

CS 354 - Machine Organization & Programming

Tuesday, October 31, 2017

Project p3 (6%): DUE at 10 pm TOMORROW, Wednesday, November 1st

Project p4 (6%): DUE at 10 pm on Wednesday, November 15th

Homework hw5 (1.5%): Assigned tomorrow

Last Time

- Exam Mechanics
- Project p3 and gdb
- Assembly Language Intro

Today

- Low-level Data
- Registers
- Instructions - MOV, PUSH, POP
- Operand Specifiers
- Operands Practice
- Operand/Instruction Caveats

Next Time

- Midterm Exams Returned
- More IA-32 Instructions
- Read:** B&O 3.5, 3.6

Low-Level View of Data

Instruction Set Architecture ISA

Specifies format and behavior of machine-level programming.

	x86 (AKA IA-32)	x64 (AKA x86-64, NOT IA-64)
address/data	32 bit	64 bit
address space	$2^{32} = 4\text{GB}$	$2^{64} = 16\text{EB}$ max mem today: 2^{48}

C's View

vars with specific types that can be complex composites built from arrays and structs

Machine's View

Memory is a large byte array
no distinction in memory for data types, pointers, etc.
Asm instructions determine how big? How many bytes
of data? What type? How to interpret the bits?

Assembly Data Formats

C	IA-32	Assembly Suffix	Size in bytes	
char	byte	<i>b</i>	1	
short	word	<i>w</i>	2	<i>NOTE: word isn't 4 bytes as we assumed</i>
int	double word	<i>l</i>	4	
long int	double word	<i>l</i>	4	
char*	double word	<i>l</i>	4	
float	single precision	<i>s</i>	4	
double	double prec	<i>l</i>	8	<i>Quad word</i>
long double	extended prec	<i>t</i>	10, typically 12 bytes	

Registers

What? Registers are: Fastest memory, which stores address and data that can be directly accessed by the ALU

General Registers

Pre-named locations that can store upto 32 bit values

	bit 31	15	8	7	1
%eax	accumulator		%ax	%ah	%al
%ecx	count		%cx	%ch	%cl
%edx	data		%dx	%dh	%dl
%ebx	base		%bx	%bh	%bl
%esi	Source Index		%si		
%edi	Destination Index		%di		
%esp	* Stack Pointer		%sp		
%ebp	* Stack Base Pointer		%bp		

Program Counter %eip Extended Instruction Pointer

It stores the address of the next instruction to be executed

Condition Code Registers

Store the status of the last ALU instruction
Used for conditioned changes in the program flow

Instructions - MOV, PUSH, POP

What? Instructions to copy data from one location to another

Why? Enables info to be moved around in registers and memory.

How?

instruction class	operation	description
MOV S,D <i>movb, movw, movl</i>	$D \leftarrow S$	move s value to d destination s and d are the same size
MOVS S,D <i>movsbw, movsbl, movswl</i>	$D \leftarrow \text{sign extended } S$	moves smaller s to d and fills d by copying most significant bit of s
MOVZ S,D <i>movzbw, movzbl, movzwl</i>		Fills d with zeros
pushl S	$R[\%esp] \leftarrow R[\%esp] - 4$ $M[R[\%esp]] \leftarrow S$	push s onto stack
popl D	$D \leftarrow M[R[\%esp]]$ $R[\%esp] \leftarrow R[\%esp] + 4$	pop top of stack to D

Practice with Data Formats

→ What data format suffix should replace the _ given the registers used?

1. mov_l %eax, %esp
2. mov_w (%eax), %dx
3. mov_b \$0xFF, %bl
4. mov_b (%esp, %edx, 4), %dh
5. push_l \$0xFF
6. mov_w %dx, (%eax)
7. pop_l %edi

		Bytes	Bits
b	bytes	1	8
w	word	2	16
l	double word	4	32

Operand Specifiers

What? Operand specifiers are:

- ♦ S source specifies value to be used by an instruction
- ♦ D destination specifies the location where the result is to be stored

Why?

Enables instructions to access constants (s only), registers and memory locations.

How?

Immediate specifies an operand value that's a constant
operand value
\$Imm Imm imm in C's format for constants

Register specifies an operand value that's in a register
operand value
%E_a R[E_a]

Memory specifies an operand value that's
operand value addressing mode

Imm M[Imm] absolute

(E_a) M[R[E_a]] indirect

Imm(E_b) M[Imm+R[E_b]] base + offset

(E_b, E_i) M[R[E_b]+R[E_i]] indexed

Imm(E_b, E_i) M[Imm+R[E_b]+R[E_i]] index + offset

(, E_i, s) M[R[E_i]*s]

(E_b, E_i, s) M[R[E_b]+R[E_i]*s] scaled indexed

Imm(, E_i, s) M[Imm+R[E_i]*s]

Imm(E_b, E_i, s) M[Imm+R[E_b]+R[E_i]*s]

s scale factor which is 1, 2, 4, 8

Operands Practice

Given:

Address	Value	Register	Value
0x100	0x FF	%eax	0x 104
0x104	0x AA	%ecx	0x 1
0x108	0x 11	%edx	0x 4
0x10C	0x 22		
0x110	0x 33		

→ What is the value being accessed? (Complete the information below.)

Operand	Value	Type/Mode	Effective Address
1. (%eax)	0xAA	Mem/Indirect	0x104
2. %edx	0x4	Reg	
3. 0x108	11	Mem/Absolute	0x108
4. \$0x108	0x108	IMM	
5. -4(%eax)	0xFF	Mem/Base+Offset	0x100
6. (%eax,%edx,2)	0x22	Mem/Scaled Index	0x10C
7. 0xF8(,%ecx,8)	0x22	Mem/SI	0x10C
8. 259(%ecx,%edx)	0x11	Mem/Indexed+Offset	0x108
9. 4(%eax,%edx,2)		Mem/SI	

Note:

Operand/Instruction Caveats

Missing Combination?

→ Identify each source and destination operand combination.

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

→ What combination is missing?

No memory to memory
instead you must do {memory to register} and then {register to memory}

Instruction Oops!

→ What is wrong with each instruction below?

1. `movl %bl, (%ebp)` Source operand is incorrect size
2. `movl %ebx, $0xA1FF` cannot have an imm type for D
3. `movw %dx, %eax` dest operand is wrong size
4. `movb $0x11, (%ax)` Mem type must use 32 bit registers
5. `movb %si, -12(%esp)` Source operand is incorrect size
6. `movw (%eax), (%ebx, %esi)` No mem to mem combination
7. `movb %sh, %bl` no register named sh