1 Minimum Program

1.1 Edit, Compile, Link and Execute

1. Open a Terminal (Console)

2. Create the folder *mynasm* in your home directory.

   ```
   $ cd; mkdir mynasm
   ```

   where $ is the user prompt of the console.

3. Create source file *mynasm/a01.asm* using *gedit*

   ```
   $ cd mynasm; gedit a01.asm &
   ```

   edit (insert) the following code and save.

   ```
   section .data
   section .bss
   section .text
   global _start

   _start:

    ; .. Your code

   mov ebx,0
   mov eax,1
   int 0x80
   ```

4. **Compile** the source file *a01.asm* by using the assembler *nasm*

   ```
   $ nasm -f elf -g -F stabs -o obj.o -l lst.l a01.asm
   ```

   where

   - Compiles the source file `a01.asm` into an object file
   - `-o obj.o` defines the object file to be `obj.o`
   - `-f elf` defines the object file format as ELF (Executable and Linkable Format)
   - `-g -F stabs` includes debugging info
   - `-l lst.l` defines the list file name as `lst.l`

5. textbfLink object file `obj.o` to create the executable file verb|exe.x|.

   ```
   $ ld -o exe.x obj.o
   ```

   where

   - this links the object file `obj.o` and creates the executable
   - `-o exe.x` makes `exe.x` to be the executable file

6. **Execute**

   ```
   $ ./exe.x
   ```

1.2 Compile and Link using make utility

1. Create a file named *Makefile* in the same directory where the source file *a01.asm* is, and insert the following.
   ```
   exe.x:obj.o
   <tab>ld -o exe.x obj.o
   obj.o:a01.asm
   <tab>nasm -f elf -g -F stabs -l lst.l -o obj.o a01.a
   ```

2. Execute the *make* utility to make the executable *exe.x*
   ```
   $ make
   ```

2 Understanding the Min.Prog

2.1 Code Content

- `section .data`
  Begining of the initialized memory (data) allocation area
- `section .bss`
  Begining of the uninitialized memory (data) allocation area
- `global _start`
  `_start:;`
  Program begins here
- `mov ebx,0`
  Assembly instruction
  `mov eax,1`
  Assembly instruction
  `int 0x80`
  Assembly instruction, Program ends here

2.2 Assembly Instruction Format

```
[label:] operator [operand1, operand2, ..] [;comment]
```

Examples

```
         mov ebx,0
mylable: mov ebx,0
mylable: mov ebx,0 ; set ebx to zero
         jmp mylable
```

- Program contains one instruction per line
- Fields in [ ] are optional
- **label** serves to label an instruction, identifier or constant. It represents the address of the proceeding instruction.
- operator identifies the operation (e.g. add, or)
- operands specify the data required by the operation
- operator manipulates operands and generate at most one result.
- Results are stored in registers or memory as specified by the instruction
- comments begin with a semicolon (;) and extend to the end of the line

- There are three categories of operands in an instruction
  - Constant
  - Register
  - Memory

  Examples
  ```
  mov eax, 324
  mov eax, ebx
  mov eax, [faculty]
  ```

2.3 Other Assembly Statements

Directives Provide information to assembler on various aspects of the assembly process, Non-executable. Example, section **.data**

Macros A shorthand notation for a group of statements, A sophisticated text substitution mechanism with parameters.

2.4 How Execution Occurs

- Each line contains a single assembly instruction (statement)
- Execution starts at `_start:`
- Then, instructions are executed sequentially

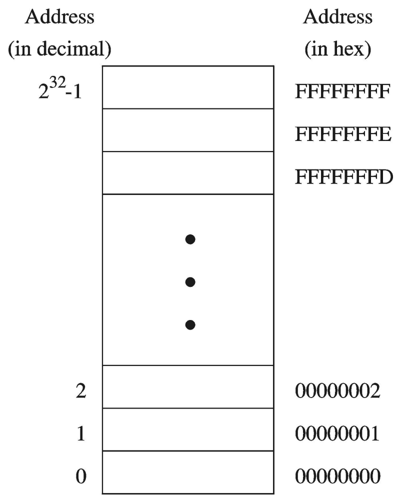
2.5 Exercises

1. Write the minimum program and execute.
2. Explain the contaent of the minimum program.
3. What is compiling?
4. What is linking?
5. What executable file contains?
6. What list file *lst.l* contains?
7. What is the format of an assembly instruction?
8. How execution occcurs?
9. What is the Operating System's role in the program execution?
10. What are the alternations required, when executable file is loaded in to memory for execution (by the Operating System)?

