Intro

Study Assembly (Recap project 0, -m32)

Motivation

- Why study assembly
- high level, assembly & machine code diff

Goal of the lecture.

```
int accum = 0;
int sum(int x, int y)
{
  int t = x+y;
  accum += t;
  return t;
}
```

gcc -O1 -S code.c →

```
pushl   %ebp
movl    %esp, %ebp
movl    12(%ebp) %eax
addl    8(%ebp) %eax
addl    %eax, accum
popl    %ebp
ret
```

Machine languages

- IA32, x86-64

addressable range of IA32 = 4GB (byte addressable)

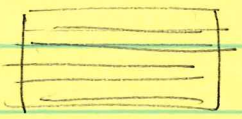
Reverse understanding

0010 0001 0010 0011

Add

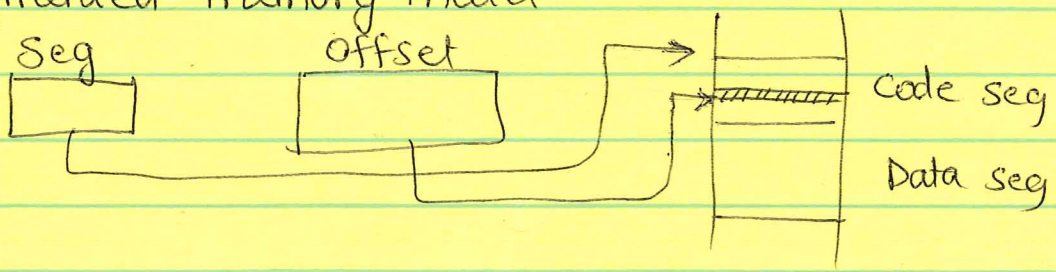
Memory models

Flat memory model



43.07, -89.40

Segmented memory model



Process Registers : Understand Processor state

→ Program Counter (PC)
Address of next instruct^n to be executed

→ Register files
"named" locations to store 32-bit values like pointers or int data
Program state / temp data

→ Condition code registers
Status of most recent arithmetic / logical instruction
e.g. overflow in addition

→ Floating point registers

Data formats
- More restrictive (less options) than C
Gerald's Sheet 2

Integer Registers
8 types of registers, sub-registers.

02/17/16

operand Forms.

| Type | Form | Operand Value | Name |
|---|---|---|---|
| Immediate | $Imm | Imm | Immediate |
| Register | $E_a$ | $R[E_a]$ | Register |
| Memory | Imm | $M[Imm]$ | Absolute |
| —11— | $(E_a)$ | $M[R[E_a]]$ | Indirect |
| | $Imm(E_b)$ | $M[Imm+R[E_b]]$ | Base + displacement |
| —11— | $(E_b, E_i)$ | $M[R[E_b]+R[E_i]]$ | Indexed |
| —11— | $Imm(E_b, E_i)$ | $M[Imm+R[E_b]+R[E_i]]$ | —11— |
| | $Imm(E_b, E_i, S)$ | $M[Imm+R[E_b]+R[E_i]\cdot S]$ | Scaled indexed |

Note: Scaled indexed format will always have "S" term.

→ xxx —

Lecture: Student questions

① Types of Registers. (PC → EIP) (Reg files → EAX, ---) (Condition Code Regs → EFLAGS)

② A → Accumulator, B → Base C → Counter ~~Index~~ D → Data.

③ How does it even work? C → Assembly w/ so few registers.

④ Why few registers (Power, area limitation)

Review

① Register AL, AH, AX, EAX.

② Data formats Byte(1) → b, word(2) → w, DW(4) → 4
char short int

③ Operand Forms.

Lecture

Practice problem 3.1

Practice Problem 3-1 (2)

| Address | Value |
|---|---|
| 0x100 | 0xFF |
| 0x104 | 0xAB |
| 0x108 | 0x13 |
| 0x10C | 0x11 |

| Register | Value |
|---|---|
| %eax | 0x100 |
| %ecx | 0x1 |
| %edx | 0x3 |

| Operand | Value |
|---|---|
| %eax | 0x100 |
| 0x104 | 0xAB |
| $0x108 | 0x108 |
| (%eax) | 0xFF |

Remember, memory is Byte addressable

Address 0x10C → Val 0x11

9(%eax, %edx) → address = 9 + 0x100 + 0x3 = 9 + 256 + 3 = 268

260(%ecx, %edx)

0xFC(, %ecx, 4)

(%eax, %edx, 4)

260(%ecx, %edx) → address = 260 + 0x1 + 0x3
= 260 + 1 + 3 = 264 = 0x108
Value = 0x13.

