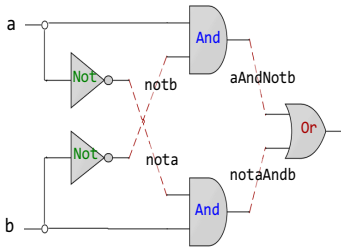
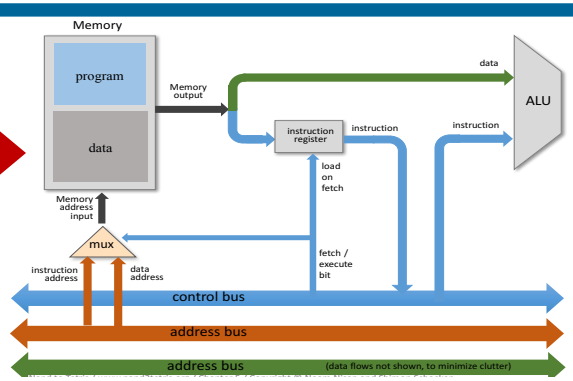
Computer Organization & Assembly Language Programming




```
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 # 37

Logical Operations

```
#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
- Logical Instructions
  - AND
  - OR
  - NOT
  - XOR
  - TEST
- Demo (***`logical.nasm`***)

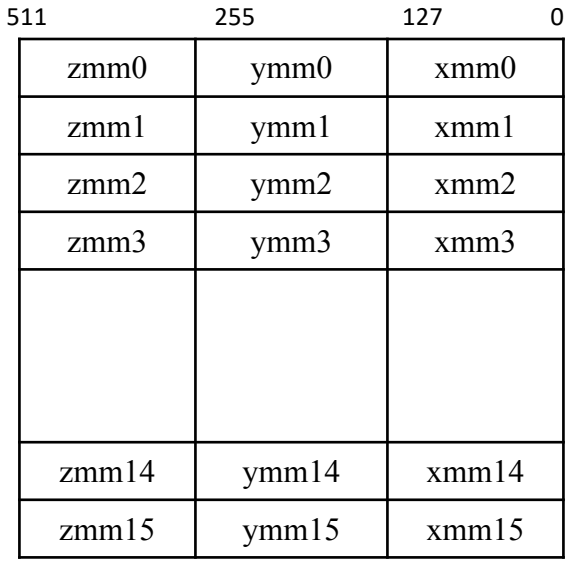
Recap

Review: x86-64 Register Set

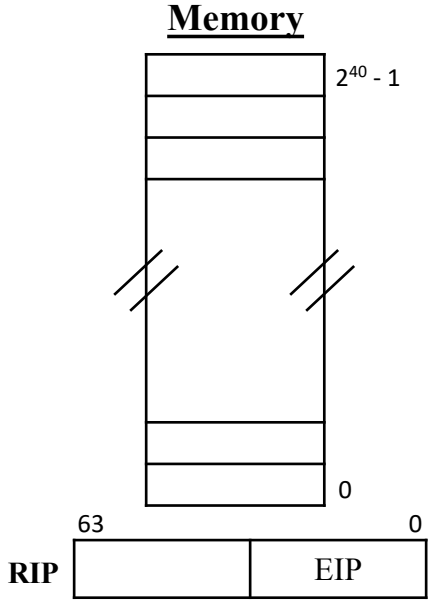
General Purpose Registers

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

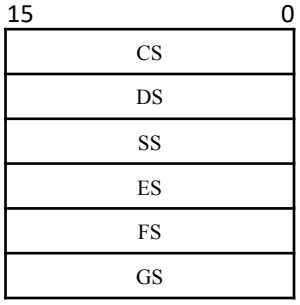
SSE Media Registers



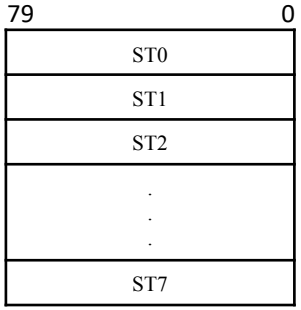
Memory

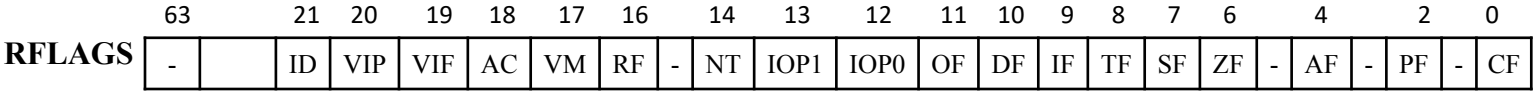


Segment Registers



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
```

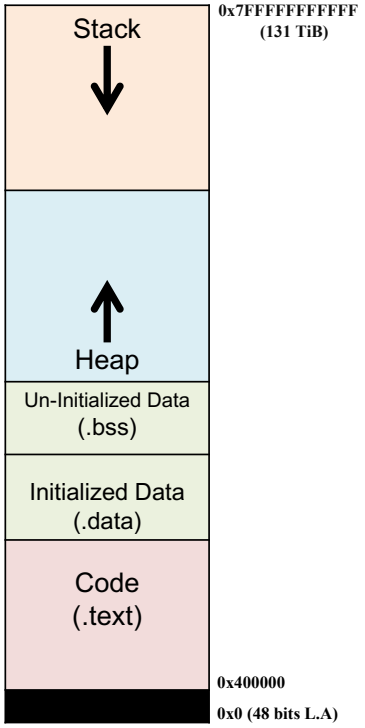
Assemble → **first.o**

