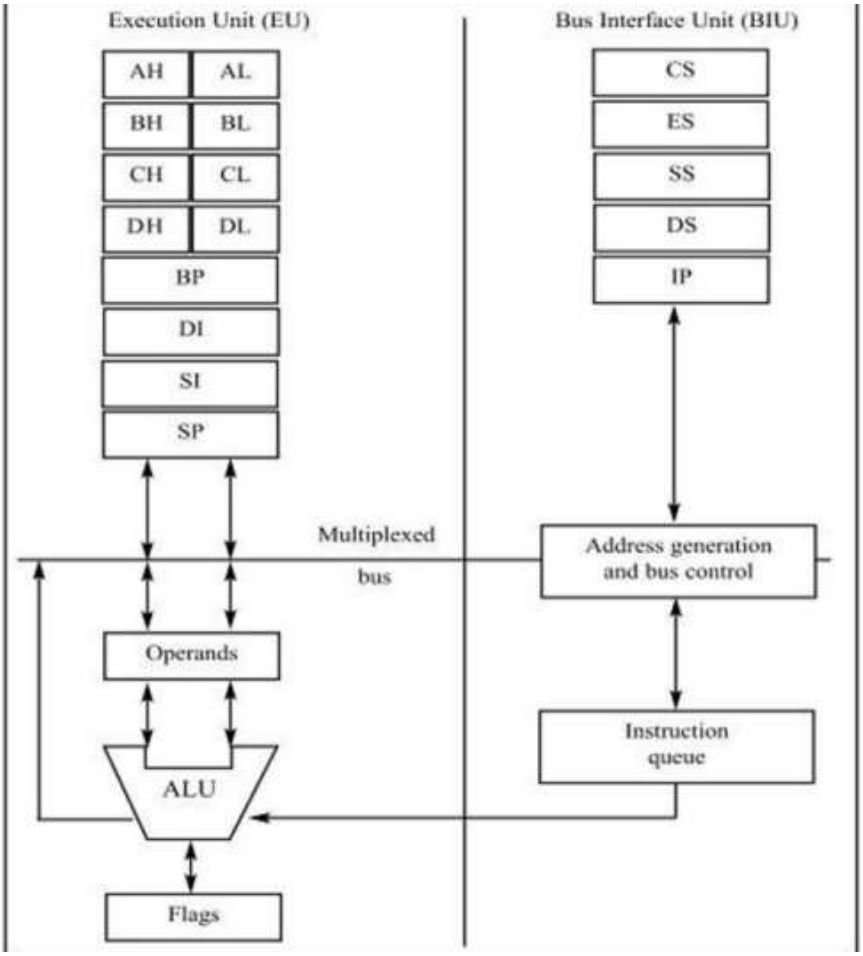
x8086 MP

THEORETICAL NOTES

- Internal architecture of a Microprocessor:

