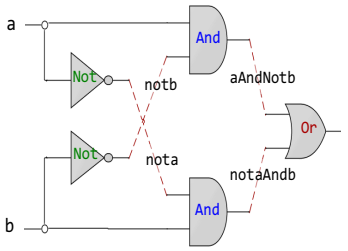
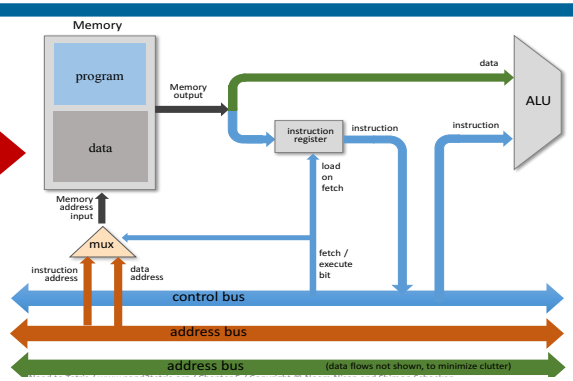
Computer Organization & Assembly Language Programming




```
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 # 35

Arithmetic Instructions - I

```
#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

- Review of x86-64 Registers & Tool Chain
- Categories of x86-64 Instruction Set
- Arithmetic Instructions
  - ADD & ADC
  - SUB & SBB
  - INC, DEC, NEG & CMP
  - CLC, STC, CMC
- Demo (***`arithmetic.nasm`***)

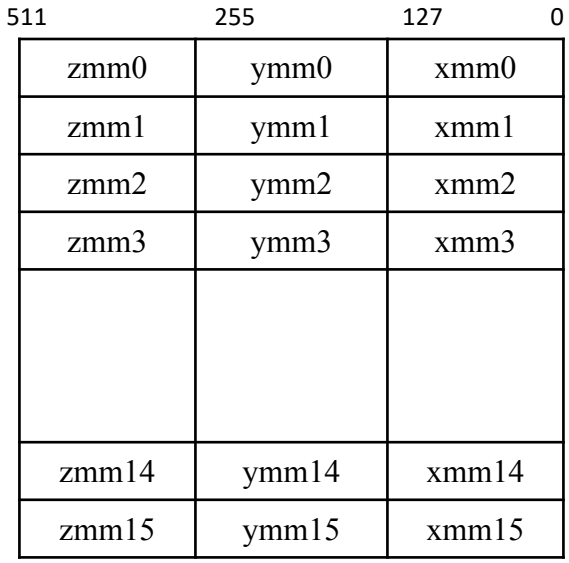
Review

Review: x86-64 Register Set

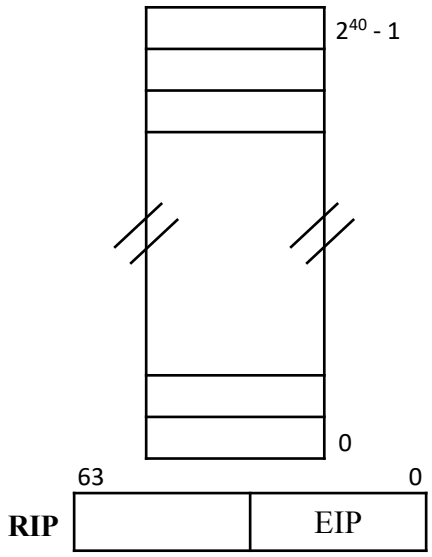
General Purpose Registers

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	r8d	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

