NASM Programming

Netwide Assembler

1) Data Types
   - Nibble (4 bits)
   - Byte (8 bits)
   - Word (16 bits)
   - Double Word (32 bits)
   - Quad Word (64 bits)
   - Ten Bytes (80 bits) – Maths Coprocessor (80387)

2) Sections

   section .data ⇒ For defining (int x = 2)

   section .bss ⇒ For declaration (int x) or reserving buffer

   section .text ⇒ For main program

   global start ⇒ Where your program execution starts

   (start can be user defined)

3) Definition Directive ⇒ section .data

Directives are also called as the directions given to the Assembler. They are never executed by the Assemblers.

```
db (define byte)
dw (define word)
dd (define doubleword)
dq (define quadword)
```

eg array dw 0032h, 0041h, 3216h
   cnt db 2

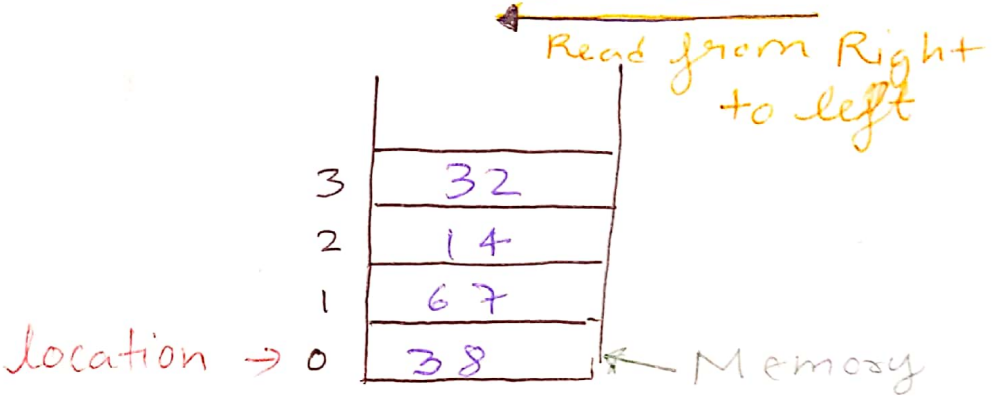
4) Declaration Directive → Section .bss

resb (reserve byte)
resw (reserve word)
resd (reserve doubleword)
resq (reserve quadword)

eg pcnt resb 50

5) i) while defining

num db 32146738h

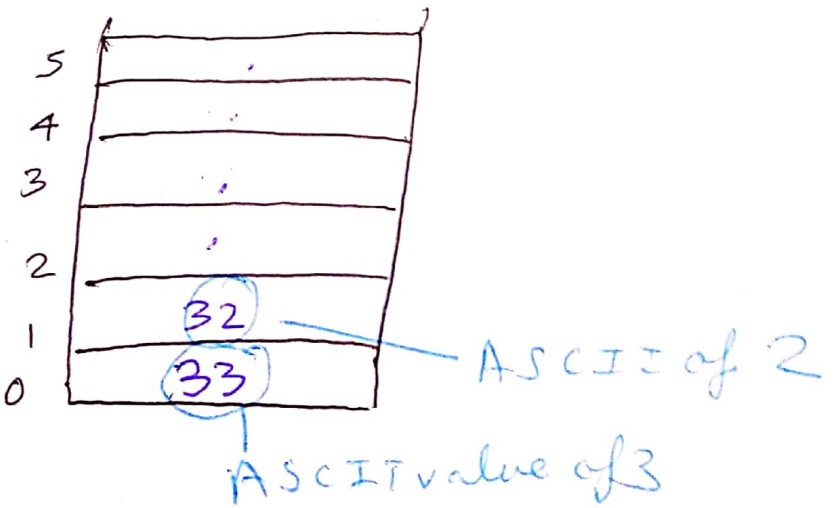


ii) While getting input from user

num resb 8

i/p ← num - 32146 7 → Read Left to Right

each digit ASCII value of digit is inserted at one memory location



6)

Newline (0AH)

welmsg db 10, "Nasm Tutorial", 10

1 2 3 4 5 6 7 8 9 10 11 12 13 14

welmsg (0)

$ ⇒ current pointer (14)

wel_len equ $-welmsg

14 - 0

wel_len = 14

7) ; Comment (No multiline comment)

8)

cnt db 7

cntl equ 7 – Immediate Operand (Constant Value) (No memory fetch cycle)

mov rcx, [cnt]

mov rcx, cntl ⇒ No need of [ ]

[cnt] ⇒ *cnt (like in C++)

9) Memory Addressing Directives

byte

word

dword

qword

eg. mov al, byte [num]