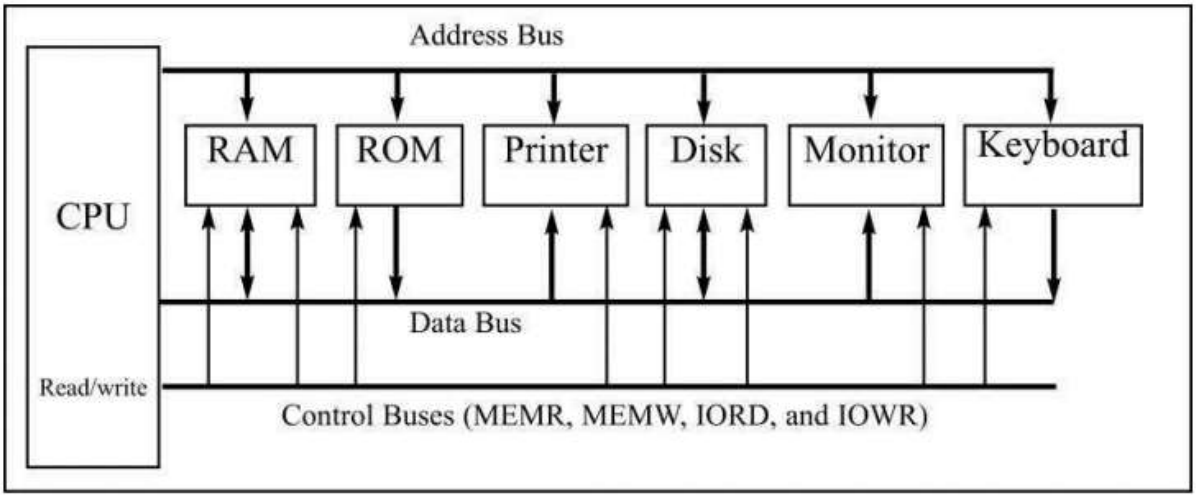


- Important Registers in the x8086

Registers

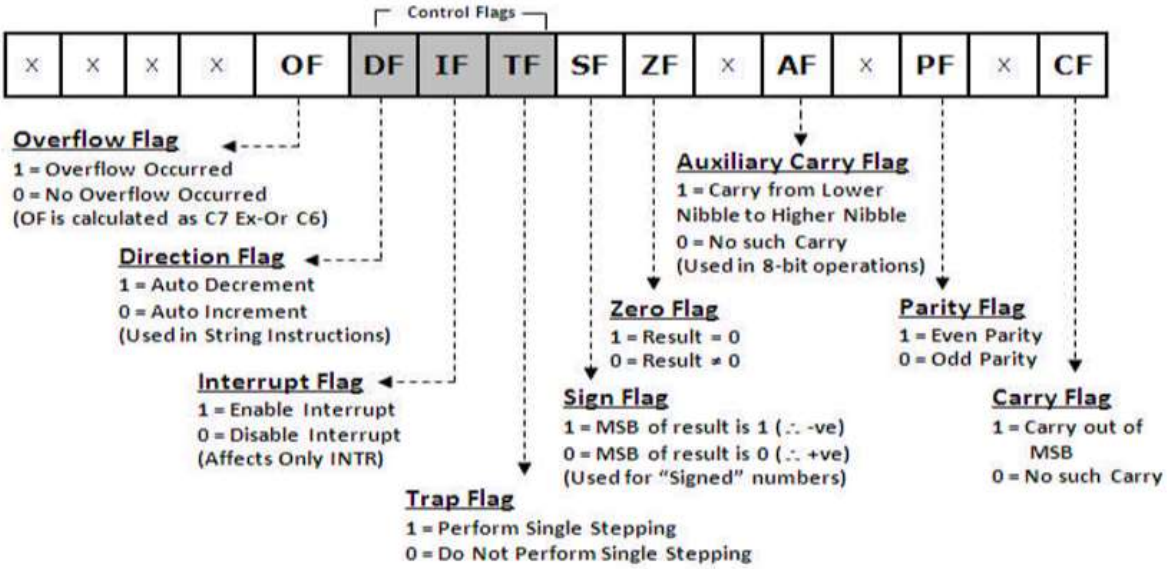
Category	Bits	Register Names
General	16	AX, BX, CX, DX
	8	AH, AL, BH, BL, CH, CL, DH, DL
Pointer	16	SP (stack pointer), BP (base pointer)
Index	16	SI (source index), DI (destination index)
Segment	16	CS (code segment), DS (data segment), SS (stack segment), ES (extra segment)
Instruction	16	IP (instruction pointer)
Flag	16	FR (flag register)

- Internal structure of a computer



- **Some Info about data busses :-**
  - The **more data buses** available, the **better the CPU.**
  - Average **bus size** is between **8** and **64**.
  - Computer **processing power** is related to **bus size.**
  - The **more address bits** available, the **larger the number of devices** that can be addressed.
  - The number of CPU address bits determines the **number of locations with which it can communicate. – Always equal to 2^x , where x is the number of address lines**, regardless of the size of the data bus.