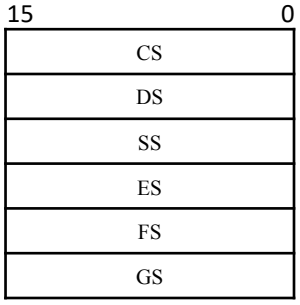
SSE Media Registers



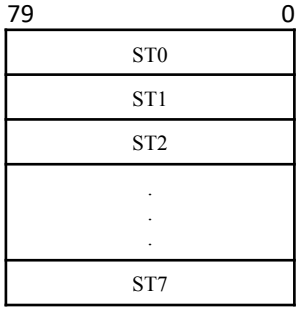
Memory

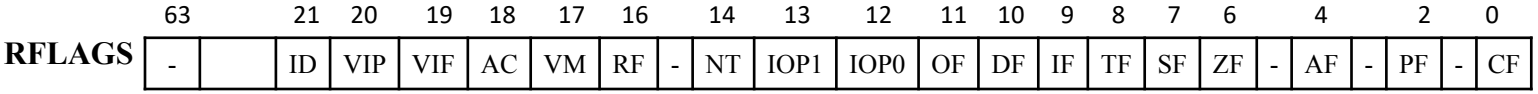


Segment Registers



FP Registers





Review: x86-64 Assembly Program

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
```

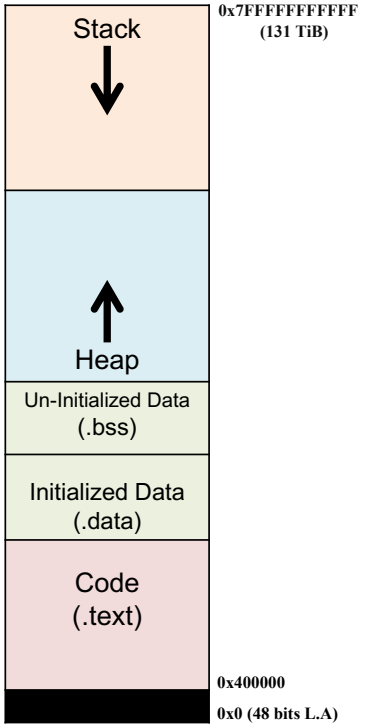
Assemble → **first.o**

```
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




```
$ nasm –felf64 first.nasm
$ ld first.o –o myexe
$./myexe
Learning is fun with Arif
```

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`

Arithmetic Instructions

Arithmetic Instructions

ADD

- **Format:** `ADD dest, source`
- **Operation:** Destination = Source + Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** AF, CF, OF, PF, SF, ZF

ADC

- **Format:** `ADC dest, source`
- **Operation:** Destination = Source + Destination + CF
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** AF, CF, OF, PF, SF, ZF

Arithmetic Instructions (cont…)

SUB

- **Format:** `SUB dest, source`
- **Operation:** Destination = Destination - Source
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/memory
- **Flags Affected:** AF, CF, OF, PF, SF, ZF

SBB

- **Format:** `SBB dest, source`
- **Operation:** Destination = Destination - (Source + CF)
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** AF, CF, OF, PF, SF, ZF

Arithmetic Instructions (cont…)

INC

- **Format:** `INC dest`
- **Operation:** Destination = Destination + 1
- **Operands:** Destination operand can be a reg/mem
- **Flags Affected:** AF, OF, PF, SF, ZF (CF is not affected)

DEC

- **Format:** `DEC dest`
- **Operation:** Destination = Destination – 1
- **Operands:** Destination operand can be a reg/mem
- **Flags Affected:** AF, CF, OF, PF, SF, ZF (CF is not affected)

Arithmetic Instructions (cont…)

NEG

- **Format:** `NEG dest`
- **Operation:** Destination = -Destination
- **Operands:** Destination operand can be a reg/mem
- **Flags Affected:** AF, CF, OF, PF, SF, ZF (CF is always 1, unless result is zero)

CMP

- **Format:** `CMP operand1, operand2`
- **Operation:** Subtracts second op from first op and set the flags
- **Operands:** Operand1 can be a reg/mem
  Operand2 can be an imm/reg/mem
- **Flags Affected:** AF, CF, OF, PF, SF, ZF

Arithmetic Instructions (cont…)

CLC

- **Format:** `CLC`
- **Operation:** Clears the CF
- **Flags Affected:** None other than CF

STC

- **Format:** `STC`
- **Operation:** Sets the CF
- **Flags Affected:** None other than CF

CMC

- **Format:** `CMC`
- **Operation:** Complements the CF
- **Flags Affected:** None other than CF

Assembling & Executing x86_64 Program

Things To Do

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