Only 1 byte to be transferred to 'al' register of num

10) Macros ⇒ For reading, write, display, file handling

%macro macroname No. of arg (optional)

%end macro

ii) i) To read from user through keyboard

```
%.macro read 2
    mov rax, 0      - system call no
    mov rdi, 0      - file discriptor (0 to read, 1 to display), file handler
    mov rsi, %.1    - buffer        (argument)
    mov rdx, %.2    - length of buffer
    Syscall - To give control to Kernel to execute this (PL=0)
%.endmacro
```

ii) To display on monitor

```
%.macro display 2
    mov rax, 1  - to display
    mov rdi, 1  - to display
    mov rsi, %.1
    mov rdx, %.2
    syscall
%.endmacro
```

iii) To exit

```
    mov rax, 60  → To exit the program and also destroy the process
    mov rdi, 0
    syscall
```

iv) To unlink or delete file

```
    move rax, 87 ; unlink system call
    move rdi, name ; name of file
    syscall
```

* There are three types of file

1. Owner
2. Group
3. Others

r - read
w - write
x - execute

Owner	Group	Others
rwx	rwx	rwx
111	111	111
7	7	7

⇒ File Permission to everyone

v) File opening

```
%.macro fopen 1
    mov rax, 2  ; open file
    mov rdi, %.1  ; filename
    mov rsi, 2  ; mode RW
    mov rdx, 0o7770 - File Permissions
    syscall
%.endmacro
```

If file open successfully its file handle returned in RAX, else -1H is returned in RAX

0 - Read
1 - Write
2 - Read/Write

vi) File closing

```
%.macro fclose 1
    mov rax, 3 ; close file
    mov rdi, %.1 ; file handler
    syscall
%.endmacro
```

vii) File writing

```
%.macro fwrite 3
    mov rax, 1 ; write/print
    mov rdi, %.1 ; file handle (like pointer)
    mov rsi, %.2 ; buffer
    mov rdx, %.3 ; buffer len
    syscall
%.endmacro
```

viii) File reading

```
%macro fread 3
mov rax,0 ;read
mov rdi,%1 ;filehandle
mov rsi,%2 ;buffer
mov rdx,%3 ;bufferlen
syscall
%endmacro
```

- After reading from file actual no. of characters read is stored in RAX.

Syscall Nos

0 - to read
1 - to display/write
2 - open file
3 - close file

12) Procedure Declaration

```
label:
    {
    ret → to return
call label → to call procedure
```

Near Procedure → Data in same segment
Far Procedure → Data in different segments

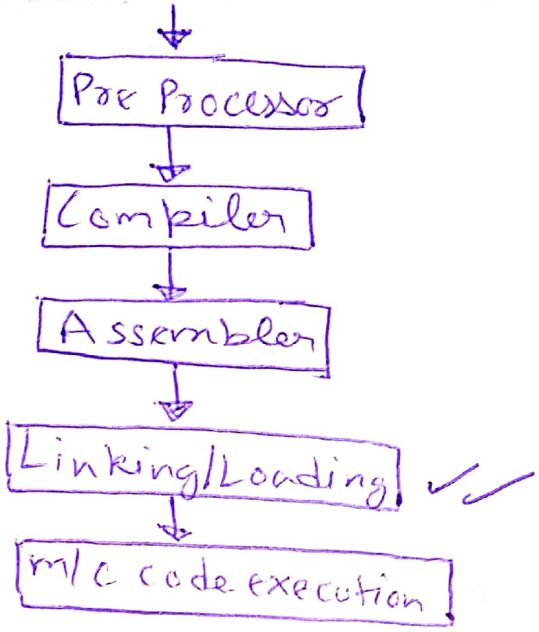
% include "macro.asm" ⇒ To include any asm file

