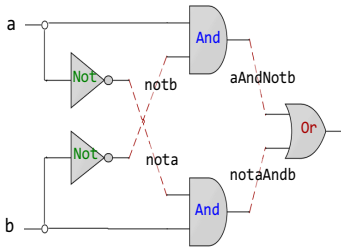
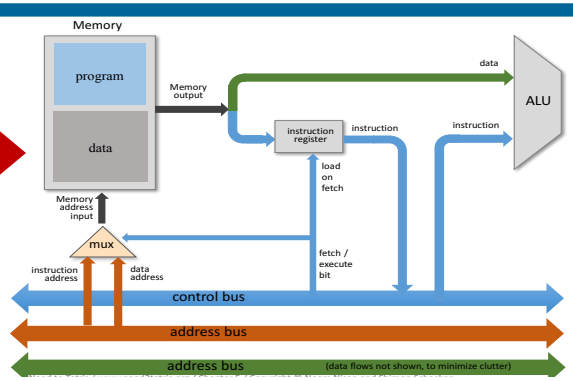
Computer Organization & Assembly Language Programming




```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```




```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 40

Control Instructions - II

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

For resources visit my personal website:
https://www.arifbutt.me
and course bitbucket repository:
https://bitbucket.org/arifpucit/coal-repo

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- **Recap:**
  - x86-64 Registers, Tool Chain & Instructions
  - x86 **`JMP`** and **`Jcc`** Instructions
- Repetition Structure using **`Jcc`** Instructions
  - Do…while Loop Demo (***`dowhile.nasm`***)
  - While/For Loop Demo (***`while.nasm`***)
- Repetition Structure using **`loop`** Instructions
  - Demo (***`loop.nasm`***)
  - Demo (***`final.nasm`***)

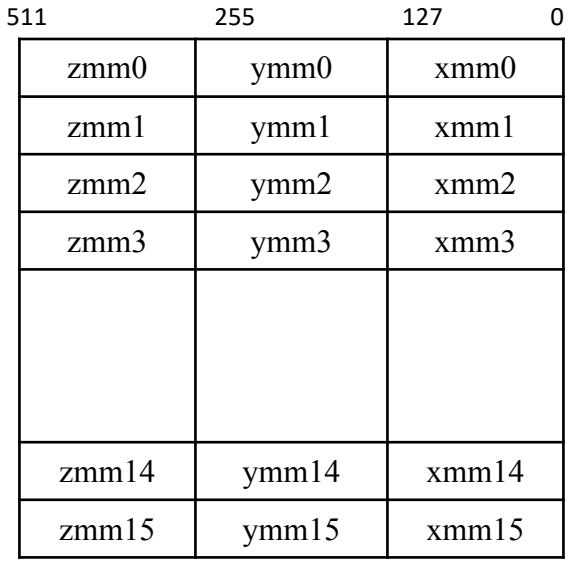
Recap

Review: x86-64 Register Set

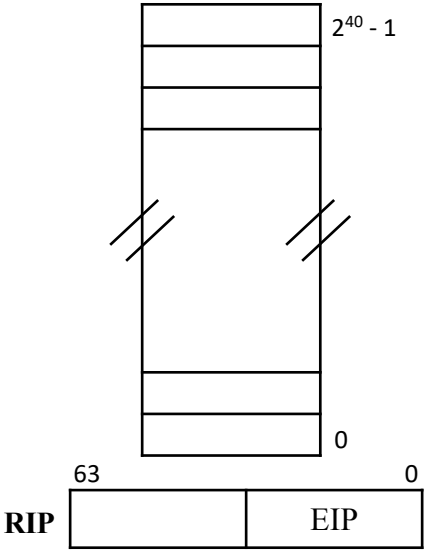
General Purpose Registers

64-bit register	Lowest 32-bits	Lowest 16-bits	Lowest 8-bits
r0/rax	eax	ax	al
r1/rbx	ebx	bx	bl
r2/rcx	ecx	cx	cl
r3/rdx	edx	dx	dl
r4/rsi	esi	si	sil
r5/rdi	edi	di	dil
r6/rbp	ebp	bp	bpl
r7/rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

