CS 354 - Machine Organization & Programming
Thursday, October 31, 2019

Midterm Exam (~18%): Thursday, November 7th, 7:15 - 9:15 pm

- **Lec 1 (2:30 pm):** room 3650 of Humanities
- **Lec 2 (4:00 pm):** room B10 of Ingraham Hall
- see “Midterm Exam 2” on course site Assignments for topics

Homework hw4 (1.5%): DUE TODAY at 10 pm on Thursday, October 31st

Project p4A (~2%): DUE at 10 pm on Tuesday, November 5th

Homework hw5 (1.5%): DUE at 10 pm Wednesday, November 6th

Project p4b (~4%): Assigned Tomorrow, DUE at 10 pm on Wednesday, November 13th

Last Time

Impact of Stride
Memory Mountain
C, Assembly, & Machine Code
Low-level View of Data
Registers
Instructions - MOV, PUSH, POP

Today

Instructions - MOV, PUSH, POP (from last time)
Operand Specifiers
Operands Practice
Operand/Instruction Caveats
Instruction - LEAL
Instructions - Arithmetic and Shift

Next Time

Finish Operands and Instructions
Re-read: B&O 3.5, 3.6

Operand Specifiers

What? Operand specifiers are

- S
- D

Why?

How?

1.) specifies an operand value that's

specifier	**operand value**
$*Imm*	*Imm*

2.) specifies an operand value that's

specifier	**operand value**
$\%E_a$	R[$\%E_a$]

3.) specifies an operand value that's

specifier	**operand value**	**effective address**	**addressing mode name**
Imm	M[EffAddr]	*Imm*	
($\%E_a$)	M[EffAddr]	R[$\%E_a$]	
Imm($\%E_b$)	M[EffAddr]	*Imm*+R[$\%E_b$]	
($\%E_b$,$\%E_i$)	M[EffAddr]	R[$\%E_b$]+R[$\%E_i$]	
Imm($\%E_b$,$\%E_i$)	M[EffAddr]	*Imm*+R[$\%E_b$]+R[$\%E_i$]	
Imm($\%E_b$,$\%E_i$,s)	M[EffAddr]	*Imm*+R[$\%E_b$]+R[$\%E_i$]*s	
($\%E_b$,$\%E_i$,s)	M[EffAddr]	R[$\%E_b$]+R[$\%E_i$]*s	
Imm(,$\%E_i$,s)	M[EffAddr]	*Imm*+R[$\%E_i$]*s	
(,$\%E_i$,s)	M[EffAddr]	R[$\%E_i$]*s	

Operands Practice

Given:

Memory Address	Value	Register	Value
0x100	0x	%eax	0x
0x104	0x	%ecx	0x
0x108	0x	%edx	0x
0x10C	0x		
0x110	0x		

→ What is the value being accessed? Also identify the type of operand, and for memory types name the addressing mode and determine the effective address.

Operand	Value	Type:Mode	Effective Address

1. `(%eax)`
2. `0xF8(,%ecx,8)`
3. `%edx`
4. `$0x108`
5. `-4(%eax)`
6. `4(%eax,%edx,2)`
7. `(%eax,%edx,2)`
8. `0x108`
9. `259(%ecx,%edx)`

Operand/Instruction Caveats

Missing Combination?

→ Identify each source and destination operand type combinations.

1. `movl $0xABCD,%ecx`
2. `movb $11,(%ebp)`
3. `movb %ah,%dl`
4. `movl %eax,-12(%esp)`
5. `movb (%ebx,%ecx,2),%al`

→ What combination is missing?

Instruction Oops!

→ What is wrong with each instruction below?

1. `movl %bl,(%ebp)`
2. `movl %ebx,$0xA1FF`
3. `movw %dx,%eax`
4. `movb $0x11,(%ax)`
5. `movw (%eax),(%ebx,%esi)`
6. `movb %sh, %bl`

Instruction - LEAL

Load Effective Address

```
leal S,D     D <-- &S
```

LEAL vs. MOV

```
struct Point {
   int x;
   int y;
} points[3];
```

```
int y = points[i].y;        mov 4(%ebx,%ecx,8),%eax
```

points[1].y;

```
int *py = &points[i].y;     leal 4(%ebx,%ecx,8),%eax
```

LEAL Simple Math

```
leal  -3(%ebx), %eax        subl $3, %ebx
                            movl %ebx, %eax
```

→ Suppose register %eax holds x and %ecx holds y.
What value in terms of x and y is stored in %ebx for each instruction below?

1. `leal (%eax,%ecx,8),%ebx`
2. `leal 12(%eax,%eax,4),%ebx`
3. `leal 11(%ecx),%ebx`
4. `leal 9(%eax,%ecx,4),%ebx`

Instructions - Arithmetic and Shift

Unary Operations

```
INC D       D <-- D + 1
DEC D       D <-- D - 1
NEG D       D <-- -D
NOT D       D <-- ~D
```

Binary Operations

```
ADD S,D     D <-- D + S
SUB S,D     D <-- D - S
IMUL S,D    D <-- D * S
XOR S,D     D <-- D ^ S
OR S,D      D <-- D | S
AND S,D     D <-- D & S
```

Bitwise exclusive or
" " or
— and

Given:

0x100	0xFF	%eax	0x100
0x104	0xAB	%ecx	0x1
0x108	0x10	%edx	0x2

→ What is the destination and result for each? (do each independently)

1. `incl 4(%eax)` 0xAC
2. `addl %ecx,(%eax)` 0xFF → 0x100
3. `addl $32,(%eax,%edx,4)` 0x10 → 0x20
4. `subl %edx,0x104`

Shift Operations

- Move bit to left or right by K positions $0 <= k <= 31$.
  must be IMM Type or Reg %CL.
- Does Fast integer multiply / divide by power of 2.

logical shift

```
SHL k,D     D <-- D << K
SHR k,D     D <-- D >> K
```

Shift Left, zero fill on right.
right left.

arithmetic shift

```
SAL k,D     D <-- D << K
SAR k,D     D <-- D >> K
```

same as SHL
shift right, most significant bit still on left to maintain the size.