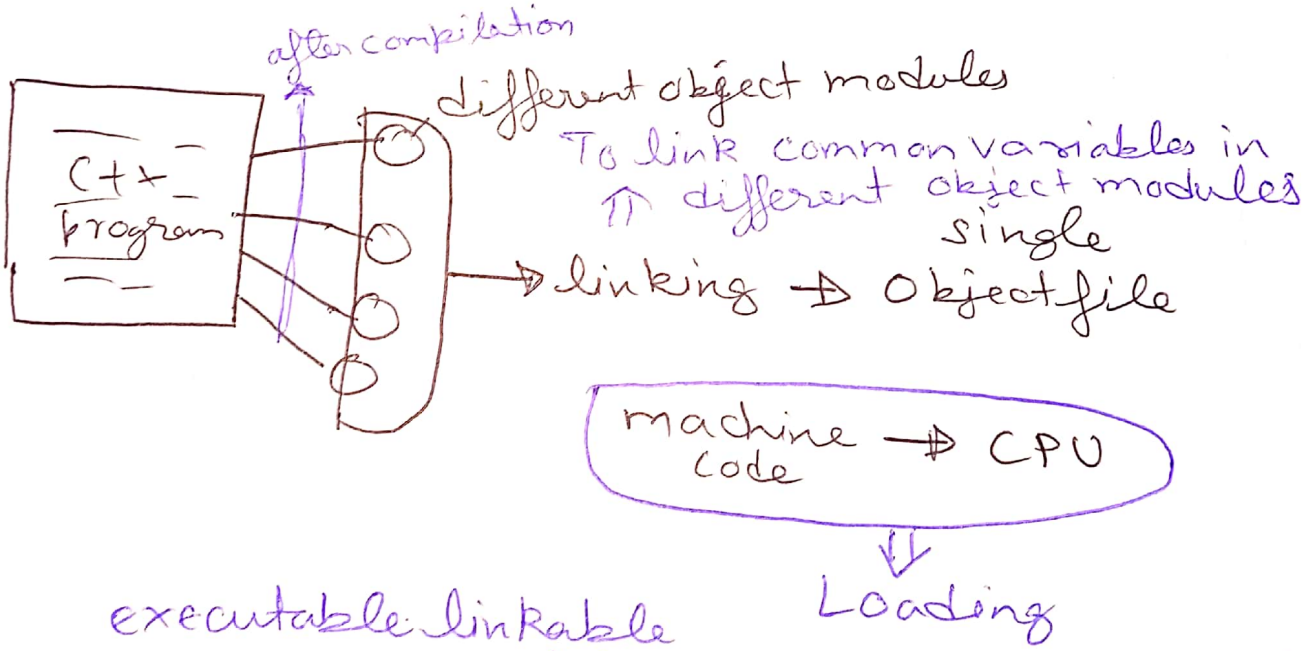
global ⇒ To declare variables global in folder

extern ⇒ To use any global variable

13)

HLL - C++, Java etc





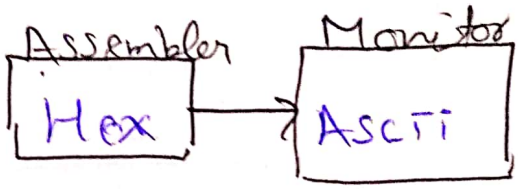
executable linkable file

nasm -f elf64 1.asm
nasm -f elf64 2.asm } Assembling step

ld -o a 1.o 2.o ⇒ Linking and Loading step
(a → alias (user defined))

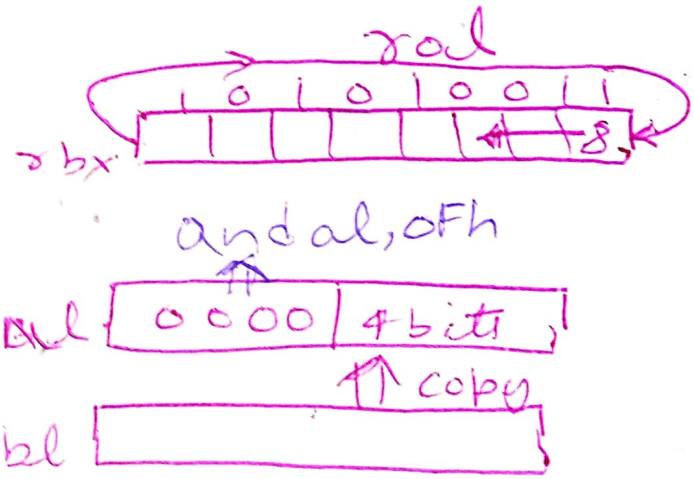
./a ⇒ execute machine code

14) Hex to Ascii (Display)



8byte ⇒ 2x8 = 16 Hex nos

```
mov rbx, [num]
mov rdi, disbuff
mov cx, 16 ⇒ 16 Hex nos
UP1:
rol rbx, 04
mov al, bl
and al, 0Fh
cmp al, 09h
jg  add-37
add al, 30h
jmp SKIP1
add_37: add al, 37h
SKIP1: mov [rdi], al
  inc rdi
  loop UP1   ⇒ until rcx becomes 0
scall 1, 1, disbuff, 16
```



15) Ascii to Hex (To take input)


```
Xor rax, rax
Xor rbx, rbx   } make contents of registers 0
Xor rcx, rcx
mov rcx, 16
mov rsi, num_ascii
UP2: rol rbx, 04
mov al, byte[rsi]
cmp al, 39h
jg sub_37
sub al, 30h
jmp SKIP2
sub_37: sub al, 37h
SKIP2:
  add rbx, rax
  inc rsi
  loop UP2
  ret
```