3 Software Architecture of IA-32

3.1 Memory

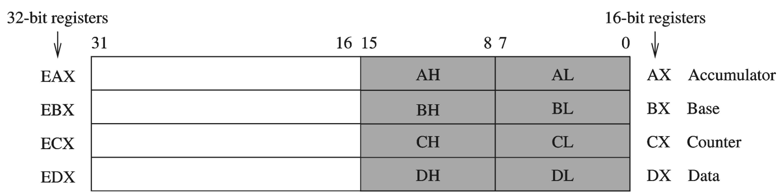
- Organized as a big array of bytes
- Addressable unit is byte
- Values larger than a byte are stored using *little-endian* storage model.
  i.e. least significant byte to most significant byte in lower address to higher address.
- Address is a unique number to distinguish bytes in read-/write operations
- GNU/Linux memory space 2^{32} Bytes i.e. 4GB



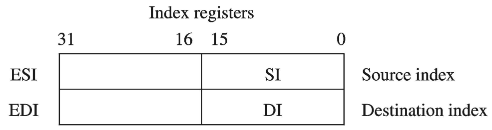
3.2 CPU Registers

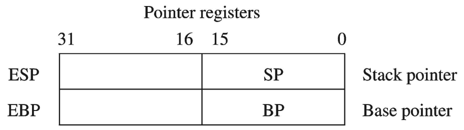
- IA-32 provides 8 general purpose 32-bit registers.
  - Group A: eax, ebx, cex, edx
    Group B: esi, edi, esp, ebp
- Segment Registers
  - cs, ss, ds, fs, gs
- Floating Point Registers: st0, st1, ..st7
- 64-bit MMX registers: mm0, mm1, ..mm7
- Control registers: cr0, cr1, ..cr4
- Debug registers: dr0, dr1, ..dr7
- Test registers: tr3, dr4, ..dr7
- Flags register EFLAGS
- Instruction Pointer EIP

3.2.1 General Purpose Registers

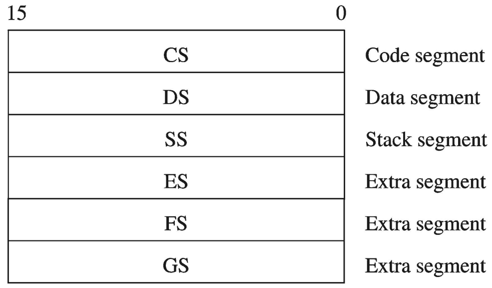


3.2.2 Index and Pointer Registers

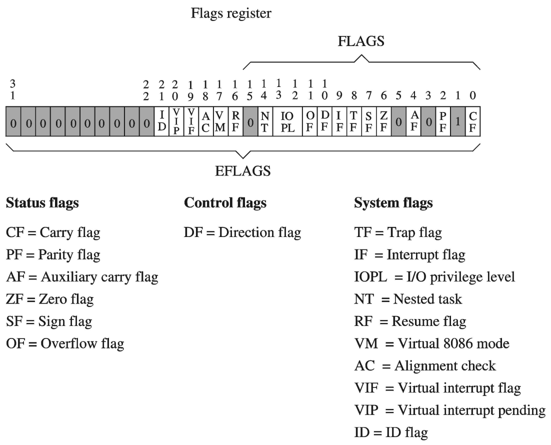


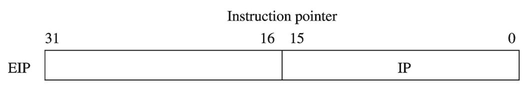


3.2.3 Segment Registers



3.2.4 Flags Registers





3.3 Exercises

1. What does CPU stand for?
2. What is the meaning of *Software Architecture* of a CPU?
3. What is the purpose of memory?
4. Differentiate between *little-endian* and *big-endian* storagte models.
5. What is address of memory cells?
6. What is the purpose of memory addresses?
7. What is the main purpose of CPU registers?
8. What is the different between special purpose and general purpose registers?
9. What is the purpose of FLAGS register?

4 Memory Allocation

4.1 Initialized Data (under .data)

`[var-name]  D?  init-value [,init-value],...`

db	Define Byte	allocates 1 byte
dw	Define Word	allocates 2 bytes
dd	Define Doubleword	allocates 4 bytes
dq	Define Quadword	allocates 8 bytes
dt	Define Ten bytes	allocates 10 bytes

4.2 Uninitialized Data (under .bss)

[var–**name**] RES? No.of Units

resb	Reserve a Byte	allocates 1 byte
resw	Reserve a Word	allocates 2 bytes
resd	Reserve a Doubleword	allocates 4 bytes
resq	Reserve a Quadword	allocates 8 bytes
rest	Reserve a Ten bytes	allocates 10 bytes

Examples:

```
response   resb   1
buffer     resw   100
Total      resd   1
```

4.3 Multiple definitions can be abbreviated

```
message db  'B'
        db  'y'
        db  'e'
        db  0DH
        db  0AH
message  db   'B','y','e',0DH,0AH
message  db   'Bye',0DH,0AH
```

```
marks  dw   0,0,0,0,0,0,0,0
marks  dw   8  dup(0)
marks  times  8  dw  0
```

4.4 Symbol Table

Assembler builds a symbol table so we can refer to the allocated storage space by the associated label.

.DATA

```
value     dw  0
sum       dd  0
marks     dw  10 dup (?)
message   db  'The grade is:',0
char1     db  ?
```

Symbol Table

name	offset
value	0
sum	2
marks	6
message	26
char1	40

5 Addressing Modes

Operands required by an operation can be specified in a variety of ways.