- Flag Registers



- Powers of 2 and Conv. names of Memory

		Power of 2
Nipple	4-bits = ½ byte	
Byte	8 bits	$2^{10} = 1024 = 1K$
Word	16 bits = 2 bytes	$2^{20} = 1024K = 1M$
Double Word	32 bits = 4 bytes	$2^{30} = 1024M = 1G$
Quad Word	64 bits = 8 bytes	$2^{40} = 1024G = 1T$

- x8086 family differences

Table 1-1: Evolution of Intel's Microprocessors (from the 8008 to the 8088)

Product	8008	8080	8085	8086	8088
Year introduced	1972	1974	1976	1978	1979
Technology	PMOS	NMOS	NMOS	NMOS	NMOS
Number of pins	18	40	40	40	40
Number of transistors	3000	4500	6500	29,000	29,000
Number of instructions	66	111	113	133	133
Physical memory	16K	64K	64K	1M	1M
Virtual memory	None	None	None	None	None
Internal data bus	8	8	8	16	16
External data bus	8	8	8	16	8
Address bus	8	16	16	20	20
Data types	8	8	8	8/16	8/16

Table 1-2: Evolution of Intel's Microprocessors (from the 8086 to the Pentium Pro)

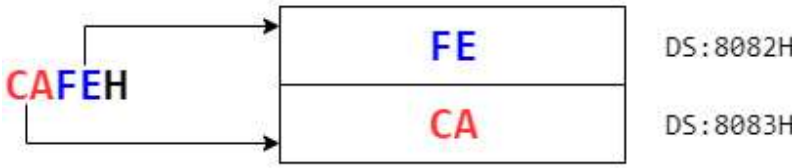
Product	8086	80286	80386	80486	Pentium	Pentium Pro
Year Introduced	1978	1982	1985	1989	1993	1995
Technology	NMOS	NMOS	CMOS	CMOS	BICMOS	BICMOS
Clock rate (MHz)	3–10	10–16	16–33	25–33	60, 66	150
Number of pins	40	68	132	168	273	387
Number of transistors	29,000	134,000	275,000	1.2 mill.	3.1 mill.	5.5 mill.
Physical memory	1M	16M	4G	4G	4G	64G
Virtual memory	None	1G	64T	64T	64T	64T
Internal data bus	16	16	32	32	32	32
External data bus	16	16	32	32	64	64
Address bus	20	24	32	32	32	36
Data types	8/16	8/16	8/16/32	8/16/32	8/16/32	8/16/32

Table 1-3: Evolution of Intel x86 Microprocessors: From Pentium II to Itanium

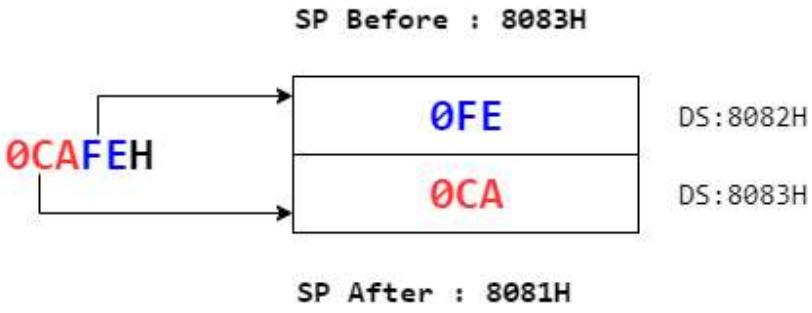
Product	Pentium II	Pentium III	Pentium 4	Itanium II
Year introduced	1997	1999	2000	2002
Technology	BICMOS	BICMOS	BICMOS	BICMOS
Number of transistors	7.5 mill.	9.5 mill.	42 mill.	220 mill.
Cache size	512K	512K	512K	3MB
Physical memory	64G	64G	64G	64G
Virtual memory	64T	64T	64T	64T
Internal data bus	32	32	32	64
External data bus	64	64	64	64
Address bus	36	36	36	64
Data types	8/16/32	8/16/32	8/16/32	8/16/32/64

- Addresses :
  - The physical address - the 20-bit address actually on the address pins
  - The logical address - segment value : offset address.
  - The offset address - a location in a 64K-byte segment range, which can range from 0000H to FFFFH.

