Lecture 14: October 22, 2013

Midterms:

- Average 75
- Median 76
- Standard Deviation 9
- Highest 93

Objective:

Lecture 14 repository: review f24.s to f41.s

Details:

F24.s

```
Movl b, %ecx            //moves global variable b into %ecx
Movl a, %eax
Cmpl %ecx, %eax         //compares %ecx with %eax
```

When a, b not prefixed by any character, they are global variables
Global variables are 4 bytes

Tip: gdb command to look up registers: info registers

Condition codes:

"Hidden" register set as a side effect of many arithmetic instructions
combines many flags

- ZF – zero flag
- OF – overflow flag
- CF – carry flag
- SF – sign flag
- PF – parody flag

Comparison instructions:

Cmpl a, b => "subl a, b" without changing b
Test a, b => "andl a, b" without changing b

Conditional jump instructions:

Different jump instructions are used for unsigned and signed variables

Cmpl %ecx, %eax

- Jge => jump if %eax - %ecx greater than or equal to zero, used for signed
- Jle => jump if %eax - %ecx less than or equal to zero, used for signed
- Ja => jump if %eax above %ecx, used for unsigned
- Jb => jump if %eax below %ecx, used for unsigned
- More at www.unixwiz.net/techtips/x86-jumps.html

F:
.LBBO-2: these are not part of instruction stream
.Ltmp0:

0010 > 0001 (always true)
1000 > 0111 (true if unsigned, false if signed)

```
         0 1 1 1
-        1 0 0 0
----------------
+        1 1 1 1
```

F25.s

```
Movl b, %eax
Cmpl x, %eax              //subtract b from x
Jne .LBBo_2               //if ZF = 1 => jump
```

Returns either a or b
If b == x return a; else return b;
Jne: jump is not equal, if ZF is 1

F26.s

If b!= x return a; else return b;

F27.s

Unsigned example

F28.s

```
 31       23       15       7         0
|________|________|__%ah___|___%al___|
```

0-15: %ax
0-31: %eax

sete: take the value of 0 flag, put into %al

0 flag is 1 if they are equal (subtraction result of 0), in this case a = 0

1 - > 00000001
0 - > 00000000

movzbl: to make sure upper bits of %eax are zeros, move zero byte to long, used for unsigned

movsbl: used for signed

F29.s

Same instructions as F28.s

a is char*
Return !a; // take all non-zeros to 0, and take all zeros to 1

NULL is 0 on x86 machines

F30.s

Return a + x;
Not okay: add a pointer to a pointer
Okay: add a pointer to an int

Char* a;
Int x;
Return &a[x];

F31.s

Return a + (x << 2);

Or char* a;
Int x;
Return &a[4*x];

Or Int* a;
Int x;
Return &a[x];

F32.s

Type of a is likely to be an unsigned char which is 1 byte
Because movzbl extend a byte quantity into a long quantity

<u>Sign Extension:</u>
Using -1 as an example:
1111 = (movzbl) => 00001111
1111 = (movsbl) => 11111111

Movzbl and movsbl can provide information on whether the variable is signed or unsigned. In this case, because movzbl is used in f32.s; therefore the variable in f32.s must be an unsigned char

F33.s

```
Movl a, %eax
Movzbl (%eax), %eax          // stores the pointee of %eax into %eax
Ret
```

a is likely to be a pointer to a char because of movzbl
return *a or a[0]

<u>General x86 assembly:</u>

Off (base, index, size)
Address is off + base + index * size

- Off defaults to 0
- Index defaults to 0
- Size can only be [1,2,4,8] and defaults to 1

Index and size are useful for arrays, especially for arrays for chars, shorts, ints as 64 bit quantities, off is useful for structs, combined useful for array of structs.

F34.s

```
movl x, %eax
movzbl (%ecx, %eax), %eax  // only 1 comma, then no size
ret

unsigned char* a;
int x;
return a[x];
```

F35.s

```
Movl (%ecx, %eax, 4), %eax

Int* a;                        // because the size is 4 bytes
Int x;
Return a[x];
```

F36.s

```
Cmpl 0, (%ecx, %eax, 4)

Return a[x] != 0;              // because setne %al loads %al with not equal
```

F37.s

```
Movl x, %eax,
Movzwl (%eax), %eax          // x is an unsigned short*
Ret                            //returns *x
```

F38.s

Sums together an array of ints

Movl instructions can't tell whether a quantity is signed or unsigned, have to check extension instructions.

```
Xorl %eax, %eax        // set %eax to 0
Movl x, %ecx           // moving 32 bits quantity x to %ecx
Testl %ecx, %ecx
Je .LBBo_3             // if %ecx is initially 0, return 0

Decl %ecx              // decrement by 1
```

- Quantity has 4 bytes because of addl

F39.s

Identity function:
Movl 4(%esp), %eax // move 4 bytes from the stack pointer

F40.s

Add the two parameters

F41.s

Same assembly code as F40.s
Add 2 parameters, but has 6 more unused parameters

Further notes:

- If there is a jump backwards, then there is a loop
- %esp points to the top of the stack, where stores the return address (4 bytes)
- Following return address, there stores parameter values, with padding of 4 bytes
- Parameters laid out in stack as if in a struct
- Extra parameter values that are not used will not affect assembly code
- Compilers modify to base, so even when input codes are using array dereferencing, compiler may choose using either pointer arithmetic or array dereferencing
- When compiling while loops into assembly codes, the conditions move to the bottom to avoid extra jump statements