- Immediate
- Register
- Memory
  - Direct
  - Indirect
    - Register Indirect $[Base]$
    - Based $[Base + Disp]$
    - Indexed $[(Index * Scale) + Disp]$
    - Based-Indexed
      - Based-Indexed with no scale factor $[Base + Index + Disp]$
      - Based-Indexed with scale factor $[Base - Index * Scale + Disp]$

5.1 Immediate

operand is in the instruction itself (immediate addressing mode)

```
mov     al, 75       ; decimal
mov     bx, 0x25AC  ; hex
mov     ebx, 0b3C466Fh    ; hex (Why 0?)
mov     eax, 33C466Fh    ; hex
mov     cl, 11011011b    ; binary
```

5.2 Register

operand is in a register (register addressing mode)

```
mov     eax,ebx
mov     bx,cx
```

5.3 Memory

Operand is in the memory. Variety of addressing modes are used.
For 32-bit addressing

Segment	+ Base	+ (Index	* Scale)	+ Displacement
CS	EAX	EAX	1	0
SS	EBX	EBX	2	8-bit
DS	ECX	ECX	4	32-bit
ES	EDX	EDX	8	
FS	ESI	ESI		
GS	EDI	EDI		
FS	EDI	EDI		
	EBP	EBP		
	ESP			

For 16-bit addressing

Segment	+ Base	+ Index	+ Displacement
CS	BX	SI	0
SS	BP	DI	8-bit
DS			16-bit
ES			

5.3.1 Direct

```
mov     EAX,[response]
mov     [table1],56
```

5.3.2 Register Indirect

The offset address is specified indirectly via a register.

```
mov     EAX, [EBX]
```

Square brackets [ ] are used to indicate that EBX is holding an offset value. EBX contains a pointer to the operand, not the operand itself.

5.3.3 Based

The offset address is specified as $[Base + Displacement]$.

```
db   myname  80
..
mov     ECX, 0
ShowChar:
mov     EAX, [myname+ECX]
..
inc     ECX
cmp     ECX, 80
jne     ShowChar
```

- Access arrays having element size of 1 byte
  - Displacement ⇒ beginning of the array
  - Base register ⇒ relative offset of an element within the array
- Access fields of a structure
  - Base register ⇒ base address of the structure
  - Displacement ⇒ relative offset within the structure

5.3.4 Indexed

The offset address is specified as $[(Index * Scale) + Displacement]$.

```
dw marks   100
..
mov     ESI, 0
mov     EAX, 0
Total:
mov     EAX, [marks+ESI*2]
add     EAX, EAX
inc     ESI
cmp     ECX, 100
jne       Total
```

- Access elements of an array (particularly if the element size is 2, 4, or 8 bytes)
  - Displacement *Rightarrow* points to the beginning of the array
  - Index register *Rightarrow* selects an element of the array (array index)
  - Scaling factor *Rightarrow* size of the array element

5.3.5 Based Indexed

The offset address is specified as $[Base + Index + Displacement]$.

```
mov    EBX, table1
mov    EAX, [EBX+ESI]
cmp    EAX, [EBX+ESI+4]
```

- Useful in accessing two-dimensional arrays
  Displacement *Rightarrow* beginning of the array, Base and index registers *Rightarrow* row and an element within that row
- Useful in accessing arrays of records
  Displacement *Rightarrow* represents the offset of a field in a record, Base and index registers hold a pointer to the base of the array and the offset of an element relative to the base of the array
- Useful in accessing arrays passed on to a procedure
  Base register *Rightarrow* points to the beginning of the array Index register *Rightarrow* represents the offset of an element relative to the base of the array

5.3.6 Based Indexed with Scale factor

The offset address is specified as $[Base + (Index * Scale) + Displacement]$.

```
mov    EBX, table1
mov    EAX, [EBX+ESI]
cmp    EAX, [EBX+ESI+4]
```

Useful in accessing two-dimensional arrays when the element size is 2, 4, or 8 bytes

- Displacement *Rightarrow* points to the beginning of the array
- Base register *Rightarrow* holds offset to a row (relative to start of array)
- Index register *Rightarrow* selects an element of the row
- Scaling factor *Rightarrow* size of the array elements

5.4 Loading offset value into a register

E.g. Loading EBX with the offset value of table1

- in assembly time
  mov **EBX**,table1
- in run time
  lea **EBX**, [table1]
  lea **EBX**,[table1+**ESI**] (load EBX with the address of an element of table1 whose index is in the ESI register. We cannot use the mov instruction to do this.)

6 Data Transfer Instructions

- **mov** - Move (Actually copy)
- **xchg** - Exchange (Exchanges two operands)
- xlat - Translate (Translates byte values using a translation table)
- **push** - Push operand on Stack
- **pop** - Pop operand from stack top to a register
- **les** - Load ES and one of the registers from memory

6.1 The mov instruction

```
mov    destination,source
```

- Copies the value from source to destination
- Source is not altered as a result of copying
- Both operands should be of same size
- Source and destination cannot both be in memory

Allowed operand combinations

Instruction Type	Example
mov register , register	**mov** **DX**, **CX**
mov register , immidiate	**mov** **BL**, 100
mov register , memory	**mov** **EBX**, [count]
mov memory, register	**mov** [count], **ESI**
mov memory, immidiate	**mov** [count], 23

6.2 Ambiguous moves: PTR directive

E.g **mov**, [**EBX**],100
Not clear whether the assembler should use byte or word equivalent of 100. Then following type specifiers

