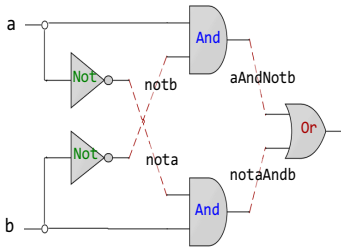
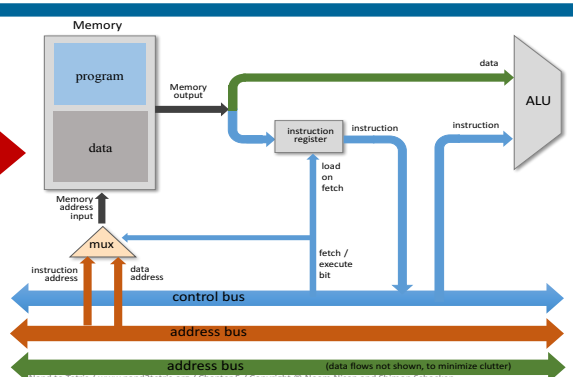
Computer Organization & Assembly Language Programming




```
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 # 43

Functions in Assembly Language - 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: Calling and Returning from Assembly Functions
- Passing and Returning Values to and from Functions
  - Demo (***proc1.nasm***)
  - Demo (***proc2.nasm***)
  - Demo (***proc3.nasm***)
- Displaying a Single Digit Decimal Number on Screen
  - Demo (***proc4.nasm***)
- Displaying an Integer on Screen
  - Demo (***proc5.nasm***)
- Multi-File Assembly Program
  - Demo (***proc6.nasm*** and ***myfunctions.nasm***)

Recap

Calling and Returning from Assembly Functions

```
; COAL Video Lecture: 42
;  proc1.nasm
SECTION .data
    msg db "PUCIT", 0xA
    len_msg equ $ - msg
SECTION .text
    global main
    global printmsg
main:
    call printmsg
    mov rax,60
    mov rdi, 0
    syscall

printmsg:
    mov rax, 1
    mov rdi,1
    mov rsi,msg
    mov rdx,26
    syscall
    ret
```

rip

return addr

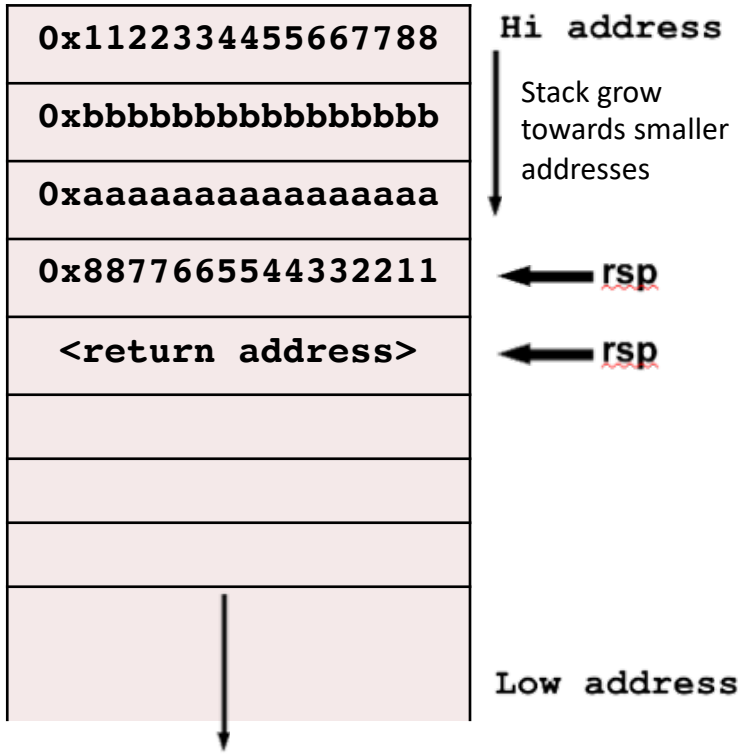
rip

rip

push rip

jmp printmsg

pop rip



Caller Saved vs Callee Saved Registers

Scratch / Callee Owned / Caller Saved

- In x86-64, the nine general purpose registers: `rax, rdi, rsi, rdx, rcx, r8, r9, r10, r11` fall in this category
- They are callee (`printmsg`) owned, therefore the callee can freely use these registers
- They are caller (`main`) saved, therefore, (if the caller wants to preserve them) the caller must push them before making the function call and later pop them after the function returns
- Used for passing arguments to functions

Preserved / Caller Owned / Callee Saved

- In x86-64, the seven general purpose registers: `rbx, rbp, rsp, r12-r15` fall in this category
- They are caller (`main`) owned, therefore, the callee CANNOT freely use these registers
- They are callee (`printmsg`) saved, therefore, (if the callee wants to use them) the callee itself must push them at the start of function and pop them at the end of function
- Used for local state of the caller that needs to preserved across further function calls

Passing and Returning Values from Functions

Passing and Returning Values from Functions

- A function may have arguments/parameters, which might be integer/floating point values as well as addresses pointing to data
- This enable a function to operate on different data with each call
- Other than parameters, most functions have a return value which is commonly an indicator of success or failure
- In the 16 and 32 bit days, since there were only eight general purpose registers, therefore, all the arguments were passed by the caller to the callee by pushing the arguments on the stack

Passing and Returning Values from Functions

- On x86-64, Linux, Solaris and Mac OS use a function call protocol called the **System-V AMD64 ABI**. In which first six integer parameters are passed via registers and first eight floating point parameters via `xmm0` to `xmm7` registers (rest on the runtime stack)
- On x86-64, MS Windows use **MS X64 Calling Convention**. In which first four integer parameters are passed via registers and first four floating point parameters via `xmm0` to `xmm3` registers (rest on the runtime stack)
- Both Linux and MS Windows use **`rax`** register to return integer values and **`xmm0`** register to return floating point values

Parameter	Qword	Dword
1	rcx	ecx
2	rdx	edx
3	r8	r8d
4	r9	r9d
>4	Stack	Stack

Parameter	Qword	Dword
1	rdi	edi
2	rsi	esi
3	rdx	edx
4	rcx	ecx
5	r8	r8d
6	r9	r9d
>6	Stack	Stack

Returning Value from Function

- The `main` function is called by the `_start` function, which contains the startup code for the C runtime environment
- Before calling the `main` function, it do some stuff other than populating the `argc` and `argv` as per the command line arguments passed to the program
- A function can return to its caller using the `ret` instruction and before that must place the value to be returned inside the `rax` register
- A function may return to the `_start` function using the `exit` call

