x86 Assembly Tutorial

COS 318: Fall 2020

Project 1 Schedule

x86 Assembly Tutorial — Mon, 9/7 Tue, 9/8

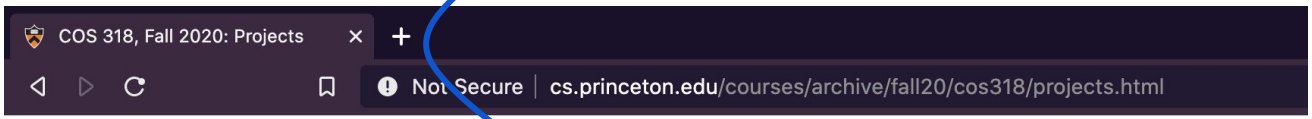
Deadline to register teams or sign-up for random team assignment — Fri, 9/11

Design Review Sessions (15 min per team) — 3-7pm ET Mon, 9/14 Tue, 9/15

Precept 1 (Design details and Q&A) — 7:30-8:20 pm ET Mon, 9/14 Tue, 9/15

Use Piazza and OHs.

Project 1 Due — **11:55 pm ET Sun, 9/20**



Project 1: Bootloader

Option 1: if picking own teammates	Team Registration Form
Option 2: if prefer random assignment	Random Team Assignment Request Form
Monday 9/7 & Tuesday 9/8, 7:30pm - 8:20pm	x86 Assembly Tutorial
Monday 9/14 & Tuesday 9/15, 3:00pm - 7:00pm	Design Review
Monday 9/14 & Tuesday 9/15, 7:30pm - 8:20pm	Precept 1
Sunday 9/20, 11:55pm	Project 1 due

The link will open on 9/12 or 9/13.

Overview

- x86 (*technically* IA-32) Assembly Overview
  - Registers, Flags, Memory Addressing, Instructions, Stack, Calling Conventions, Directives, Segments
- BIOS (Basic Input/Output System) +
  GDB (GNU Debugger)
- Design Review

Registers

General Purpose Registers: 8,16,32 bits

31	15	7	0	
	AH	AL	AX = AH | AL	EAX
	BH	BL	BX = BH | BL	EBX
	CH	CL	CX = CH | CL	ECX
	DH	DL	DX = DH | DL	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

Segment Registers: 16 bits
(hold 16-bit segment selectors
to identify memory segment)

CS	code segment
DS	data segment
SS	stack segment
ES	data segment
FS	data segment
GS	data segment