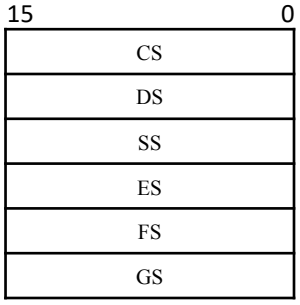
SSE Media Registers



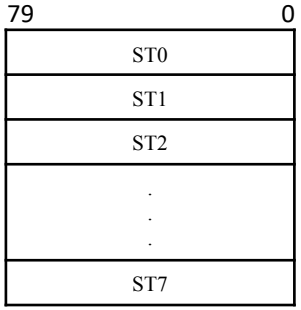
Memory

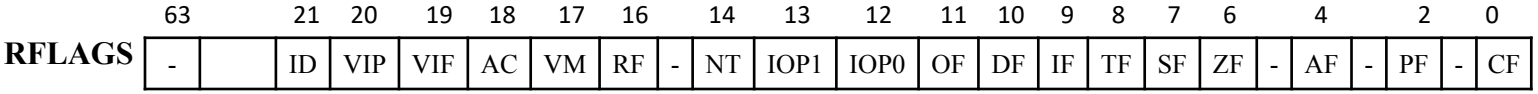


Segment Registers



FP Registers





Review: x86-64 Tool Chain

first.nasm

```
; COAL Video Lecture: 30
;  Programmer: Arif Butt
;  first.nasm
SECTION .data
   msg db "Learning…", 0xA
   EXIT_STATUS equ 54
SECTION .bss
;nothing here
SECTION .text
   global _start
   _start:
;display a message on screen
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,26
      syscall
;exit the program
      mov rax,60
      mov rdi, EXIT_STATUS
      syscall
```

Assemble →

first.o

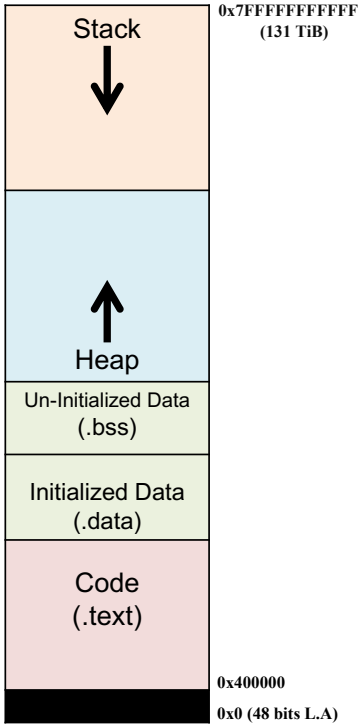
```
10001000
01000001
1000101001001001
0101011000011111
0001010011110000
10001000
01001101
10001000
01001001
1000101001001000
0101011000011000
0001010010010001
10001010
01001011
```

Link →

myexe

```
1000101001001001
0101011000011111
0001010011110000
10001000
01001101
10001000
10001000
01000001
0101011000011111
0001010011110000
10001000
1000101001001000
0001010010010001
10001010
01001011
0001010011110000
10001000
01001101
10001000
```

Load & Execute →



- **Processor:** Core 2duo/i3/i5/i7 (64 bit processor)
- **Operating System:** 64 bit Linux Distro (Ubuntu, Kali)
- **Editor:** gedit, vim, atom, sublime, Visual Studio, Eclipse, Xcode
- **Assembler:** NASM, YASM, GAS, MASM
- **Linker:** LD a GNU linker
- **Loader:** Default OS
- **Debugging/RE:** gdb, radare2, objdump and readelf

Review: Categories of x86-64 Instructions

Category	Description	Examples
Data Transfer	Move from source to destination	`mov, movzx, movsx, lea, lds, lss, xchg, push, pop, pusha, popa, pushf, popf`
Arithmetic	Arithmetic on integer	`add, addc, sub, subb, mul, imul, div, idiv, neg, inc, dec, cmp`
Bit Manipulation	Logical & bit shifting operations	`and, or, not, xor, test, shl/sal, shr, sar, ror, rol, rcr, rcl`

Control Transfer	Conditional and unconditional jumps, and procedure calls	`jmp`
		`jcc(jz,jnz,jg,jge,jl,jle,jc,jnc,...)`
		`call, ret`

String	Move, compare, input and output	`movsb, movsw, lodsb, lodsw, stosb, stosw, rep, repz, repe, repnz, repne`
Floating Point	Arithmetic	`fld, fst, fstp, fadd, fsub, fmul, fdiv`
Conversion	Data type conversions	`cbw, cwd, cdq, xlat`
Input Output	For input and output	`in, out`
Miscellaneous	Manipulate individual flags	`clc, stc, cld, std, sti`

Unconditional JUMP Instruction

- **Format:** `JMP destination`
- **Operation:** Transfers program control to a different location in the instruction stream (unconditionally). The destination operand specifies the address of the instruction being jumped to
- **Operand:** Destination operand is normally a label, i.e., a memory address pointing to some instruction. But can also be a register or immediate value
- **Types:**
  - **Short jump:** A jump where the jump range is limited to -128 to +127 from the current RIP value. (CS do not change)
  - **Near jump:** A jump within the current code segment. (CS do not change)
  - **Far jump:** A jump to an instruction located in a different segment than the current code segment
  - **Task switch:** A jump to an instruction located in different task

```
; COAL Video Lecture: 39
;  Programmer: Arif Butt
;  uncondjump1.nasm
SECTION .data
   msg1  db   "Study COAL", 0xA
   len_msg1  equ  $ - msg1
   msg2  db   "Play Cricket", 0xA
   len_msg2  equ  $ - msg2
SECTION .text
   global _start
_start:
   mov rax, 1
   mov rdi, 1
   mov rsi, msg1
   mov rdx, len_msg1
   syscall
   JMP _end
   mov rax, 1
   mov rdi, 1
   mov rsi, msg2
   mov rdx, len_msg2
   syscall
_end:
   mov rax, 60
   mov rdi, 0
   syscall
```

