), %eax	t4 = t3-z

Condition Codes

a register known as EFLAGS on x86

CF: **carry flag**. Set if the most recent operation caused a carry out of the msb. Overflow for unsigned addition.

ZF: **zero flag**. Set if the most recent operation generated a result of the value 0.

SF: **sign flag**. Set if the most recent operation generated a result that is negative.

OF: **overflow flag**. Set if the most recent operation caused 2's complement overflow.

Instructions related to EFLAGS

`sete` `setz`	`D`	set D to 0x01 if ZF is set, 0x00 if not set (place zero extended ZF into D)
`sets`	`D`	set D to 0x01 if SF is set, 0x00 if not set (place zero extended SF into D)
		. . . many more `set` instructions . . .

`cmpb`
`cmpw`
`cmpl`
	`S2, S1`	do S1 - S2 to set EFLAGS

`testb`
`testw`
`testl`
	`S2, S1`	do S1 & S2 to set EFLAGS

Instruction		Synonym	Effect	Set condition
`sete`	*D*	`setz`	*D* ← `ZF`	Equal / zero
`setne`	*D*	`setnz`	*D* ← `~ZF`	Not equal / not zero
`sets`	*D*		*D* ← `SF`	Negative
`setns`	*D*		*D* ← `~SF`	Nonnegative
`setg`	*D*	`setnle`	*D* ← `~(SF ^ OF) & ~ZF`	Greater (signed >)
`setge`	*D*	`setnl`	*D* ← `~(SF ^ OF)`	Greater or equal (signed >=)
`setl`	*D*	`setnge`	*D* ← `SF ^ OF`	Less (signed <)
`setle`	*D*	`setng`	*D* ← `(SF ^ OF) | ZF`	Less or equal (signed <=)
`seta`	*D*	`setnbe`	*D* ← `~CF & ~ZF`	Above (unsigned >)
`setae`	*D*	`setnb`	*D* ← `~CF`	Above or equal (unsigned >=)
`setb`	*D*	`setnae`	*D* ← `CF`	Below (unsigned <)
`setbe`	*D*	`setna`	*D* ← `CF | ZF`	Below or equal (unsigned <=)

setl and flags for 2's complement (Refer 3.6.2 in CSAPP Textbook)

1. When no overflow occurs: OF is 0
   a < b if a-b <0 indicated by SF = 1
   a >=b if a-b >=0 indicated by SF =0

2. When overflow occurs: OF is 1
   a <b if a-b >0 (positive overflow) [SF = 0]
   a >b if a-b <0 (negative overflow) [SF = 1]
   (no overflow when a is equal to b)

3. So, to test for a < b, we use SF ^ OF

4. Other signed comparison tests are based on other combinations of SF ^ OF and ZF

Control Instructions

`jmp`	`label`	goto `label`; %eip gets `label`
`jmp`	`*D`	indirect jump; goto address given by `D`
`je` `jz`	`label`	goto `label` if ZF flag is set; jump taken when previous result was 0
`jne` `jnz`	`label`	goto `label` if ZF flag is *not* set; jump taken when previous result was *not* 0
`js`	`label`	goto `label` if SF flag is set; jump taken when previous result was negative
`jns`	`label`	goto `label` if SF flag is *not* set; jump taken when previous result was *not* negative

More Control Instructions

`jg` `jnle`	`label`	goto `label` if EFLAGS set such that previous result was greater than 0
`jge` `jnl`	`label`	goto `label` if EFLAGS set such that previous result was greater than or equal to 0
`jl` `jnge`	`label`	goto `label` if EFLAGS set such that previous result was less than 0
`jle` `jng`	`label`	goto `label` if EFLAGS set such that previous result was less than or equal to 0

Jump: Relative vs Absolute

(Relevant for Linking which we will cover in later lecture)

- Assembly Jump statements use labels but assembler and later linker translate these labels to actual instruction addresses.
- **PC Relative:** difference between address of target instruction and address right after the jump instruction. (offsets use 1, 2 or 4 bytes)
- **Absolute:** use 4 bytes to directly specify target instruction
- Advantages of PC Relative:
  1. Instruction can be **compactly** encoded
  2. Object code can be **shifted to different positions in memory without alteration**

“if” and “if else” Stmts in Assembly

Overview of “if” and “if else” statement:

if(condition){
  statements;
}
	if(condition){
	  statements1;
	}else{
	  statements2;
	}

General Approach:

1. Use compare instructions to set the condition codes
2. Then use the jump instructions to execute the right set of instructions

IF STATEMENT EXAMPLE

```
if (y == x) {
    x++;
}
```

Assumptions:

- `x` and `y` are both integers
- `x` is already in `%ecx`
- `y` is already in `%edx`

```
    cmpl   %ecx, %edx
    jne    skip_incr    ZF set if they were equal
    incl   %ecx                 x++
skip_incr:
```

"if else" example

```
if(x<y){
    return y-x;
}else{
    return x-y;
}
```

x at %ebp+*8, y at* %ebp+*12*

```
1 movl 8(%ebp), %edx        Get x
2 movl 12(%ebp), %eax       Get y
3 cmpl %eax, %edx           Compare x:y
4 jge .L2                   if >= go to L2
5 subl %edx, %eax           result = y-x
6 jmp .L3                   Goto done
7 .L2:
8 subl %eax, %edx           result = x-y
9 movl %edx, %eax           %eax  = result
10 .L3: done: Begin completion code
```