- Methods of storing 16-bit data in 8-bit Memory :
  - **Little Endian (WE USE THIS IN x8086):** Lower bit takes the smaller address in memory.
    - Example (Data Segment) : Storing CAFEH



    - Example (Stack Segment) :



  - Big Endian : The opposite :)
- Segments :
  - A segment includes up to 64K bytes.
    begins on an address divisible by 16 (such an address ends in 0H)

Stack Name	Base Reg	Offset Reg
Code Segment	CS	IP
Data Segment	DS	not a specific one but you can use any of these (AX, BX, CX, DX, SI, DI)
Stack Segment	SS	SP, BP
Extra Segment	ES	Same as Data Segment

x8086 Assembly Programming Basic knowledge and syntax

- **For Fast (Binary ⇐ ⇒ Hex ⇐ ⇒ Decimal) Conversion:**

Decimal	Hex	Binary
0	00	0000
1	01	0001
2	02	0010
3	03	0011
4	04	0100
5	05	0101
6	06	0110
7	07	0111
8	08	1000
9	09	1001
10	0A	1010
11	0B	1011
12	0C	1100
13	0D	1101
14	0E	1110
15	0F	1111

- **Addressing Modes in general**

Addressing Mode	Operand	Default Segment
Register	reg	none
Immediate	data	none
Direct	[offset]	DS
Register indirect	[BX]	DS
	[SI]	DS
	[DI]	DS
Based relative	[BX]+disp	DS
	[BP]+disp	SS
Indexed relative	[DI]+disp	DS
	[SI]+disp	SS
Based indexed relative	[BX][SI]+disp	DS
	[BX][DI]+disp	DS
	[BP][SI]+disp	SS
	[BP][DI]+disp	SS

- **Program Parts:**

1. Program Model

Example :

```
.MODEL SMALL
    ;TINY   : data+code = 64KB
    ;SMALL  : data = 64KB and code = 64KB
    ;MEDIUM : data = 64KB but no code restrictions;
    ;COMPACT : code = 64KB but no data restriction
    ;LARGE  : Single set of data can not exceed 64KB
    ;HUGE   : No restriction
```

2. Defining Stack Size

Example :

```
.STACK 64   ;64 BYTES for stack
```

3. Defining variables (named memory chunks) and constants (like C++ #define)

```
                    .DATA                ; Defining DATA Segment
------------------------------------------------------------------------
; Different Sizes
BYTE_DATA           DB   52h             ; DB:Define Single Byte
WORD_DATA           DW   1234h           ; DW:Define word
DOUBLE_WORD_DATA    DD   12345678h       ; DD:Define Double
Quad_Word_DATA      DQ   12345678H       ; DQ:define quadword
Ten_Word_DATA       DT   1234567890H     ; DT:define TEN (10 Bytes)
------------------------------------------------------------------------
; Strings are just byte arrays
BYTE_STRING         DB   'I am Khaled'; Define string
YAString            DB   "With quotes"; Single or double quotes
------------------------------------------------------------------------
; ORG: Go to memory location
                     ORG 50h
------------------------------------------------------------------------
; Reserving Dummy Memory Locations
data2               DB   ?               ; Reserve Location
DUMMY               DB   6 DUP(0FFH)     ; Fill six bytes with FF
;Fill six bytes with FF ( 2 * 3 dummy 0FFH)
DUMMY2              DB   2 DUP(3 DUP (0FFH))
EMPTYW              DW   10 DUP(?)       ;Reserve 10 words
------------------------------------------------------------------------
; Defining Constants
Count               EQU 25               ;Define a constant
------------------------------------------------------------------------
; Array Examples
BYTE_DATA_MANY      DB   12h,15h,19h     ; multiple bytes
ArrW                 DW   1234h,5678h,9123h
```