Type Specifier	Bytes addressed
BYTE	1
WORD	2
DWORD	4
TWORD	8
QWORD	10

can be appropriately used to clarify as
mov, **WORD** [**EBX**],100
mov, **BYTE** [**EBX**],100

7 Arithmetic Instructions

- **add** dest, src - dest=dest+src
- **add** dest, src - dest=dest+src with Carry
- **sub** dest, src - dest=dest-src
- **sbb** dest, src - dest=dest-src with Borrow
- **mul** src - multiply
  ax=al×src, dx:ax=ax×src, edx:eax=eax×src,
- **imul** src - multiply signed numbers
  ax=al×src, dx:ax=ax×src, edx:eax=eax×src,

- **imul** dest, src - dest=dest*scr
- **imul** dest, src1, src2 - dest=src1*scr2
- **div** src - division
  remainder:quotient=ah:al=ax÷src, dx:ax=dx:ax ÷ src, edx:eax=edx:eax ÷ src,
- **cmp** op1, op2 - compare operand FLAGS=op1-op2
- **inc** op - increment by 1, op++
- **dec** op - decrement by 1, op–

8 Control Instructions

jmp - Unconditional Jump
jmpf - Unconditional Far Jump

Conditional Jump

- **jz** - jump if Zero Flag is set
- **jnz** - jump if Zero Flag is not set
- **jc** - jump if Carry Flag is set
- **jnc** - jump if Carry Flag is not set
- **jp** - jump if Parity Flag is set
- **jnp** - jump if Parity Flag is not set
- **jo** - jump if Overflow Flag is set
- **jno** - jump if Overflow Flag is not set

Conditional jump for unsigned numbers

- **je** - jump if op1==op2
- **jne** - jump if op1! =op2
- **ja** - jump if op1>op2 (above)
- **jna** - jump if op1<=op2
- **jb** - jump if op1<op2 (below)
- **jnb** - jump if op1>=op2

Conditional jump for signed numbers

- **je** - jump if op1==op2
- **jne** - jump if op1! =op2
- **jg** - jump if op1>op2 (greater)
- **jng** - jump if op1<=op2
- **jl** - jump if op1<op2 (lesser)
- **jnl** - jump if op1>=op2
- **loop** -

```
    mov ecx, 10
    mov eax, 0
addme:
    add eax, ecx
    loop addme ; ecx--, jump to addme if ecx!=0
```

9 Logic Instructions

- **and** op1, op2 - Bitwise logical And
- **or** op1, op2, **xor** op1, op2 - Bitwise logical Or, Xor
- **not** op1, Bitwise logical NOT of op1
- **test** op1, op2, Bitwise Logical AND, affects only FLAGS
- **shl** op1, op2, Bitwise shift left op1=op1«op2
- **shr** op1, op2, Bitwise shift left op1=op1»op2
- **rol** op1, op2, Bitwise cyclic left shift
- **ror** op1, op2, Bitwise cyclic right shift
- **rcl** op1, op2, Bitwise cyclic left shift through Carry bit
- **rcr** op1, op2, Bitwise cyclic right shift through Carry bit

10 Subroutines, call and ret

```
 ;main code ..
..
..
call func_name
..
..
;Subroutines
func_name:
  ..
  ..
  ret
```

11 Examples

1. No operations. Listing 1.
2. basics
   (a) Read a character from keyboard.
   (b) write a character to Console
   (c) Display content of a register in hexadecimal.
   (d) Write macros for above.
3. Model IOP
   (a) User can input a character through Keyboard. The character is stored in memory. Program add 1 to the character value and output on the console. Give your observations on the program output? what is the reason behind observed results? Listing 2
   (b) Modify the above programto change the lower-case input character to upper case and display. Pogram shuld be user friendly. Eg. Give information on the program at the begining, promt as *Enter lower case character*, etc. Listing 3
   (c) Add error handling to the abovep rogram. Eg. Error message if upper-case character or any non-alphebetical character is entered.
   (d) User can input a single digit (0-9)and program calculates the square of it and displays. Make the program user friendly and handle errors. Listing 4.
4. If
   (a) Check if the user input character is letter A. Listing 5
   (b) Check if the user input character is letter A or not. Listing 6
   (c) check if the user input number is greater than 87.
   (d) check if the user input character is letter, digit or non-alpha-numeric. Listing 7
5. For
   (a) Display the character * for 10 times. Listing 8
   (b) Display the character * for 10x15 times. listing 9
6. Integer Arithmetic
7. Procedures
8. IO
9.

Listing 1: Minimum Program (Do nothing!)

```
section .data ; initialized date
section .bss  ; uninitialized data
section .text ; code segment
global _start
_start:    ; program begining

nop      ; No operation

mov ebx,  0 ;End: exit code
mov eax,  1 ;sys. call no (sys_exit)
int 0x80    ;call kernel
```

