



# UIC - CS 261 - Midterm Exam 1 Prep

## Lecture 11: Stack & Intro to Procedures

```
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq    %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
     movl    $5, -16(%rbp)
     movl    $-1, -12(%rbp)
    movsd    .LC0(%rip), %xmm0
    movsd    %xmm0, -8(%rbp)
    movl    $0, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
```

Trong 1 đoạn code có:

- `movl $5, -16(%rbp)`
- `movl $-1, -12(%rbp)`

Và:

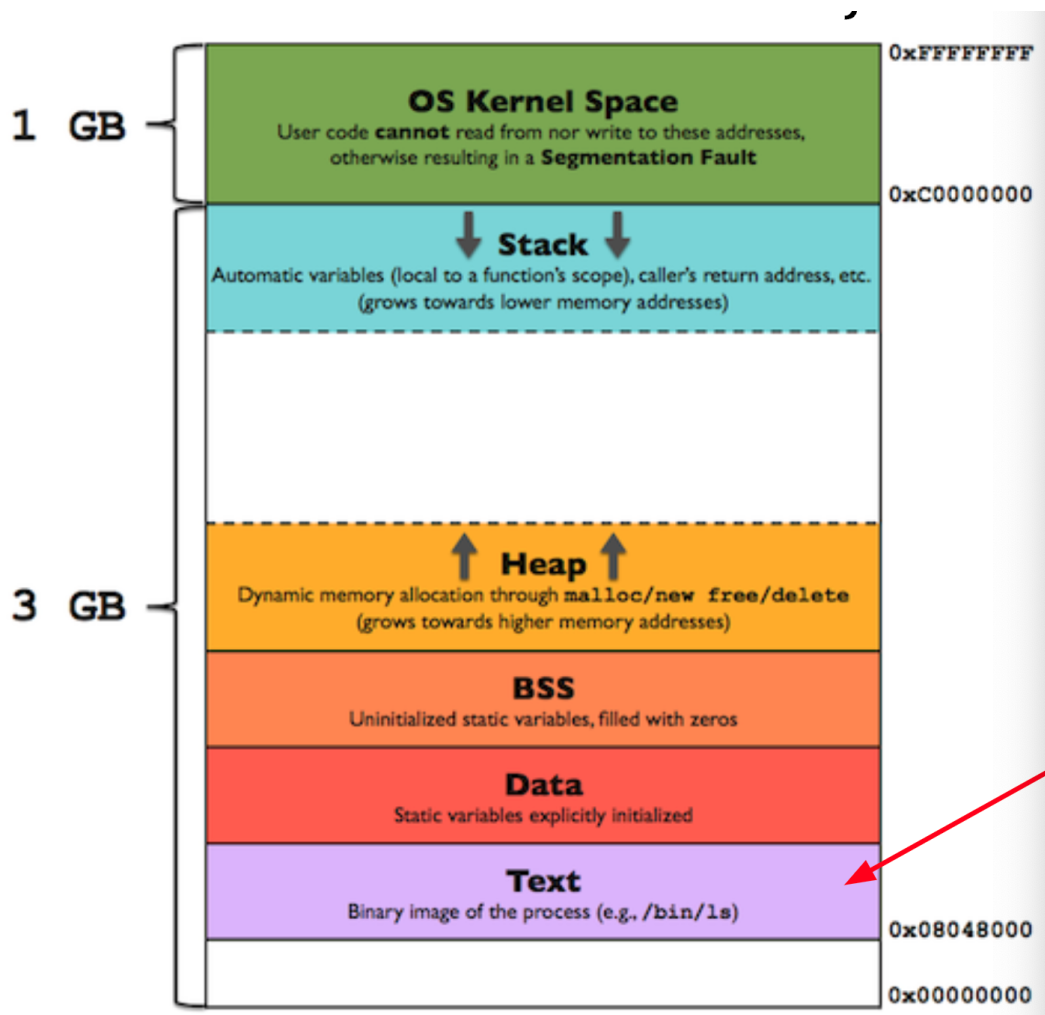
- `pushq`

- %rbp
- \$rip
- popq
- ret

→ Là cái gì zậy ?\_?

I. OK giả thuyết ở đây là Instruction Pointer:

1. Điều đầu tiên, phải biết cái instruction tiếp theo
  2. Sau đó là Memory hoạt động như nào...
- 2.1.



## 2.2. Nhìn vô hiểu gì ko??

Đầu tiên sẽ có OS Kernel với 1GB memory

NHƯNG, sau đó sẽ có 3GB memory chứa Stack, Heap, BSS, Data, và Text (code)

!!! Stack nó sẽ đi thứ tự từ trên xuống.