```
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




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

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`

Logical Operations

Bitwise AND Instruction

AND

- **Format:** `AND dest, source`
- **Operation:** Destination = Source & Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to clear specific bits while preserving the others. A zero mask bit clears the corresponding bits, while a one mask bit preserves the corresponding bits
- **Example 1:**
  - To check whether a given number in AL register is even or odd

```
MOV al, 27d ;0001 1011
AND al, 01h ;0000 0001
          ;AL = 0000 0001 = 1(odd)
```

```
MOV al, 26d ;0001 1010
AND al, 01h ;0000 0001
          ;AL = 0000 0000 = 0(even)
```

Bitwise AND Instruction (cont…)

AND

- **Format:** `AND dest, source`
- **Operation:** Destination = Source & Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to clear specific bits while preserving the others. A zero mask bit clears the corresponding bits, while a one mask bit preserves the corresponding bits
- **Example 2:**
  - To clear the higher order bits of AX register to zero

```
MOV ax, 5bh   ;0101 1011
AND ax, 0Fh   ;0000 1111
              ;AL = 0000 1011 = 0bh
```

Bitwise AND Instruction (cont…)

AND

- **Format:** `AND dest, source`
- **Operation:** Destination = Source & Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to clear specific bits while preserving the others. A zero mask bit clears the corresponding bits, while a one mask bit preserves the corresponding bits
- **Example 3:**
  - To clear the sign bit of a number in AL register, leaving the other bits unchanged

```
MOV al, -5d   ;1111 1011
AND al, 7fh   ;0111 1111
          AL = 0111 1011
```

Bitwise AND Instruction (cont…)

AND

- **Format:** `AND dest, source`
- **Operation:** Destination = Source & Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to clear specific bits while preserving the others. A zero mask bit clears the corresponding bits, while a one mask bit preserves the corresponding bits
- **Example 4:**
  - To clear the even number of bits of a register, leaving the other bits unchanged

```
MOV al, 27d   ;0001 1011
AND al, aah   ;1010 1010
          AL = 0000 1010
```

Bitwise AND Instruction (cont…)

AND

- **Format:** `AND dest, source`
- **Operation:** Destination = Source & Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to clear specific bits while preserving the others. A zero mask bit clears the corresponding bits, while a one mask bit preserves the corresponding bits
- **Example 5:**
  - Suppose AL register contains the ASCII code of a lower case alphabet, convert it to upper case

```
MOV al, 61h   ; 0110 0001 (97d or 'a')
AND al, dfh   ; 1101 1111
                ;AL = 0100 0001 (65d or 'A')
```

Bitwise AND Instruction (cont…)

AND

- **Format:** `AND dest, source`
- **Operation:** Destination = Source & Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to clear specific bits while preserving the others. A zero mask bit clears the corresponding bits, while a one mask bit preserves the corresponding bits
- **Example 6:**
  - Suppose AL register contain a ASCII code of a decimal digit, convert it to decimal number (atoi)

```
MOV al, 32h   ; 0011 0010 (50d)
AND al, 0fh   ; 0000 1111
                ;AL = 0000 0010 (2d)
```

Bitwise OR Instruction

OR

- **Format:** `AND dest, source`
- **Operation:** Destination = Source | Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to set specific bits while preserving the others. A one mask bit sets the corresponding bits, while a zero mask bit preserves the corresponding bits
- **Example 1:**
  - To set the most and the least significant bits of a number, leaving the other bits unchanged, OR the number with 81h (for 8 bit number)

```
MOV al, 72h   ;0111 0010
AND al, 81h   ;1000 0001
          AL = 1111 0011
```

Bitwise OR Instruction (cont…)

OR

- **Format:** `AND dest, source`
- **Operation:** Destination = Source | Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to set specific bits while preserving the others. A one mask bit sets the corresponding bits, while a zero mask bit preserves the corresponding bits
- **Example 2:**
  - Suppose AL register contain a upper case alphabet, convert it to lower case

```
MOV al, 5ah   ; 0101 1010 (90d or 'Z')
OR  al, 20h   ; 0010 0000
              ;AL = 0111 1010 (7ah or 122d or 'z')
```

Bitwise OR Instruction (cont…)

OR

- **Format:** `AND dest, source`
- **Operation:** Destination = Source | Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to set specific bits while preserving the others. A one mask bit sets the corresponding bits, while a zero mask bit preserves the corresponding bits
- **Example 3:**
  - Suppose AL register contain a decimal digit, covert it to its corresponding ASCII code (itoa)

```
MOV al, 5d  ; 0000 0101
OR  al, 30h ; 0011 0000
            ;AL = 0011 0101 (53d)
```