Listing 2: Input Process and Output (single kbd character)

```
section .data
section .bss
  buf resb 1     ; reserves 1 bytes

section .text
global _start
_start:

;---------------------------------------------
; Read 1 byte and store in buf
mov ebx, 0     ; file to read from (0=stdin)
mov   ecx, buf  ; store read data at buf
mov edx, 1     ; length (read 1 bytes)
mov eax, 3     ; system call number "sys_read"
int    0x80    ; call kernel

; Process Data

mov al, Byte[buf]
add al, 1
mov byte[buf], al

; Display results on console

mov      ebx,1      ; write to file (1=stdout)
mov      edx,1      ; length (1 byte)
mov      ecx,buf    ; pointer to message to write
mov      eax,4      ; system call number (sys_write)
int      0x80     ; call kernel

;---------------------------------------------
;End the progrma
mov ebx,0    ;first syscall argument: exit code
mov    eax,1      ;system call number (sys_exit)
int      0x80     ;call kernel
```

Listing 3: Change input lower case character to upper case

```
section .data    ;initialized data
  msg1 db 0xa, 0xd,'Enter lower case letter: '
  msg2 db 0xa, 0xd,'Upper case is: '
section .bss    ; uninitialized data
buf resb 2     ; reserves 2 bytes
section .text    ; code
global _start
_start:

;------------------------------------------------------

; Display results on console

mov      ebx,1      ; write to file (1=stdout)
mov      edx,27     ; length (27 bytes)
mov      ecx,msg1   ; pointer to message to write
mov      eax,4    ; system call number (sys_write)
int      0x80     ; call kernel

; Read 1 byte and store in buf
mov ebx, 0     ; file to read from (0=stdin)
mov   ecx, buf  ; store read data at buf
mov edx, 1     ; length (read 1 bytes)
mov eax, 3     ; system call number "sys_read"
int    0x80    ; call kernel
```

```
; Change to upper case

mov al, Byte[buf] ; al = character
sub al, 'a'-'A'   ;
mov Byte[buf], al ; Load al to buf

; Display results on console

mov     ebx,1     ; write to file (1=stdout)
mov     edx,1     ; length (2 byte)
mov     ecx,buf   ; pointer to message to write
mov     eax,4     ; system call number (sys_write)
int     0x80      ; call kernel

;----------------------------------------------------
;End the progrma
mov ebx,0    ;first syscall argument: exit code
mov   eax,1      ;system call number (sys_exit)
int     0x80     ;call kernel
```

Listing 4: Calculate the square of a user input single digit number

```
section .data   ;initialized data
  msg1 db 0xa, 0xd,'Enter number between 0-9 : '
  msg2 db 0xa, 0xd,'Square is: '
  newln db 0xa, 0xd
section .bss    ; uninitialized data
  buf resb 2    ; reserves 2 bytes
section .text   ; code
global _start
_start:

;----------------------------------------------------

; Display results on console

mov     ebx,1     ; write to file (1=stdout)
mov     edx,29    ; length (27 bytes)
mov     ecx,msg1  ; pointer to message to write
mov     eax,4    ; system call number (sys_write)
int     0x80      ; call kernel

; Read 1 byte and store in buf
mov ebx, 0     ; file to read from (0=stdin)
mov    ecx, buf  ; store read data at buf
mov edx, 1     ; length (read 1 bytes)
mov eax, 3     ; system call number "sys_read"
int    0x80     ; call kernel



; Calculate Square

mov al, Byte[buf] ; al = character
sub al, '0'    ; convert to digit
mul al        ; square
mov bl, 10     ; bl = 10
div bl        ; ah:al=al/bl
add al, '0'    ; al to ASCII
add ah, '0'    ; ah to ASCII
mov word[buf], ax ; Load ax to buf

; Display results on console

; Display the message 'Square is'
mov     ebx,1     ; write to file (1=stdout)
mov     edx,13    ; length (13 bytes)
mov     ecx,msg2  ; pointer to message to write
mov     eax,4    ; system call number (sys_write)
int     0x80     ; call kernel

; Display the square value
mov     ebx,1      ; write to file (1=stdout)
mov     edx,2      ; length (2 byte)
mov     ecx,buf    ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
int     0x80     ; call kernel

;  New line
mov     ebx,1      ; write to file (1=stdout)
mov     edx,2      ; length (12 bytes)
mov     ecx,newln ; pointer to message to write
mov     eax,4    ; system call number (sys_write)
int     0x80     ; call kernel

;----------------------------------------------------
;End the progrma
mov ebx,0    ;first syscall argument: exit code
mov   eax,1      ;system call number (sys_exit)
int     0x80     ;call kernel
```

Listing 5: Check if the user input character is A

```
section .data   ;initialized data
  msg1 db 0xa, 0xd,'Enter number a character : '
  msg2 db 0xa, 0xd,'Entered Character is A.    '
  msg3 db 0xa, 0xd,'Entered Character is not A.'
  newln db 0xa, 0xd
section .bss    ; uninitialized data
  buf resb 2  ; reserves 2 bytes
section .text   ; code
global _start
_start:

;----------------------------------------------------

; Display results on console

mov     ebx,1      ; write to file (1=stdout)
mov     edx,29     ; length in bytes)
mov     ecx,msg1   ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
int     0x80     ; call kernel

; Read 1 byte and store in buf
mov ebx, 0     ; file to read from (0=stdin)
mov    ecx, buf  ; store read data at buf
mov edx, 1     ; length (read 1 bytes)
mov eax, 3     ; system call number "sys_read"
int    0x80     ; call kernel


; Convert to Upper case

mov al, Byte[buf] ; al = character
cmp al, 'A'    ; Compare al - 'A'
jne END_PROG

; Display the message 'A'
mov     ebx,1      ; write to file (1=stdout)
mov     edx,29     ; length in bytes)
mov     ecx,msg2   ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
```

```
int     0x80     ; call kernel

END_PROG:
; New line
mov     ebx,1      ; write to file (1=stdout)
mov     edx,2      ; length (12 bytes)
mov     ecx,newln  ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
int     0x80       ; call kernel

;-----------------------------------------------------
;End the progrma
mov ebx,0   ;first syscall argument: exit code
mov   eax,1     ;system call number (sys_exit)
int     0x80     ;call kernel
```

