ECE 220
Honors Lab Section

Lab3: x86 Assembly

x86 Registers

- Registers
  - General registers
    - _AX – accumulator register (arithmetic)
    - _BX – base register (pointer to data)
    - _CX – counter register (loops)
    - _DX – data register (arithmetic and I/O)
    - _SP – stack pointer register (points to top of stack)
    - _BP – stack base pointer register (points to base of the stack)
    - _SI – source index register (pointer to source in stream ops)
    - _DI – destination index register (pointer to destination in stream ops)
  - Segment registers
    - CS, DS, ES, FS, GS, SS
  - Not easily accessible registers
    - _IP – instruction pointer
    - EFLAGS – holds state of the processor (condition codes)

x86 Registers

REG	64 bits			
		32 bits		
			16 bits	
			8 bits	8bits
_AX	RAX	EAX	AH (AX)	AL (AX)
_BX	RBX	EBX	BH (BX)	BL (BX)
_CX	RCX	ECX	CH (CX)	CL (CX)
_DX	RDX	EDX	DH (DX)	DL (DX)
_SP	RSP	ESP	SP	
_BP	RBP	EBP	BP	
_SI	RSI	ESI	SI	
_DI	RDI	EDI	DI	

Endianness

- What is it and types?
  - Order in which bytes are stored in memory
  - Little endian – least significant in lowest memory address
  - Big endian – least significant in highest memory address
- xABCD starting at x0000

Memory	Little	Big	Middle	Middle
x0000	D	A	C	B
x0001	C	B	D	A
x0002	B	C	A	D
X0003	A	D	B	C

- Little endian systems?
  - LC3, x86
- Big endian systems?
  - Internet

x86 Notation

- Comments - #this is a comment!
- Immediate values - $0xX (hex), $0X (octal) $x (decimal)
- Register - %ebp
- Labels – same as LC3
- Syntax
  - Two syntaxes: AT&T (Unix) vs Intel (MS-DOS)

	AT&T	Intel
Parameter order	movl $5, %eax	mov eax, 5
Parameter size	addl $4, %eax	add esp, 4
Sigils	Look above	None
Memory	DISP(BASE, INDEX, SCALE)	[BASE+INDEX*SCALE+DISP]

  - Intel syntax is similar to LC3
- **WE WILL USING AT&T SYNAX!**

X86 Operations

- Format
  - <OP><SIZE> <SRC>, <DEST>
    - <SIZE>
      - b = byte (8 bits)
      - w = word (16 bits)
      - l = long (32 bits)
      - q = quad (64 bits)
      - t = ten (80 bits)
    - <SRC>, <DEST>
      - Label
      - Memory
      - Register
      - Immediate value
      - BUT can only have 2 reads or read/write with respect to memory
- Examples
    - addl $4, %EAX or addl $4, %eax → EAX = EAX + 4
    - movb %dl 1(%ebx, %ecx, 4) → M[EBX + ECX * 4 + 1] = DL

x86 Operations

- Quiz: Solve
  - EAX = EAX + 0x8
  - AX = BX
  - CH = CL & M[EBX+ ECX * 4]
  - EBX = EBX | M[ECX]
  - M[BX] = M[BX] – AL
  - M[0] = M[0] + EAX
- Solutions
  - EAX = EAX + 0x8 → addl $0x8, %eax
  - AX = BX → movw %bx, %ax
  - CH = CL & M[EBX+ ECX * 4] → andb (%ebx, %ecx, 4), %cl
  - EBX = EBX ^ M[ECX] → xorl (%ecx), %ebx
  - M[BX] = M[BX] – AL → subb %al, (%bx)
  - M[0] = M[0] + EAX → addl %eax, 0

x86 Operations

- Quiz 2: Valid or Not
  - movl %eax, %ebx
  - movl %eax, 5
  - movl %eax, $5
  - movl %eax, 5(%ebx)
  - movl 4, %ebx
  - movl $4, %ebx
  - movl (%eax), (%ebx)
  - movl (%eax, %ebx, %ecx), %edx
- Solutions
  - True
  - True
  - False
  - True
  - True
  - True
  - False
  - False

x86 Branching

- CMP_ and TEST_ instructions set condition codes (EFLAGS)
  - cmpl %eax, %ebx → EFLAGS = EBX – EAX
  - testl %eax, %ebx → EFLAGS = EBX & EAX
- Comparisons perform jump

Unsigned	jne	jb	jbe	je	jae	ja
symbol	≠	<	≤	=	≥	>
Signed	jne	jl	jle	je	jge	jg

  - Unsigned uses 'above/below', signed uses 'less/greater'
  - Can insert 'n' after 'j' so jnae = jb
  - Uses flags set to check if should branch:

```
cmpl %eax, %ebx
jg <label>
```

  - Jumps if EBX > EAX

x86/C Example Program

- Check out from subversion!

GDB

- Similar to lc3sim!
- Go to lecture/max; *make; ./bin/max (run commands separately)*
- *gdb ./bin/max*
  - Commands
    - List
    - Break
    - Info
    - Delete
    - Run
    - Continue
    - Next
    - Step
    - Print
    - Quit