4. Defining CODE Segment and MAIN PROC

```
                        .CODE           ; Defining CODE Segment where
                                          we write our code
-------------------------------------------------------------------------
MAIN        PROC FAR        ;Procedure length (default:NEAR)
            MOV AX,@DATA    ;WE Need to mov DS,@DATA to use
                              the variables we defined
            MOV DS,AX       ;But we can not do it in a single step
-------------------------------------------------------------------------
             ; Here you can add your main code
-------------------------------------------------------------------------
            MOV AH,4CH      ; This stops the program execution
            INT 21H
MAIN                ENDP
-------------------------------------------------------------------
; Here you can add any PROCS
-------------------------------------------------------------------
END MAIN        ; End of the program
```

5. JUMPS and Labels (Labels represent something like variables that carry code line address which you use to go to by JUMP instructions)

```
MyLabel:
    ADD AX,50
    JMP MyLabel
```

6. User Defined Procedures

a. Definition and Implementation

```
MYPROC    PROC
         ; Here goes your code
          RET
```

```
MYPROC    ENDP
```

b. Invoking a PROC (can be invoked in the MAIN, the PROC itself or any other PROC)

```
CALL MYPROC
```

c. Jumps and Procs Lengths :

`JMP SHORT MyLabel`	(Jumps only) Address of the target location is within -128 to + 127 bytes of memory relative to the address of the current IP

`JMP MyLabel`
`----------`
`MyProc PROC`
`CALL MyProc`
	(Default case - Near JMP) within the current code segment. Only IP is saved

`JMP FAR PTR MyLabel`
`----------`
`MyProc PROC FAR`
`CALL FAR PTR MyProc`
	(Far JMP) jump out of the current code segment. CS/IP are saved

7. User Defined Macros:

a. Macro defn. and impln. (parameters are optional):

```
MYMACRO MACRO P1, P2, P3
    ; Your code goes here
ENDM MYMACRO
```

b. Including a MACRO (if written in external `.inc` file in the same directory as the using file)

```
#INCLUDE MYMACROFILE.inc
```

c. Invoking a MACRO (can be invoked in MAIN PROC or any other PROC or even in a MACRO but **not inside itself**)

```
MAIN PROC FAR
    ....
    MYMACRO  14H, 15H, 16H
    ....
MAIN  ENDP
```

8. Multiple Files (Modular programming):

a. Exporting things out of a file

```
; In MyFile.asm
PUBLIC VALUE1, VALUE2, SUM, PRODUCT
.MODEL SMALL
.STACK 64
.DATA
        VALUE1 DB 2
        VALUE2 DB 3
        SUM DB ?
        PRODUCT DB ?
```

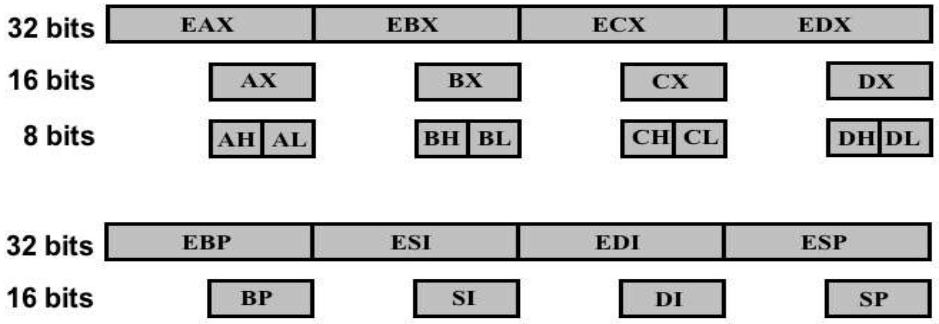
b. Importing in another file

```
; In Main.asm
        EXTRN VALUE1:BYTE
        EXTRN VALUE2:BYTE
        EXTRN SUM:BYTE
.MODEL SMALL
.CODE
...
```

- **32-bit Programming**
  - Registers

X86-32 register board:

Hybrid puzzle: type-1 and type-2.