Conditional JUMP Instructions: `Jcc dest`

Conditional jump instructions are mostly used after a **`cmp op1, op2`** instr

Unsigned Jumps	Description	Condition for jump
JA	Jump if above than	If op1 > op2, ZF=0 and CF=0
JAE	Jump if above than or equal to	If op1 >= op2, CF=0
JB	Jump if below than	If op1 < op2, CF=1
JBE	Jump if below than or equal to	If op1 <= op2, ZF=1 and CF=1

Signed Jumps	Description	Condition for jump
JG	Jump if greater than	If op1 > op2, ZF=0 and SF=OF
JGE	Jump if greater than or equal to	If op1 >= op2, SF=OF
JL	Jump if less than	if op1< op2, SF<>OF
JLE	Jump if less than or equal to	If op1 <= op2, ZF=1 and SF<>OF

Single Flag Jumps	Description	Condition for jump
JE/JZ	Jump if equal	ZF=1, if op1==op2
JNE/JNZ	Jump if not equal	ZF=0, if op1 <> op2

https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf

Repetition Structure

Example: dowhile.nasm

High Level Code:

```
int i = 0;
do{
    //loop instructions
    i++;
}while (i < 5);
```

```
;  COAL Video Lecture: 40
;  Programmer: Arif Butt
;  dowhile.nasm
SECTION .text
    global start
_start:
    mov rax, 0 ;initialize counter
_loopbody:
    ; loop instructions comes here
    inc rax
    cmp rax, 5
    jl _loopbody

; exit gracefully
    mov rax, 60
    mov rdi, 0
    syscall
```

Assembling & Executing x86-64 Program

Example: while.nasm

High Level Code:

```
int i = 0;
while (i < 5){
    //loop instructions
     i++;
}
```

High Level Code:

```
for (int i = 0; i < 5; i++){
    //loop instructions
}
```

```
;  COAL Video Lecture: 40
;  Programmer: Arif Butt
;  while.nasm
SECTION .text
    global start
_start:
    mov rax, 0 ;initialize counter
    cmp rax, 5
    jge _endofloop
_loopbody:
    ; loop instructions comes here
    inc rax
    cmp rax, 5
    jl _loopbody
_endofloop:
; exit gracefully
    mov rax, 60
    mov rdi, 0
    syscall
```

Assembling & Executing x86-64 Program

Repetition Structure using x86 LOOP Instruction

LOOP <label>

- Performs a loop operation using the RCX, ECX or CX register as a counter
- Each time the **LOOP** instruction is executed, the count register is automatically decremented by one
- If the counter register rcx is not zero, a short jump (-128 to 127 from current rip value) is performed to the label
- The loop is terminated when the value of counter register becomes equal to zero, and program execution continues with the instruction following the **LOOP** instruction
- Other variations are **loope** and **loopne**

```
;  COAL Video Lecture: 40
;  Programmer: Arif Butt
;  loop.nasm
SECTION .text
   global start
_start:
   mov rcx, 5 ;initialize counter
_loopbody:
  ; loop instructions
   loop _loopbody

; exit gracefully
   mov rax, 60
   mov rdi, 0
   syscall
```

Assembling & Executing x86-64 Program

Example: final.nasm

```
SECTION .data
  msg   db    "PUCIT", 0xA
  len_msg   equ   $ - msg
SECTION .text
  global start
_start:
   _start:
   mov rcx, 5 ;initialize counter
_loopbody:
  mov rax, 1
  mov rdi, 1
  mov rsi, msg
  mov rdx, len_msg
  syscall
  loop _loopbody
; exit gracefully
   mov rax, 60
   mov rdi, 0
   syscall
```

Example: final.nasm

Note: Remember to preserve the value of loop counter inside the body of loop, as it may change during a system/library/procedure call

```
SECTION .data
  msg   db    "PUCIT", 0xA
  len_msg   equ   $ - msg
SECTION .text
  global start
_start:
    _start:
    mov rcx, 5 ;initialize counter
_loopbody:
  mov rax, 1
  mov rdi, 1
  mov rsi, msg
  mov rdx, len_msg
  push rcx
  syscall
  pop rcx
  loop _loopbody
; exit gracefully
    mov rax, 60
    mov rdi, 0
    syscall
```

Things To Do

Coming to office hours does NOT mean you are academically week!