Instruction Pointer (EIP): 32 bits

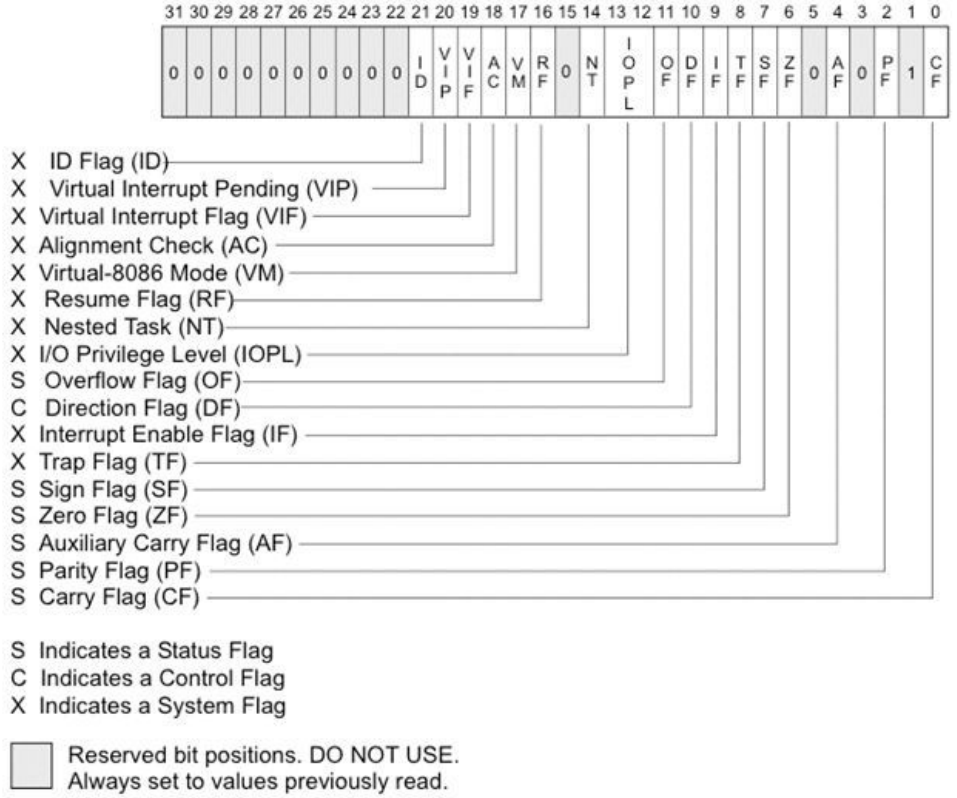
Flags (EFLAGS): 32 bits

Flags

The 32-bit EFLAGS register:

Flags types:

- Status
- Control
- System



- Important Flags
  - CF: Carry flag
  - ZF: Zero flag
  - SF: Sign flag
  - IF: Interrupt flag (sti, cli)
    [details: https://en.wikipedia.org/wiki/Interrupt_flag]
  - DF: Direction flag (std, cld)

Instruction Syntax Conventions

	Gnu Syntax (AT&T)	Intel
Immediate operands	Preceded by "$" e.g.:`push $4` `movl $0xd00a, %eax`	Undelimited e.g.: `push 4` `mov ebx, d00ah`
Register operands	Preceded by "%" e.g.: `%eax`	Undelimited e.g.: `eax`
Argument order (e.g. adds the address of C variable "foo" to register EAX)	source1, [source2,] dest e.g.: `addl $_foo, %eax`	dest, source1 [, source2] e.g.: `add eax, _foo`
Single-size operands	Explicit with operand sizes opcode{b,w,l} e.g.: `movb foo, %al`	Implicit with register name, **byte ptr**, **word ptr**, or **dword ptr** e.g.: `mov al, foo`
Address a C variable "foo"	`_foo`	`[_foo]`
Address memory pointed by a register (e.g. EAX)	`(%eax)`	`[eax]`
Address a variable offset by a value in the register	`_foo(%eax)`	`[eax + _foo]`
Address a value in an array "foo" of 32-bit integers	`_foo(,%eax,4)`	`[eax*4+foo]`
Equivalent to C code *(p+1)	`1(%eax)`	If EAX holds the value of p, then `[eax+1]`

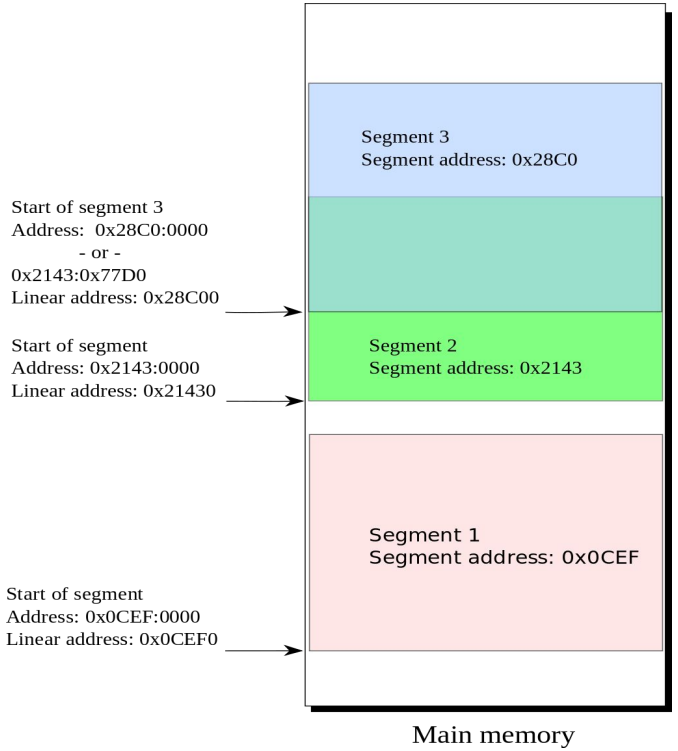
source: http://flint.cs.yale.edu/cs422/doc/pc-arch.html

Memory Addressing

- Memory addressing modes:
  1. Real address (unprotected) 2. Protected 3. System Management
- Real address mode:
  - 1MB of memory (20-bit addresses)
  - Valid address range: 0x00000 ~ 0xFFFFF
  - 16-bit segment add. [times 16, i.e., +4 bits] + 16-bit offset add.