  - Using in the code

    Write this in the top of the file

```
.386
```

    And use these regs and Double Words freely

- **Using assembly alongside C/C++**

```
#include "stdafx.h"
#include<iostream>
using namespace std;
int main()
{
    int16_t data1, data2, sum;
    data1 = 12;
    data2 = 24;
    _asm {
        mov ax, data1
            mov bx, data2
            add ax, bx      //Add the numbers
            mov sum, ax
    }
    cout << data1 << "+" << data2 << "=" << sum << "\n";
}
```

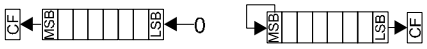
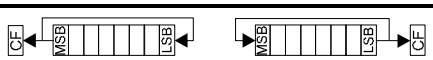
- **Proc. vs Macro**

Proc	**Macro**
Called every time	Doesn’t need to be called as code is repeated everywhere it’s written
Requires more time	Faster
Less Memory	More Memory
Preffered for long-repeated code	Preferred for short code written once but called many times

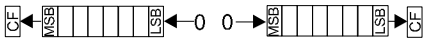
Instructions

TRANSFER				Flags								
Name	**Comment**	**Code**	**Operation**	**O**	**D**	**I**	**T**	**S**	**Z**	**A**	**P**	**C**
MOV	Move (copy)	MOV Dest,Source	Dest:=Source									
XCHG	Exchange	XCHG Op1,Op2	Op1:=Op2 , Op2:=Op1									
STC	Set Carry	STC	CF:=1									1
CLC	Clear Carry	CLC	CF:=0									0
CMC	Complement Carry	CMC	CF:= ⇓ CF									±
STD	Set Direction	STD	DF:=1 (string op's downwards)		1							
CLD	Clear Direction	CLD	DF:=0 (string op's upwards)		0							
STI	Set Interrupt	STI	IF:=1			1						
CLI	Clear Interrupt	CLI	IF:=0			0						
PUSH	Push onto stack	PUSH Source	DEC SP, [SP]:=Source									
PUSHF	Push flags	PUSHF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL									
PUSHA	Push all general registers	PUSHA	AX, CX, DX, BX, SP, BP, SI, DI									
POP	Pop from stack	POP Dest	Dest:=[SP], INC SP									
POPF	Pop flags	POPF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL	±	±	±	±	±	±	±	±	±
POPA	Pop all general registers	POPA	DI, SI, BP, SP, BX, DX, CX, AX									
CBW	Convert byte to word	CBW	AX:=AL (signed)									
CWD	Convert word to double	CWD	DX:AX:=AX (signed)	±				±	±	±	±	±
CWDE	Conv word extended double	CWDE 386	EAX:=AX (signed)									
IN *i*	Input	IN Dest, Port	AL/AX/EAX := byte/word/double of specified port									
OUT *i*	Output	OUT Port, Source	Byte/word/double of specified port := AL/AX/EAX									

i for more information see instruction specifications Flags: ±=affected by this instruction ?=undefined after this instruction