Listing 6: Check if the user input character is A or not

```
section .data   ;initialized data
  msg1 db 0xa, 0xd,'Enter number a character : '
  msg2 db 0xa, 0xd,'Entered Character is A.    '
  msg3 db 0xa, 0xd,'Entered Character is not A.'
  newln db 0xa, 0xd
section .bss     ; uninitialized data
  buf resb 2   ; reserves 2 bytes
section .text    ; code
global _start
_start:

;-----------------------------------------------------

; Display results on console

mov     ebx,1      ; write to file (1=stdout)
mov     edx,29     ; length in bytes)
mov     ecx,msg1   ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
int     0x80       ; call kernel

; Read 1 byte and store in buf
mov ebx, 0     ; file to read from (0=stdin)
mov    ecx, buf  ; store read data at buf
mov edx, 1     ; length (read 1 bytes)
mov eax, 3     ; system call number "sys_read"
int    0x80     ; call kernel


; Calculate Square

mov al, Byte[buf] ; al = character
cmp al, 'A'    ; Compare al - 'A'
jne NOT_A

; Display the message 'A'
mov     ebx,1      ; write to file (1=stdout)
mov     edx,29     ; length in bytes)
mov     ecx,msg2   ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
int     0x80       ; call kernel
jmp END_PROG

NOT_A:
; Display the message 'NOT A'
mov     ebx,1      ; write to file (1=stdout)
mov     edx,29     ; length in bytes)
mov     ecx,msg3   ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
int     0x80       ; call kernel
jmp END_PROG


END_PROG:
; New line
mov     ebx,1      ; write to file (1=stdout)
mov     edx,2      ; length (12 bytes)
mov     ecx,newln  ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
int     0x80       ; call kernel

;-----------------------------------------------------
;End the progrma
mov ebx,0   ;first syscall argument: exit code
mov   eax,1     ;system call number (sys_exit)
int     0x80     ;call kernel
```

Listing 7: If-Else:Digit or a Character?

```
..
PutCh 20, msg1     ; Display "Enter Character : "
GetCh 1, buf      ; Read the character from kbd and store

mov al, Byte[buf] ; Check if the character is a digit
cmp  al, '0'     ; r=al-'0'
jl NOTDIGIT    ; if r<0, jump to NOTDIGIT
cmp al, '9'    ; r=al-'9'
jg NOTDIGIT    ; if r>'9'jump to NOTDIGIT

PutCh 7, msg2    ; Display "Digit"
jmp    ENDP     ; End the program

NOTDIGIT:
PutCh 11, msg3     ; Display "Not a digit"

ENDP:
PutCh 2, nln
```

Listing 8: Display character * for 10 times

```
section .data   ;initialized data
  disp_c  db '*'
  newln db 0xa, 0xd
section .bss     ; uninitialized data
section .text    ; code
global _start
_start:

;-----------------------------------------------------


mov cl, 0

LOOP_1:
push ecx

; Display *
mov     ebx,1      ; write to file (1=stdout)
mov     edx,1      ; length in bytes)
mov     ecx,disp_c  ; pointer to message to write
mov     eax,4      ; system call number (sys_write)
int     0x80       ; call kernel
pop ecx
```

```
inc cl
cmp cl, 10
jl LOOP_1
```

Listing 9: Display character * for 10x15 times

```
; ================================================
; Two LOOPs
; Display * for 10x15 times
; + Use of PUSHAD and POPAD
; PUSHAD pushes EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI
; ================================================
section .data    ;initialized data
  disp_c  db '*'
  newln db 0xa, 0xd
section .bss     ; uninitialized data

section .text    ; code
global _start
_start:

;------------------------------------------------
mov ch, 0
LOOP_OUTER:
  mov cl, 0
  ;----------------------------
  LOOP_INNER:
  push ecx
  ; Display *
  mov      ebx,1      ; write to file (1=stdout)
  mov      edx,1      ; length in bytes)
  mov      ecx,disp_c  ; pointer to message to write
  mov      eax,4      ; system call number (sys_write)
  int      0x80     ; call kernel
  pop ecx
  inc cl
  cmp cl, 10
  jl LOOP_INNER
  ;----------------------------
; New line
pushad
mov      ebx,1      ; write to file (1=stdout)
mov      edx,2      ; length (12 bytes)
mov      ecx,newln ; pointer to message to write
mov      eax,4      ; system call number (sys_write)
int      0x80     ; call kernel

popad

inc ch
cmp ch, 15
jl LOOP_OUTER
;------------------------------------------------

END_PROG:
; New line
mov      ebx,1      ; write to file (1=stdout)
mov      edx,2      ; length (12 bytes)
mov      ecx,newln ; pointer to message to write
mov      eax,4      ; system call number (sys_write)
int      0x80     ; call kernel

;--------------------------------------------------
;End the progrma
mov ebx,0   ;first syscall argument: exit code
mov    eax,1     ;system call number (sys_exit)
int      0x80     ;call kernel
```