```
; COAL Video Lecture: 43
;  proc1.nasm
SECTION .text
   global main
main:
; instructions comes here
   mov rax,60
   mov rdi, 54
   syscall      ;exit(54)
```

```
; COAL Video Lecture: 43
;  proc1.nasm
SECTION .text
   global main
main:
; instructions comes here
   mov rax, 54
   ret
```

```
$ nasm –felf64 proc1.nasm
$ gcc proc1.o –o myexe
$ ./myexe
$ echo $?
54
```

Assembling & Executing x86-64 Program

Passing Arguments to Function: proc2.nasm

```
; COAL Video Lecture: 43
;  proc2.nasm
SECTION .text
  global main
  main:
     mov rdi, 50
     mov rsi, -15
     mov rdx, 35
     call sumOfThree
     ret
  sumOfThree:
     add rdi, rsi
     add rdi, rdx
     mov rax, rdi
     ret
```

```
$ nasm –felf64 proc2.nasm
$ gcc proc2.o –o myexe
$ ./myexe
$ echo $?
70
```

The C exit status, and bash return code, both cover the range 0-255, so you cannot return a value >255. Moreover, Bash uses 127 code for command not found and 128-255 for different signals, so one must use the return value in the range of 0 to 126

Count # of 1-bits in a 64 bit Data Value: proc3.nasm

```
; COAL Video Lecture: 43
;  proc3.nasm
SECTION .text
  global main
main:
   mov rdi, 0x12481248ffff0000
   call countOnes
   ret

countOnes:
   mov rcx, 64
   xor rax, rax
_repeat:
      mov rsi, rdi
      and rsi, 1
      add rax, rsi
      sar rdi, 1
      loop _repeat
   ret
```

```
$ nasm –felf64 proc3.nasm
$ gcc proc3.o –o myexe
$ ./myexe
$ echo $?
24
```

Assembling & Executing x86-64 Program

Displaying a Single Decimal Digit on Screen

```
; COAL Video Lecture: 43
;  proc4.nasm
SECTION .data
   str: db 0x0,0xa
SECTION .text
   global _start
_start:
   mov rdi, 7
   call printdigit
; exit gracefully
   mov rax, 60
   mov rdi, 0
   syscall
printdigit:
   add rdi, 48 ; convert digit to ascii
   mov byte [str], dil
   mov rax, 1
   mov rdi, 1
   mov rsi, str
   mov rdx, 2
   syscall
   ret
```

Char	ASCII Code (Decimal)
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57

Assembling & Executing x86-64 Program

Multi-File Assembly Program