ARITHMETIC				Flags								
Name	**Comment**	**Code**	**Operation**	**O**	**D**	**I**	**T**	**S**	**Z**	**A**	**P**	**C**
ADD	Add	ADD Dest,Source	Dest:=Dest+Source	±				±	±	±	±	±
ADC	Add with Carry	ADC Dest,Source	Dest:=Dest+Source+CF	±				±	±	±	±	±
SUB	Subtract	SUB Dest,Source	Dest:=Dest-Source	±				±	±	±	±	±
SBB	Subtract with borrow	SBB Dest,Source	Dest:=Dest-(Source+CF)	±				±	±	±	±	±
DIV	Divide (unsigned)	DIV Op	Op=byte: AL:=AX / Op AH:=Rest	?				?	?	?	?	?
DIV	Divide (unsigned)	DIV Op	Op=word: AX:=DX:AX / Op DX:=Rest	?				?	?	?	?	?
DIV 386	Divide (unsigned)	DIV Op	Op=doublew.: EAX:=EDX:EAX / Op EDX:=Rest	?				?	?	?	?	?
IDIV	Signed Integer Divide	IDIV Op	Op=byte: AL:=AX / Op AH:=Rest	?				?	?	?	?	?
IDIV	Signed Integer Divide	IDIV Op	Op=word: AX:=DX:AX / Op DX:=Rest	?				?	?	?	?	?
IDIV 386	Signed Integer Divide	IDIV Op	Op=doublew.: EAX:=EDX:EAX / Op EDX:=Rest	?				?	?	?	?	?
MUL	Multiply (unsigned)	MUL Op	Op=byte: AX:=AL*Op if AH=0 ♦	±				?	?	?	?	±
MUL	Multiply (unsigned)	MUL Op	Op=word: DX:AX:=AX*Op if DX=0 ♦	±				?	?	?	?	±
MUL 386	Multiply (unsigned)	MUL Op	Op=double: EDX:EAX:=EAX*Op if EDX=0 ♦	±				?	?	?	?	±
IMUL *i*	Signed Integer Multiply	IMUL Op	Op=byte: AX:=AL*Op if AL sufficient ♦	±				?	?	?	?	±
IMUL	Signed Integer Multiply	IMUL Op	Op=word: DX:AX:=AX*Op if AX sufficient ♦	±				?	?	?	?	±
IMUL 386	Signed Integer Multiply	IMUL Op	Op=double: EDX:EAX:=EAX*Op if EAX sufficient ♦	±				?	?	?	?	±
INC	Increment	INC Op	Op:=Op+1 (Carry not affected !)	±				±	±	±	±	
DEC	Decrement	DEC Op	Op:=Op-1 (Carry not affected !)	±				±	±	±	±	
CMP	Compare	CMP Op1,Op2	Op1-Op2	±				±	±	±	±	±
SAL	Shift arithmetic left (≡ SHL)	SAL Op,Quantity		*i*				±	±	?	±	±
SAR	Shift arithmetic right	SAR Op,Quantity		*i*				±	±	?	±	±
RCL	Rotate left through Carry	RCL Op,Quantity		*i*								±
RCR	Rotate right through Carry	RCR Op,Quantity		*i*								±
ROL	Rotate left	ROL Op,Quantity		*i*								±
ROR	Rotate right	ROR Op,Quantity		*i*								±

i for more information see instruction specifications ♦ then CF:=0, OF:=0 else CF:=1, OF:=1

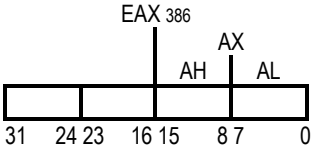
LOGIC				Flags								
Name	**Comment**	**Code**	**Operation**	**O**	**D**	**I**	**T**	**S**	**Z**	**A**	**P**	**C**
NEG	Negate (two-complement)	NEG Op	Op:=0-Op if Op=0 then CF:=0 else CF:=1	±				±	±	±	±	±
NOT	Invert each bit	NOT Op	Op:= ⇓ Op (invert each bit)									
AND	Logical and	AND Dest,Source	Dest:=Dest Ù Source	0				±	±	?	±	0
OR	Logical or	OR Dest,Source	Dest:=Dest Ú Source	0				±	±	?	±	0
XOR	Logical exclusive or	XOR Dest,Source	Dest:=Dest (exor) Source	0				±	±	?	±	0
SHL	Shift logical left (≡ SAL)	SHL Op,Quantity		*i*				±	±	?	±	±
SHR	Shift logical right	SHR Op,Quantity		*i*				±	±	?	±	±

MISC				Flags								
Name	**Comment**	**Code**	**Operation**	**O**	**D**	**I**	**T**	**S**	**Z**	**A**	**P**	**C**
NOP	No operation	NOP	No operation									
LEA	Load effective address	LEA Dest,Source	Dest := address of Source									
INT	Interrupt	INT Nr	interrupts current program, runs spec. int-program			0	0					