## 3. Phải hiểu instruction pointer \$rip (còn gọi là \$pc) <program counter> là gì??

3.1. Mình có info registers của cái \$rip này:

- x/1i \$rip
- x/1i \$pc

4. Ok, giờ nhớ lại trong C language:

**return / output**

Recall: Procedures

```
int addTimes3(int x, int y){  
    int w = y*3;  
    int z = x + w;  
    return z;  
}
```

**parameters / input**

### Giải thích:

Cái function calls sẽ dựa vào cái jump, xong rồi run instructions thích hợp rồi return kết quả sau cái chỗ mà call. Nhưng mà, nó khó để lưu trữ Program Counter hay mấy cái value từ một cái register nào đó

⇒ NOTE: cái này để hiểu function calls nó như thế nào để còn ứng dụng trong assembly.

## 5. Học về STACK nè:

Top of the stack có thể lấy elements nhưng Bottom lại không nha!

5.1. Stack dùng để chứa function's scope: **caller's return address** chẳng hạn

5.2. Mục đích của Stack để làm gì?

- Để chứa những data tạm thời như local variables của một vài functions

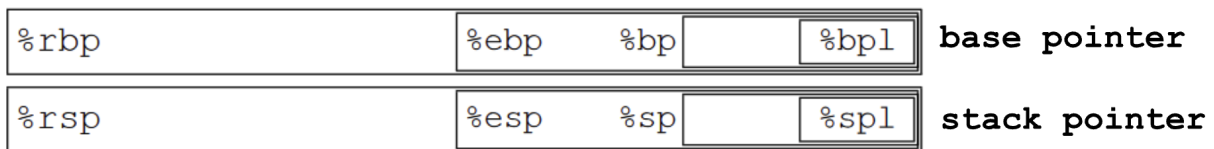
5.3. Mình dùng **PUSH** và **POP** instructions để dùng Stack

- Last In First Out

5.4. Registers để giữ Stack hoạt động:

Tại mình có:

- %rbp → %ebp → %bp → %bpl : gọi là Base Pointer
- %rsp → %esp → %sp → %spl: gọi là Stack Pointer



## 6. Dùng Stack nhưng trong Assembly Instructions thì sao?

- push register
- pop register
- call label
- ret
- leave

## 7. Giả định:

Khi mình có hàm main, với scope là func1.

func1 chứa func2

func2 chứa func3

### 7.1. Thứ tự call:

- Để chạy chương trình, mình call **main**
- Sau đó call func1. Stack lúc này có main → func1
- Call func2, Stack lúc này có main → func1 → func2
- Nhưng func3 lúc này có giá trị return ngay tại func2  
→ nên là Pop func2 ra khỏi Stack  
→ Stack lúc này có main → func1 → func3
- func3 có giá trị return để kết thúc chương trình  
→ Pop func3 ra  
→ Stack lúc này còn main → func1  
POP hết cho tới khi Stack is empty

## 8. Hiểu thêm về Push Instruction

### 8.1. Các bước hoạt động

Step 1: Update Stack Pointer

Step 2: Copy value to top of Stack

### 8.2. Ví dụ:

Ta có:

- %rbp là 0x123
- %rsp là 0x108

### 8.3. **pushq %rbp** tương ứng với gì?

NOTE: q là 8 bytes

→ Step 1: `subq $8, %rsp` (vì rsp đang là top của Stack)

→ Step 2: `movq %rbp, %rsp` (thế chỗ)

Lúc này ta có:

- %rbp là 0x123
- Nhưng, %rsp là 0x100 (vì rsp bị thế chỗ bởi address khác)

## 9. Hiểu thêm về Pop Instruction

### 9.1. Các bước hoạt động

Step 1: Bỏ cái element trong stack ra

Step 2: thêm 8 bytes vô cho cái chỗ vừa lấy

### 9.2. Ví dụ:

Ta có:

- %rbp là 0x000
- %rsp là 0x100

### 9.3. **popq %rbp** tương ứng với gì?

→ Step 1: `movq (%rsp), %rbp`

→ Step 2: `addq $8, %rsp`

Lúc này ta có:

- %rbp là 0x123
- %rsp là 0x108

( cả 2 đều thay địa chỉ)

## 10. Hiểu thêm về Call Label:

### 10.1. Các bước hoạt động:

Step 1: Push return address

Step 2: Jump to label

### 10.2. Tương ứng với gì?

`subq $8, %rsp`

`movq $retadd, %rsp`

`jump label`

Example: call func1

## 11. Hiểu thêm về **ret instruction**?

### 11.1. Các bước hoạt động:

Pop return address ra khỏi Stack

Xong rồi, next instruction sẽ là return address (Jump to that address  
aka update rip)

### 11.2. Tương ứng:

```
addq $8, %rsp
```

```
movq -8(%rsp), %rip
```

## 12. **Leave** để làm gì?

- Để restore %rsp và %rbp, rồi pop khỏi Stack
- Tương ứng:

```
mov %rbp, %rsp
```

```
pop %rbp
```