Memory Addressing (Real Mode)

- Format (AT&T syntax):
  - **segment:displacement(base,index,scale)**
- Offset = Base + Index * Scale + Displacement
- Address = (Segment * 16) + Offset
- Displacement: Constant
- Base: %bx, %bp
- Index: %si, %di
- Segment: %cs, %ds, %ss, %es, %fs, %gs



Data Types

Name	Size (bits)
byte	8
word	16
double-word (long in gnu assembler)	32
quad-word	64

Instructions: Arithmetic & Logic

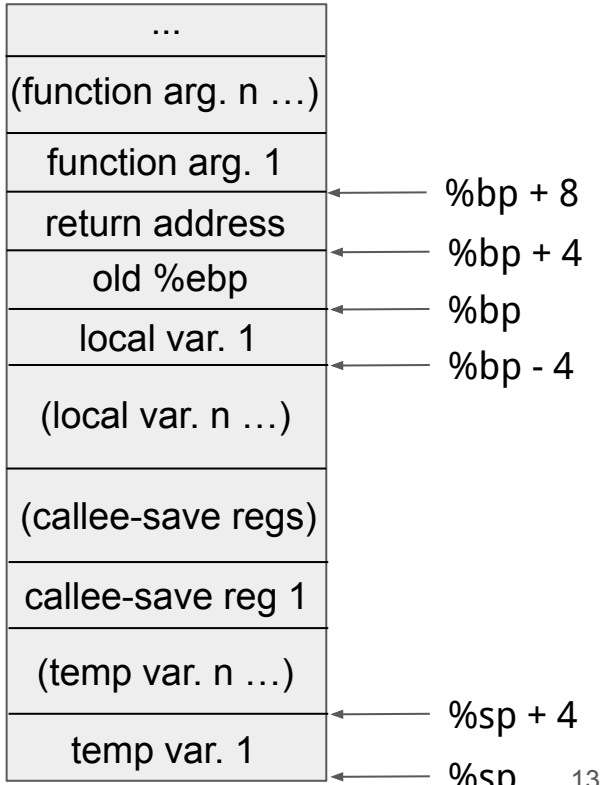
- Arithmetic, such as:
  - `add/sub{l,w,b} source,dest`
  - `inc/dec/neg{l,w,b} dest`
  - `cmp{l,w,b} source,dest`
- Logic, such as:
  - `and/or/xor{l,w,b} source,dest ...`
- Restrictions
  - No more than one memory operand

Instructions: Data Transfer

- `mov{l,w,b} source, dest`
- `xchg{l,w,b} source, dest` (exchange)
- `movsb/movsw` (move byte/word)
  - `%es:(%di) ← %ds:(%si)`
  - Often used with %cx to move a number of bytes
    - `movw $0x10,%cx`
    - `rep movsw` (repeat)

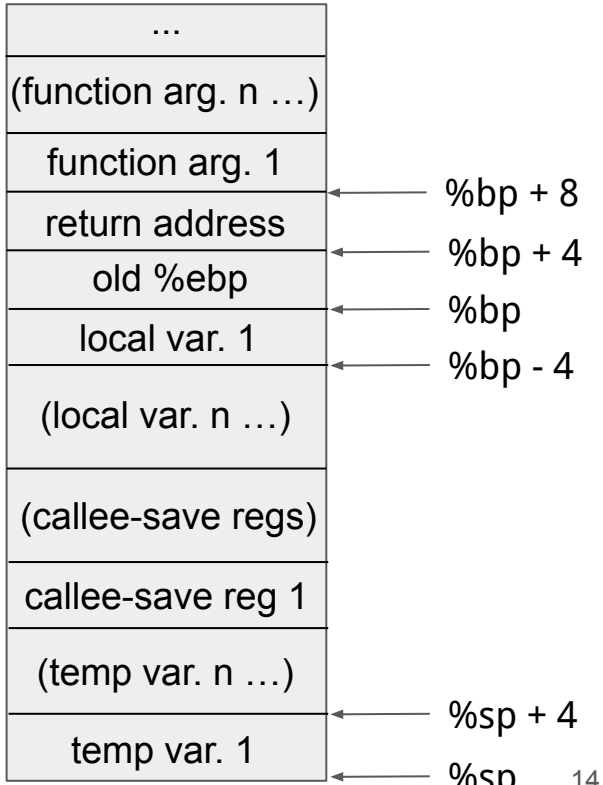
Stack Layout

- Grows from high to low
  - **Lowest address = “top” of stack**
- **%sp points to top** of the stack
  - Used to reference temporary variables
- **%bp points to bottom** of stack frame
  - Used for local vars + function args.