0xFC(, %ecx, 4) ⇒ address = 0xFC + 0x1 * 4 =
= 0xFC + 4 = 15x16 + 12 + 4
= 256. = 0x100
Value = 0xFF

(%eax, %edx, 4) Address = 0x100 + 0x3 x 4 = 0x
= 0x10C
Value = 0x11

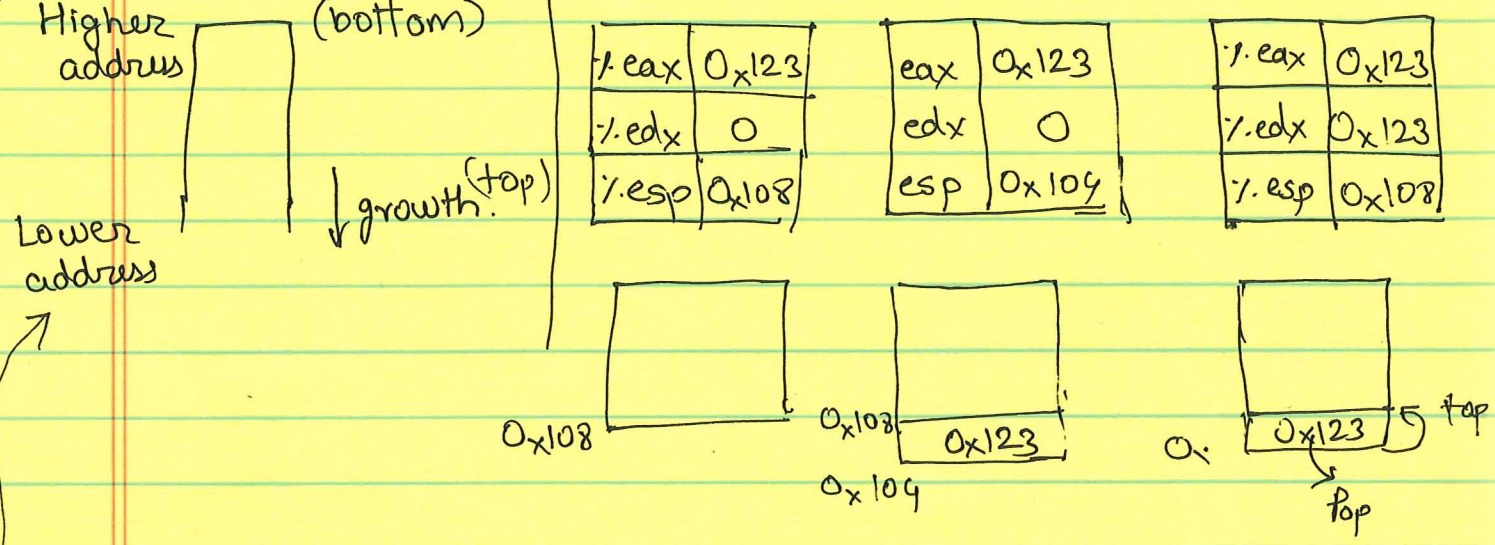
~~Stack~~ Data Movement

~~MOV~~ & Gerald Sheet 4

02/17/16

Move to San Diego story.

Stack operations.




Practice Problem 3.2 : Determine Suffix (b/w/l)

| | | |
|---|---|---|
| mov__ | %eax, %esp. | (l) |
| mov__ | (%eax), %dx | (w) |
| mov__ | $0xFF, %bl | (b) |
| push__ | $0xFF | (b) |

# Move $mem_1 \rightarrow memory_2$ not allowed in 1 instruct$^n$

↳ Move $mem_1$ reg, move reg, $mem_2$

e.g. Gerald pg. 5

---

Leal (Gerald pg. 6)

↳ has no variation in suffix. (why?)

02/19/16 Examples from last lecture

~~0xFC~~

(%eax, %edx, 4)  Address = $0_x100 + 0_x3 \times 4$ (No Imm)
$= 0_x10C$

Contents = $0_x11$

→ explain movb (%eax), %dh

mov_ (%eax) %dx

2 possibilities → move a byte (b) : bw
move 2 bytes (w) → w

mov mem → mem not allowed : split in 2.

| $0_x100$ | $0_x101$ | $0_x102$ | $0_x103$ |
|---|---|---|---|
| 00 | AA | BB | 22 |

after Stack

movb mem_address, %al
movw mem_address, %ax
movl ——11—— , %eax
} All are valid.

Stack Operations. : Pg 3 02/17/16.

leal (Gerald Pg 6) : Only take eg ①

Arithmetic operations : Gerald pg ⑦
addl %ecx, (%eax)
S D
D ← D+S

$0_x100$ [FF] $0_x104$ [AB]

value at S = $0_x1$  value at D = M[%eax] = $0_xFF$

$0_x100 \rightarrow 0_x$00FF
$0_x104 \rightarrow 0_x$00AB
new value at D = $0_x1 + 0_xFF = 0_x100$.

| $0_x100$ | $0_x101$ | $0_x102$ | $0_x103$ | $0_x104$ | $0_x105$ | $0_x106$ | $0_x107$ |
|---|---|---|---|---|---|---|---|
| 00 | 01 | 00 | 00 | AB | 00 | 00 | 00 |

e.g.

$0_x100$ $0_xFF$ Let's assume $0_x101 - 0_x103 : 0_x00$.

$0_x104$ $0_xAB$ %ecx $- 0_x1$

%eax $- 0_x100$.

addl %ecx, (%eax) (Carry)

Shift instruct$^n$ : $0 \leq k \leq 31$

left shift : add zeros. ri

3.4.4, 3.4.5 (Reading).

Control : Gerald Pg 8, 9, 10.

Worksheet.