---

# Lecture 10 - Intro to Assembly

Trong lớp CS 261 này sẽ dùng Assembly x86-64 + UNIX operating system

## 1. Có bao nhiêu registers?

tổng cộng **16 cái**

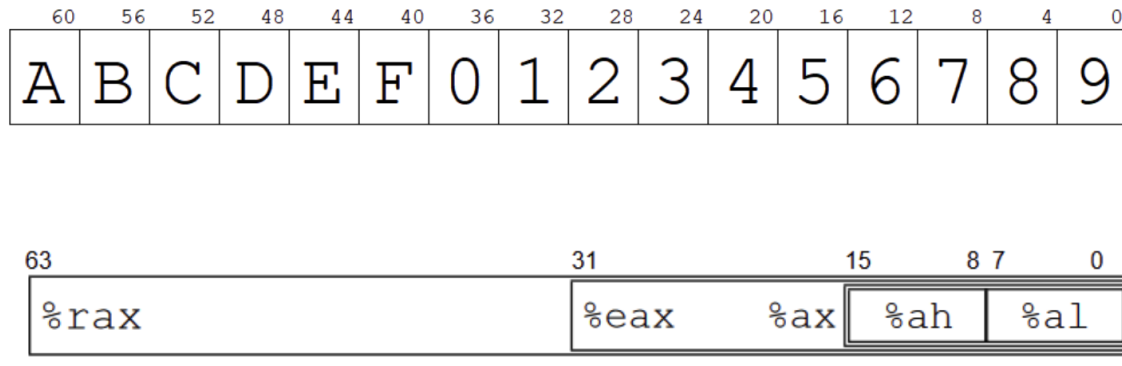
|      |       |       |     |       |   |                  |
|------|-------|-------|-----|-------|---|------------------|
| 63   | 31    | 15    | 8   | 7     | 0 |                  |
| %rax | %eax  | %ax   | %ah | %al   |   | Return value     |
| %rbx | %ebx  | %bx   | %bh | %bl   |   | Callee saved     |
| %rcx | %ecx  | %cx   | %ch | %cl   |   | 4th argument     |
| %rdx | %edx  | %dx   | %dh | %dl   |   | 3rd argument     |
| %rsi | %esi  | %si   |     | %sil  |   | 2nd argument     |
| %rdi | %edi  | %di   |     | %dil  |   | 1st argument     |
| %rbp | %ebp  | %bp   |     | %bpl  |   | Callee saved     |
| %rsp | %esp  | %sp   |     | %spl  |   | Stack pointer    |
| %r8  | %r8d  | %r8w  |     | %r8b  |   | 5th argument     |
| %r9  | %r9d  | %r9w  |     | %r9b  |   | 6th argument     |
| %r10 | %r10d | %r10w |     | %r10b |   | Callee saved     |
| %r11 | %r11d | %r11w |     | %r11b |   | Used for linking |
| %r12 | %r12d | %r12w |     | %r12b |   | Unused for C     |
| %r13 | %r13d | %r13w |     | %r13b |   | Callee saved     |
| %r14 | %r14d | %r14w |     | %r14b |   | Callee saved     |
| %r15 | %r15d | %r15w |     | %r15b |   | Callee saved     |

LV

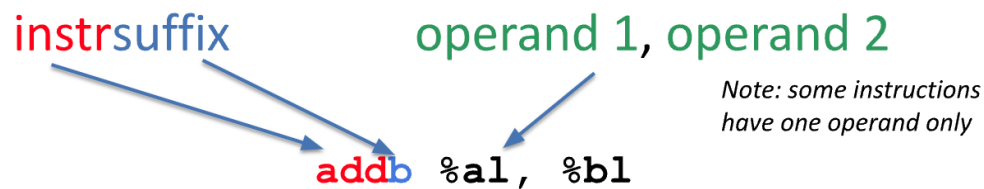
## 2. Trong 1 register có gì?

- %rax: 64 bits
- %eax: 32 bits
- %ax: 16 bits
- %ah: high 8 bits of %ax
- %al: low 8 bits of %ax





3. Trong **x86 Instructions** có gì?



3.1. instr gồm:

3.1.1: Data Movement:

- mov
  - movz: moves và biến remaining bytes thành zero hết
  - movzbq: moves 1 byte to a quad word xong biến remaining thành zero
  - movs: moves nhưng sign remaining bytes
  - movsbw: moves 1 byte to a word nhưng sign cái remaining bytes
  - movabs: move absolute
- push
- pop

3.1.2: Arithmetic:

- add, sub
- mul (unsigned), imul (signed)

- div (unsigned), idiv (signed)
- inc, dec
- neg

### 3.1.3: Shifting:

- and, or, xor, not
- sal: left shift
- shl: same as sal
- sar: arithmetic right shift (care sign bit)
- shr: logical right shift (not care sign bit)

### 3.1.4: Transfer"

- jmp: jump to address
- call: call the function (push return address and jump)
- ret: return the value (pop return address ra khỏi Stack only)

### 3.1.5: **leaq** ( Load Effective Address)

ví dụ: **leaq 6(%rax, %rcx, 4), %rdx**

### 3.2. suffix gồm:

- b: byte (8 bit)
- w: word (16 bit)
- l: long word (32 bit)
- q: quad word (64 bit)

### **ví dụ: iClicker**

a1 = 25 = 0001 1001

shlb \$4, %al: 1001 0000

sarb \$2, %al: 1110 0100

shrb \$1, %al: 0111 0010