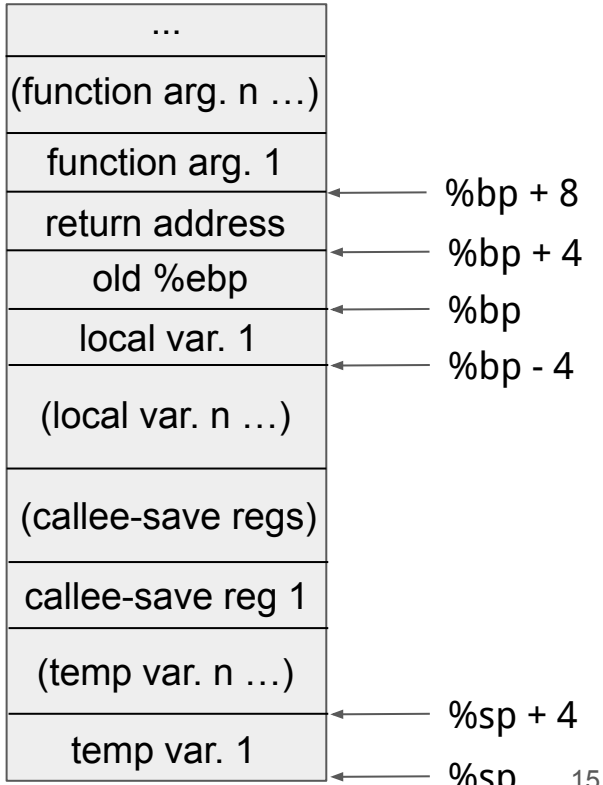
Calling Convention

- When calling a function:
  - 1. Push caller-save regs onto stack
  - 2. Push function args on to stack
  - 3. Push return address + branch
- In subroutine:
  - 1. Push old %bp + set %bp = %sp
  - 2. Allocate space for local variables
  - 3. Push callee-save regs if necessary



Instructions: Stack Access

- `pushl source`
  - %sp ← %sp - 4
  - %ss:(%sp) ← source
- `popl dest`
  - dest ← %ss:(%sp)
  - %sp ← %sp + 4



Instructions: Control Flow

- `jmp label`
  - %eip ← label
- `ljmp NEW_CS, offset`
  - %cs ← NEW_CS
  - %eip ← offset
- `call label`
  - push %eip
  - %eip ← label
- `ret`
  - pop %eip

Instructions: Conditional Jump

- Relies on %eflags bits
  - Most arithmetic operations change %eflags
- j{e,ne,l,le,g,ge}
  - Jump to label if {=,!=,<,<=,>,>=}

Assembler Directives

- Commands that speak directly to the assembler
  - Are not instructions
- Examples:
  - .globl - defines a list of symbols as global
  - .equ - defines a constant (like #define)
  - .bytes, .word, .asciz - reserve space in memory

https://docs.oracle.com/cd/E26502_01/html/E28388/eoiyg.html

Assembler Segments

- Organize memory by data properties
  - .text - holds executable instructions
  - .bss - holds zero-initialized data (e.g. static int i;)
  - .data - holds initialized data (e.g. char c = ‘a’;)
  - .rodata - holds read-only data
- Stack / Heap - Set up by linker / loader / programmer

Basic Input/Output System (BIOS) Services

- Use BIOS services through int instruction
  - Must store parameters in specified registers
  - Triggers a software interrupt
- `int INT_NUM`
  - int $0x10: Video services
  - int $0x13: Disk services

sending a character to the display at the current cursor position

`ah` = 0x0e, indicating this is function 0x0e
`al` = holding the character to write
`bh` = active page number (Use 0x00)
`bl` = foreground color (graphics mode only) (Use 0x02)

Useful GDB Commands

- r - show register values
- sreg - show segment registers
- s - step into instruction
- n - next instruction
- c - continue
- u <start> <stop> - disassembles C code into assembly
- b - set a breakpoint
- d <n> - delete a breakpoint
- bpd / bpe <n> - disable / enable a breakpoint
- x/Nx addr - display hex dump of N words, starting at addr
- x/Ni addr - display N instructions, starting at addr

Design Review

- Write print_char and print_string assembly functions
- Be ready to describe:
  - How to move the kernel from disk to memory
  - How to create disk image
  - (More specific guidelines are provided on the project page)