```
; COAL Video Lecture: 43
;  proc6.nasm
SECTION .text
  global main
  extern sumOfThree
  extern printUnsignedInt
main:
  mov rdi, 1500
  mov rsi, 411
  mov rdx, 110
  call sumOfThree
  mov rdi, rax
  call printUnsignedInt
  ret
```

```
$ nasm –felf64 proc6.nasm
```

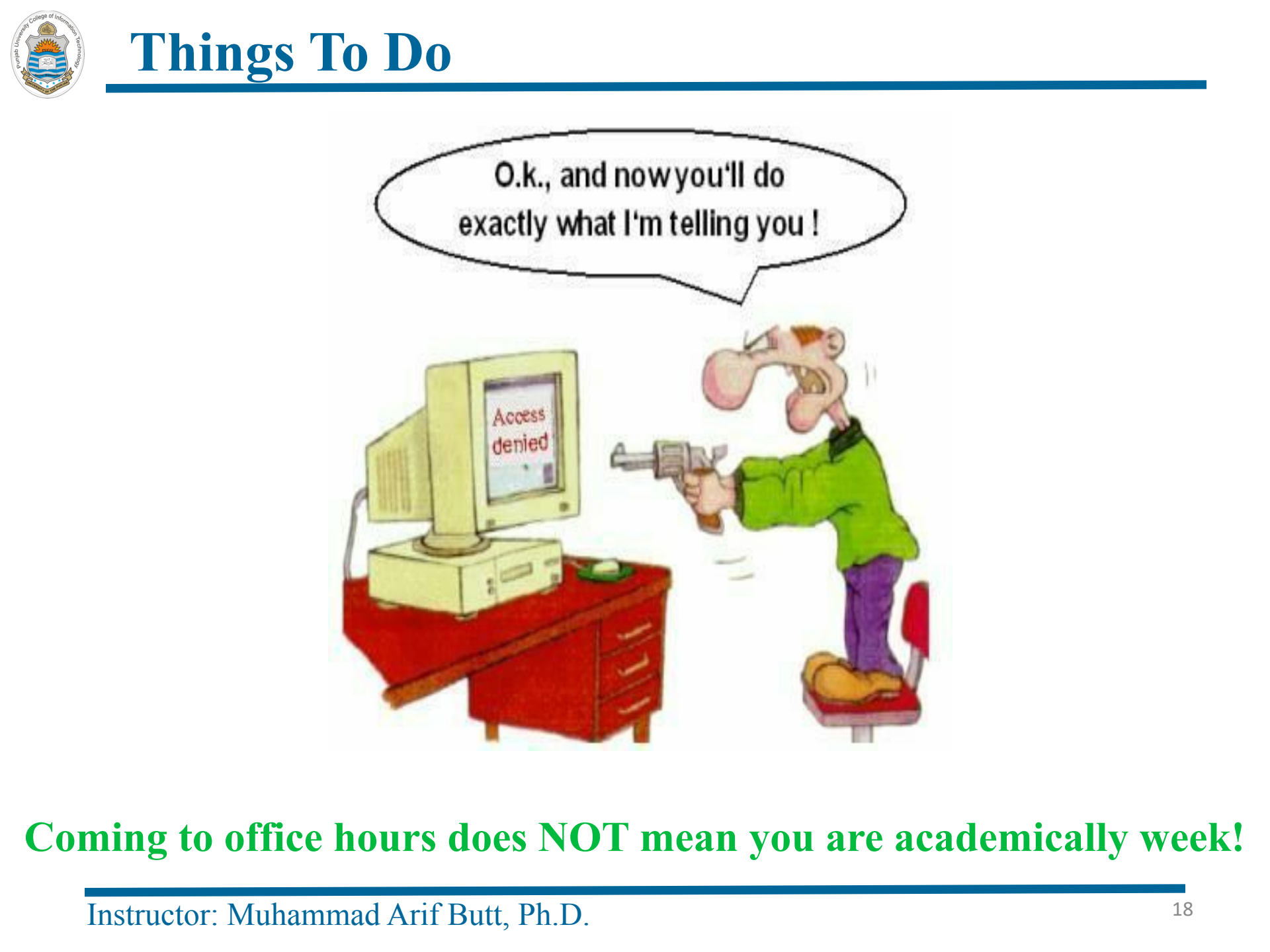
```
; COAL Video Lecture: 43
;  myfunctions.nasm
SECTION .text
  global sumOfThree
  global printUnsignedInt
sumOfThree:
   ;code of the function
   ret

printUnsignedInt:
   ;code of the function
   ret
```

```
$ nasm –felf64 myfunctions.nasm
```

```
$ gcc --static proc6.o myfunctions.o -o myexe
$ ./myexe
2021
```

Assembling & Executing x86-64 Program



Things To Do

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