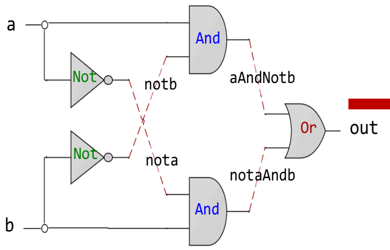
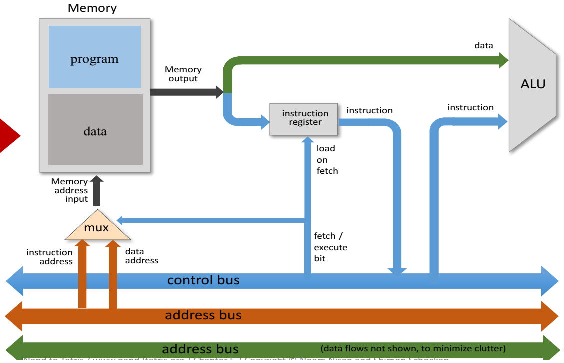
Computer Organization & Assembly Language Programming



```
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 # 32

Data Types and Endianness

```
#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

- Data Types & Special Tokens in NASM
  - For Initialized Data
  - For Un-initialized Data
- Endianness
  - Little Endian Machines (x86, ARM)
  - Big Endian Machines (MIPS, MC68000)

Data Types

Data Types for Initialized Data

- A define directive sets aside storage in memory for variables. The general format for variable declaration is:

```
<varName>  <defineDir>  <iv> [,iv]
```

- NASM provides various define directives for reserving storage space for variables as shown in the table:

Directives	Purpose	Storage Space
DB	Define Byte	8 bits
DW	Define Word	16 bits
DD	Define Double Word	32 bits
DQ	Define Quad Word	64 bits
DT	Define Ten Bytes	80 bits
DO	IEEE-754 Quad	128 bits

- **Examples:**

```
bVar1   db   10              ;byte variable
bVar2   db   0x54            ;byte variable
cVar    db   "H"             ;single character
msg     db   "Hello!"        ;string variable
wVar    dw   0x1234          ;16-bit variable
dVar    dd   0x12345678      ;32-bit variable
qVar    dq   0x123456789abcdef0     ;64-bit variable
bVar3   db   0x3a,0xbb,0x43    ;array of three bytes
arr     dd   10,20,30          ;array of 32-bit var
flt1    dd   3.14159           ;IEEE-754 single precision
flt2    dt   1.2345e20         ;IEEE-754 double precision
flt3    do   3.14159           ;IEEE-754 quad precision
```

Data Types for Un-Initialized Data

- To declare uninitialized data, you use the RES (reserve) directive to reserve uninitialized space in memory for your variables
- The RES (reserve) directives take a single operand that specifies the number of units of space to be reserved. The assembler allocates contiguous memory for multiple variable definitions
- Each define directive has a related reserve directive. NASM present various RES directives, as shown in the table
- The general format for variable declaration is:

```
<varName>  <resDirective>  <count>
```

Examples:

```
bArr    resb   12    ;reserve 12 bytes
wArr    resw   5    ;reserve 5 words
dArr    resd   1    ;reserve 1 double word
qArr    resq   10   ;reserve 10 quad words
```

Directives	Purpose
RESB	Reserve Byte
RESW	Reserve Word
RESD	Reserve Double Word
RESQ	Reserve Quad Word
REST	Reserve 10 Bytes
RESO	Reserve 16 Bytes

Special Tokens used by NASM

- **`$:`** evaluates to the current line
- **`$$:`** evaluates to the beginning of the current section
- **`equ:`** is used to define a constant
- **`times:`** is used to repeat data and instruction

Examples:

```
msg     db    “Hello World!”,0xa ;A string message
msglen equ   $ - msg          ;Calculates the length of the string
zerobuf  times 64   db   0    ;Initialize 64 bytes buffer with zero
```

Example Code: `datatypes.nasm`

```
SECTION .data
  msg: db "Learning is fun with Arif Butt!",0xa
  len_msg: equ $ - msg
  EXIT_STATUS equ 0
  var1: db 0x11, 0x22
  var2: dw 0x3344
  var3: dd 0xaabbccdd
  var4: dq 0xaabbccdd11223344
  repeat_buffer: times 128 db 0xAA

SECTION .bss
  buffer: resb 64

SECTION .text
  global _start
_start:
  mov rax,1
  mov rdi,1
  mov rsi,msg
  mov rdx,len_msg
  syscall
```

```
; cont…

  mov r8, var4
  mov r9, [var4]

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

Code taken from SLAE by Vivek Ramachandran

Demo

Demo

32/datatypes.nasm

x86-64 is Little Endian

Endianness of a Machine

- In computing, endianness is the order in which multi-byte data is stored or retrieved from computer memory. Endianness is primarily expressed as big-endian (BE) or little-endian (LE)
- A big-endian system stores the most significant byte of a word at the smallest memory address (MSB first). Used by MIPS, MC68000 and Internet
- A little-endian system, in contrast, stores the least-significant byte of a word at the smallest address (LSB first). Used by x86 and ARM
- Let us store a 8 byte number `0x1122334455667788` in memory

Low Address

Hi Address

Addresses	0	1	2	3	4	5	6	7
Little Endian	Byte 0 88	Byte 1 77	Byte 2 66	Byte 3 55	Byte 4 44	Byte 5 33	Byte 6 22	Byte 7 11
Big Endian	Byte 7 11	Byte 6 22	Byte 5 33	Byte 4 44	Byte 3 55	Byte 2 66	Byte 1 77	Byte 0 88

Example Code: `endian.nasm`

```
SECTION .text
global _start
_start:
   mov rax, [var1]
   mov rbx, [var2]
   mov rax,60
   mov rdi,0
   syscall
SECTION .data
   var1: db 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
   var2: db 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11
```

Code taken from SLAE by Vivek Ramachandran

Demo

Demo

`32/endianness.nasm`

Things To Do

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