Listing 10: Add two Integers

```
section .data
section .bss
c_mem resb 4   ; reserve 4 bytes
section .text
global _start
_start:

mov eax, 1234h  ; eax = 1234h
mov ebx, 5678h  ; ebx = 5678h
add eax, ebx   ; eax = eax + ebx
mov   [c_mem], eax   ; [c_mem] = eax

mov ebx,0    ;End: exit code
mov eax,1    ;sys. call no (sys_exit)
int 0x80     ;call kernel
```

Listing 11: Add two Integers again

```
section .data
section .bss
c_mem resb 4
section .text
global _start
_start:

mov eax, 100   ; eax = 100
mov ebx, -50   ; ebx = -50
add eax, ebx   ; eax = eax + ebx
mov   [c_mem], eax   ; [c_mem] = eax

mov ebx,0    ;End: exit code
mov eax,1    ;sys. call no (sys_exit)
int 0x80     ;call kernel
```

Listing 12: Integer Arithmetic

```
mov eax, 100
mov ebx, -50
sub eax, ebx
; -------------------------
mov eax, 100
mov ebx, 50
sub eax, ebx
; -------------------------
mov eax, 100
mov ebx, 500
sub eax, ebx
; -------------------------
mov eax, 0x87654321
mov ebx, 0xcdef9876
mov ecx, 0x00000ba9
mov edx, 0x000000ab
clc
add eax, ebx
adc eax, edx
; -------------------------
```

Listing 13: Write Character to Console

```
section .data    ; initialized data
  msg db "X"
..
mov      edx,1      ;message length
mov      ecx,msg    ;pointer to message to write
mov      ebx,1      ;file handle (stdout)
```

```
mov     eax,4     ;system call number (sys_write)
int     0x80     ;call kernel
..
```

Listing 14: Read from Keyboard

```
section .data    ; initialized data
  msg db "X"
..
; Read 1 byte and store in buf
mov ebx, 0     ; file to read from (0=stdin)
mov ecx, buf     ; store read data at buf
mov edx, 1     ; length (read 1 bytes)
mov eax, 3     ; system call number "sys_read"
int    0x80     ; call kernel
..
```

Listing 15: Hello World

```
section .data    ; initialized data
  msg db "Hello, World"
section .bss     ; uninitialized data
  ;c_mem resb 4 ; reserves 4 bytes

section .text    ; code
global _start
_start:

mov     edx,5      ;message length
mov     ecx,msg    ;pointer to message to write
mov     ebx,1      ;file handle (stdout)
mov     eax,4      ;system call number (sys_write)
int     0x80     ;call kernel

;End the progrma
mov ebx,0 ;first syscall argument: exit code
mov     eax,1    ;system call number (sys_exit)
int     0x80   ;call kernel
```

Listing 16: Swap

```
;swap eax and ebx
mov ecx, eax ; method 1
mov eax, ebx
mov ebx, ecx

push eax   ; method 2
push ebx   ; using stak
pop eax
pop ebx

;swap mem1 and mem2
mov eax, Word[mem1]  ;method 1
mov ebx, Word[mem1]
mov ecx, eax ;
mov eax, ebx
mov ebx, ecx

;swap mem1 and mem2
push Word[mem1]  ;method 2
push Word[mem1]
pop [mem1]
pop [mem2]
```

Listing 17: Subroutine
Parameter Passing via registers

```
; ==================================================
; Procedures + Passing Parameters:
; Register Method
; ==================================================
%include "mylib.mac"
section .data   ;initialized data
  ascii_code  db '0','1','2','3','4','5',..,'c','d','e',

section .bss    ; uninitialized data
  buf resb 8
  eaxbuf resb 4
.START
;---------------------------------------------
mov eax, 0x12345678 ; num1
mov ebx, 0x87654321 ; num2

call sum2ints   ; Call Procedure "sum2ints"
  ; 1) Processor pushes "return address" to stack
  ; 2) Processor loads address of "sum2ints" to EIP
mov eax, 0x12345678
call disp_reg_hex

PutCh 8, buf
;---------------------------------------------
.EXIT

; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Procedure: sum2ints
; Sum two integers
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sum2ints:   ; name of the procedure
  add eax,ebx ; procedure boady
  ret    ; 1) Processesor pops "return address"
        ;    from the stack, and
      ; 2) loads EIP with the "return address"
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Procedure: disp_reg_hex
; Display the content of register eax in hex
; To use this you must have defined
; the table (under .data)
;   ascii_code  db '0','1','2','3',..,'c','d','e','f'
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
disp_reg_hex:
  mov ebx, ascii_code
  mov [eaxbuf], eax
  mov edx, 0xf0000000
  mov cl, 28
  mov esi, 0

NEXTNIBBLE:
  and eax, edx
  shr eax, cl
  xlatb   ; trnaslate using ascii_code
  mov byte[buf+esi], al
  shr edx, 4
  sub cl, 4
  inc esi
  mov eax, [eaxbuf]
  cmp cl, 0
  jge NEXTNIBBLE

  ret
```