Bitwise NOT Instruction

NOT

- **Format:** `NOT dest`
- **Operation: :** Destination = ~ Destination
- **Operands:** Destination operand can be a reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to perform 1's complement
- **Example:**
  - Calculate the 1s complement of a number

```
MOV al, 72h ;0111 0010
NOT al
            ; AL = 1000 1101 (141 if unsigned, else -115)
```

Bitwise XOR Instruction (cont…)

XOR

- **Format:** `XOR dest, source`
- **Operation: :** Destination = Source ^ Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to complement specific bits, while preserving the others. A one mask bit complements the corresponding bits, while a zero mask bit preserves the corresponding bits
- **Example 1:**
  - To clear the entire register bits to zero, XOR the register with itself

```
MOV al, 42h   ; 0100 0010
XOR al, al    ; 0100 0010
              ;AL = 0000 0000
```

Bitwise XOR Instruction (cont…)

XOR

- **Format:** `XOR dest, source`
- **Operation: :** Destination = Source ^ Destination
- **Operands:** Destination operand can be a reg/mem
  Source operand can be an imm/reg/mem
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to complement specific bits, while preserving the others. A one mask bit complements the corresponding bits, while a zero mask bit preserves the corresponding bits
- **Example 2:**
  - To convert upper case alphabet to lower case alphabet and vice versa

```
MOV al, 66d ;0100 0010 ('B')
XOR al, 20h ;0010 0000
       ;AL = 0110 0010 ('b')
```

```
MOV al, 98d ;0110 0010 ('b')
XOR al, 20h ;0010 0000
       ;AL = 0100 0010 ('B')
```

Bitwise TEST Instruction

TEST

- **Format:** `TEST operand1, operand2`
- **Operation: :** Perform bitwise AND of operand1 and operand2
- **Operands:** Operand1 can be a reg/mem
  Operand2 can be an imm/reg
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to test whether bit(s) in the first operand is/are having a value of 1. Mask should contain one in the bit positions to be tested and zero elsewhere
- **Example 1:** To check if AL contains a positive or negative number

```
MOV  al, 5d   ; 0000 0101
TEST al, 80h  ; 1000 0000
              ;ZF = 1
```

```
MOV  al, -5d  ; 1111 1011
TEST al, 80h  ; 1000 0000
              ;ZF = 0
```

If ZF is set to one, the number inside register is positive else negative

Bitwise TEST Instruction (cont…)

TEST

- **Format:** `TEST operand1, operand2`
- **Operation: :** Perform bitwise AND of operand1 and operand2
- **Operands:** Operand1 can be a reg/mem
  Operand2 can be an imm/reg
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to test whether bit(s) in the first operand is/are having a value of 1. Mask should contain one in the bit positions to be tested and zero elsewhere
- **Example 2:** To check if AL contains a zero, we need to test all the bits

```
MOV  al, 5d  ; 0000 0101
TEST al, ffh ; 1111 1111
               ;ZF = 0
```

```
MOV  al, 00h  ; 0000 0000
TEST al, ffh  ; 1111 1111
                ;ZF = 1
```

If ZF is set to one, the number inside register is zero

Bitwise TEST Instruction (cont…)

TEST

- **Format:** `TEST operand1, operand2`
- **Operation: :** Perform bitwise AND of operand1 and operand2
- **Operands:** Operand1 can be a reg/mem
  Operand2 can be an imm/reg
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to test whether bit(s) in the first operand is/are having a value of 1. Mask should contain one in the bit positions to be tested and zero elsewhere
- **Example 3:** To check if AL contains an even or an odd number, we need to test the least significant bit

```
MOV  al, 5d  ; 0000 0101
TEST al, 01h ; 0000 0001
               ;ZF = 0
```

```
MOV  al, 6d  ; 0000 0110
TEST al, 01h ; 0000 0001
               ;ZF = 1
```

If ZF is set to one, the number inside register is even else odd

Bitwise TEST Instruction (cont…)

TEST

- **Format:** `TEST operand1, operand2`
- **Operation: :** Perform bitwise AND of operand1 and operand2
- **Operands:** Operand1 can be a reg/mem
  Operand2 can be an imm/reg
- **Flags Affected:** The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined
- **Usage:** Used to test whether bit(s) in the first operand is/are having a value of 1. Mask should contain one in the bit positions to be tested and zero elsewhere
- **Example 4:** To check if AL contains ASCII code of lower or upper case alphabet, we need to test the bit#5 having weight of 32

```
MOV  al, 65d ;('A') 0100 0001
TEST al, 32d ;      0010 0000
                 ;ZF = 1
```

```
MOV  al, 97d  ; 0110 0001
TEST al, 32d  ; 0010 0000
                ;ZF = 0
```

If ZF is set to one, the ASCII of alphabet inside register is upper case else lower

Assembling & Executing x86-64 Program

Things To Do

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