COS318 Assembly Intro

Based on slides of 2010

Introduction

- 16-bit real mode
  - 1MB memory
  - Programs can access any memory address
- CPU Registers
  - General purpose: %AX (H/L), %BX, %CX, %DX
  - Segment: %CS, %DS, %SS, %ES
  - Pointer: %IP, %BP, %SP
  - Index: %SI, %DI
  - Flags: 9 bits used, ZF, CF, etc.
  - http://www.cpu-world.com/Arch/8086.html

Segmentation

- 1 Megabyte Memory
  - Valid address range 0x00000 to 0xFFFFF
  - Use two 16-bit values: segment and offset
    - Written as segment:offset
    - Address = 16*segment + offset
  - 0x047C:0x0048 = ?
    - Addresses are not unique
      - Address 0x04808 referenced by 0x047C:0x0048, 0x047D:0x0038, ...

Assembly I

- AT&T Syntax vs. Intel Syntax
  - AT&T Syntax
    - Register names are prefixed with %
      - i.e. %ax
    - Source on left, destination on right
      - i.e. movw %ax, %bx *(load bx with value in ax)*
    - Prefix all constants and immediate values with $
      - i.e. movw $0x000d, %bx *(load bx with 0x000d)*
    - Suffix assembly instructions with size
      - i.e. *b* for byte (8 bits), *w* for word (16 bits), *l* for long (32 bits)
      - No need if the size is clear: mov %ax, %bx; mov $0x02, %al, but suggested

Assembly II

- AT&T Syntax (continued)
  - Addressing Syntax
    - i.e. movw $0x074b, 0x0 *(defaults to segment %ds)*
    - i.e. movw $0x074b, %es:(0x0) *(override default segment)*
    - i.e. lodsw *(%ax ← Mem[%ds:%si], %si++)*
- Recognizing Types of Assembly
  - Intel
    - Lack of prefixes/suffixes
    - Destination on the left, source on the right
  - Register naming: "**e**ax" is 32-bit code, "**r**ax" is 64-bit code

Assembly III

- Stack
  - push x
    - %sp--
    - Mem[%ss:%sp] ← x
  - pop x
    - x ← Mem[%ss:%sp]
    - %sp++
- Jumps
  - ljmp <imm1>, <imm2>
    - %cs ← imm1
    - %ip ← imm2
  - jmp <imm>
    - %cs stays
    - %ip ← imm
- Call and Ret
  - call <label>
    - push %ip
    - jmp label
  - ret
    - pop %ip

Assembly IV

- Arithmetic
  - addw / subw x,y
    - y ← y +/- x
  - mulw r
    - %dx%ax ← %ax * r
  - divw r
    - %ax ← %dx%ax div r
    - %dx ← %dx%ax mod r
  - inc / dec r
    - r ← r +/- 1
  - addb / subb x,y
    - y ← y +/- x
  - mulb r
    - %%ax ← %al * r
  - divb r
    - %al ← %ax div r
    - %ah ← %ax mod r

Assembly V

- If-Else
  - if(x < 10) { *foo* } *else* { *bar* }
    - movw ($x), %ax
    - cmpw $0xa, %ax
    - jnc elseClause
  - thenClause:
    - ***foo***
    - jmp endIf
  - elseClause:
    - ***bar***
  - endIf:

From "PC Assembly Language" (pcasm.pdf)
JNC branches only if CF is unset

Assembly VI

- For Loop
  - for(x=0; x<10; x++) { *foo* }
    - movw $0, %cx # use reg %cx to hold x
  - continueLoop:
    - *foo*
    - incw %cx
    - cmpw $0xa, %cx
    - jc continueLoop
  - breakLoop:

From "PC Assembly Language" (pcasm.pdf)
JC branches only if CF is set

Assembly VII

- Interrupts
  - int <imm> : invoke a software interrupt
    - int 0x10 *(console output)*
    - int 0x13 *(disk I/O)*
    - int 0x16 *(keyboard input)*
  - Each interrupt offers several functions
    - Specific function chosen by %ah
    - int 0x21 cannot be used

Assembly VIII

- Assembler Directives
  - Begin with a period (.)
  - Are not instructions
    - .equ name, value
      - Works just like #define
    - .byte, .word, .asciz
      - Reserve some memory
  - Used to segment a .s file
    - .text begins text (code) segment
    - .data begins data segment
    - .globl defines a list of symbols as global
      - does not define symbol, only declares as global
  - http://web.mit.edu/gnu/doc/html/as_7.html