Listing 18: Subroutine-Parameter passing via stack

```
; =========================================
; Procedures + Passing Parameters:
```

```
; Stack Method
; =========================================
%include "mylib.mac"
section .data        ;initialized data
  ascii_code  db '0','1','2',..,'c','d','e','f'

section .bss         ; uninitialized data
; buf resb 8
; eaxbuf resb 4
.START
;-----------------------------------------

mov eax, 0x30313233

push eax     ; input parameter is pushed to stack
call disp_reg ; call the procedure

;-------------------------------------------
.EXIT

; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Procedure: disp_reg_hex
; Display the content of register eax in hexadecimal
; To use this you must have defined
; the table (under .data)
;   ascii_code  db '0','1','2',..,'c','d','e','f'
;(Stack is used for input/output parameter passing
; and local variables)
; 1. Input parameters must be pushed before
;    calling the procedure
; 2. Access input parameters with [ebp+8],..
; 3. Do NOT use ebp for any other purpose
; 4. Allocate space in bytes xx for local variables
;    by enter xx, 0
; 5. Clear stack (pushed input parameters) by yy bytes.
; 6. Usually yy=xx, unless returned paramters are
;    pushed into stack and cleared in the
;    main program (after using them)
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
disp_reg:
  enter 8, 0  ;four bytes  for local variables
              ;[ebp-4], [ebp-5], ..
  pushad


  mov ebx, ascii_code
  ;mov eax, [ebp+8] ; First input parameter
                    ; at [ebp+8]
   ;mov word[ebp-4], 0x44; First local variable

  mov edx, 0xf0000000
  mov cl, 28
  mov esi, ebp
  sub esi, 4

NEXTNIBBLE:
  mov eax, [ebp+8]
  and eax, edx
  shr eax, cl
  xlatb       ; trnaslate using ascii_code

  mov byte[esi], al
  PutCh 1,esi
  shr edx, 4
  sub cl, 4
  dec esi
  cmp cl, 0
  jge NEXTNIBBLE

   popad
  leave
  ret 8
```

Listing 19: LPT port

```
 ; ==============================================
; Printer Port LPT1: adr= 0x378
;
; ==============================================
section .data   ;initialized data

section .bss    ; uninitialized data
  r_buf resb 6
section .text   ; code
global _start
_start:

jmp MYPROG

delay:
  mov ebx, 0xfff
OUTLOOP: mov eax, 0xfffff
INLOOP:
  dec eax
  jnz INLOOP
  dec ebx
  jnz OUTLOOP
  ret

MYPROG:
;---------------------------------------------
; Permission for IO operations
; for 378h to 37Ah
;---------------------------------------------
  mov eax, 101     ; sytem call number (sys_ioperm)
  mov ebx, 0x378     ; start IO address range
  mov ecx, 3     ; number of 8-bit ports to be given per
  mov edx, 1     ; turn-on value (1-permit, 0 - do not p
  int 0x80

; ---------------------------------------------
; write to data port 378h
; ---------------------------------------------
  mov  al, 0xFF   ; Bit patern
  mov dx, 0x378
        out dx,al    ; Send AL to the Address 378h

  call delay
  call delay
  call delay
  call delay
  call delay
  call delay
  mov al, 0xFF     ; Bit patern
  mov dx, 0x378
        out dx,al    ; Send AL to the Address 378h

  call delay

  mov  al, 0x55   ; Bit patern
  mov dx, 0x378
        out dx,al    ; Send AL to the Address 378h

  call delay
```

```
 mov al, 0xaa     ; Bit patern
 mov dx, 0x378
       out dx,al    ; Send AL to the Address 378h

 call delay
 mov  al, 0x55    ; Bit patern
 mov dx, 0x378
       out dx,al    ; Send AL to the Address 378h

 call delay

 mov al, 0xaa     ; Bit patern
 mov dx, 0x378
       out dx,al    ; Send AL to the Address 378h

 call delay
 mov  al, 0x55    ; Bit patern
 mov dx, 0x378
       out dx,al    ; Send AL to the Address 378h

 call delay

 mov al, 0xaa     ; Bit patern
 mov dx, 0x378
       out dx,al    ; Send AL to the Address 378h

 call delay

; ----------------------------------------------
; wait untill user preses 'q'
CHECK_Q:

  mov eax, 3     ; system call "read"
  mov ebx, 0     ; read from stdin
  mov   ecx, r_buf  ; store read data at r_buf
  mov edx, 1     ; length (read 1 bytes)
  int      0x80     ; call kernel

  mov   al, [r_buf]
  cmp   al, 'q'
  jnz CHECK_Q


; ----------------------------------------------
; off the IO access permission

; mov eax, 101     ; sytem call number (sys_ioperm)
; mov ebx, 0x378     ; start IO address range
; mov ecx, 3     ; number of 8-bit ports to be given per
; mov edx, 0     ; turn-on value (1-permit, 0 - do not p
; int 0x80


; ----------------------------------------------
; End the progrma
  mov ebx,0 ;first syscall argument: exit code
  mov     eax,1   ;system call number (sys_exit)
  int      0x80  ;call kernel

; ----------------------------------------------
```