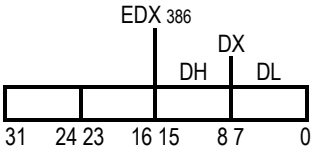
JUMPS (flags remain unchanged)							
Name	**Comment**	**Code**	**Operation**	**Name**	**Comment**	**Code**	**Operation**
CALL	Call subroutine	CALL Proc		RET	Return from subroutine	RET	
JMP	Jump	JMP Dest					
JE	Jump if Equal	JE Dest	(≡ JZ)	JNE	Jump if not Equal	JNE Dest	(≡ JNZ)
JZ	Jump if Zero	JZ Dest	(≡ JE)	JNZ	Jump if not Zero	JNZ Dest	(≡ JNE)
JCXZ	Jump if CX Zero	JCXZ Dest		JECXZ	Jump if ECX Zero	JECXZ Dest	386
JP	Jump if Parity (Parity Even)	JP Dest	(≡ JPE)	JNP	Jump if no Parity (Parity Odd)	JNP Dest	(≡ JPO)
JPE	Jump if Parity Even	JPE Dest	(≡ JP)	JPO	Jump if Parity Odd	JPO Dest	(≡ JNP)

JUMPS Unsigned (Cardinal)				JUMPS Signed (Integer)			
JA	Jump if Above	JA Dest	(≡ JNBE)	JG	Jump if Greater	JG Dest	(≡ JNLE)
JAE	Jump if Above or Equal	JAE Dest	(≡ JNB ≡ JNC)	JGE	Jump if Greater or Equal	JGE Dest	(≡ JNL)
JB	Jump if Below	JB Dest	(≡ JNAE ≡ JC)	JL	Jump if Less	JL Dest	(≡ JNGE)
JBE	Jump if Below or Equal	JBE Dest	(≡ JNA)	JLE	Jump if Less or Equal	JLE Dest	(≡ JNG)
JNA	Jump if not Above	JNA Dest	(≡ JBE)	JNG	Jump if not Greater	JNG Dest	(≡ JLE)
JNAE	Jump if not Above or Equal	JNAE Dest	(≡ JB ≡ JC)	JNGE	Jump if not Greater or Equal	JNGE Dest	(≡ JL)
JNB	Jump if not Below	JNB Dest	(≡ JAE ≡ JNC)	JNL	Jump if not Less	JNL Dest	(≡ JGE)
JNBE	Jump if not Below or Equal	JNBE Dest	(≡ JA)	JNLE	Jump if not Less or Equal	JNLE Dest	(≡ JG)
JC	Jump if Carry	JC Dest		JO	Jump if Overflow	JO Dest	
JNC	Jump if no Carry	JNC Dest		JNO	Jump if no Overflow	JNO Dest	
				JS	Jump if Sign (= negative)	JS Dest	
				JNS	Jump if no Sign (= positive)	JNS Dest	

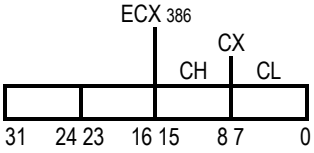
General Registers:



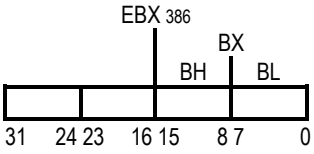
Accumulator



Data mul, div, IO



Count loop, shift



BaseX data ptr

Example:

```
                .DOSSEG             ; Demo program
                .MODEL SMALL
                .STACK 1024
Two             EQU 2               ; Const
                .DATA
VarB            DB ?                ; define Byte, any value
VarW            DW 1010b            ; define Word, binary
VarW2           DW 257              ; define Word, decimal
VarD            DD 0AFFFFh          ; define Doubleword, hex
S               DB "Hello !",0      ; define String
                .CODE
main:           MOV AX,DGROUP       ; resolved by linker
                MOV DS,AX           ; init datasegment reg
                MOV [VarB],42       ; init VarB
                MOV [VarD],-7       ; set VarD
                MOV BX,Offset[S]    ; addr of "H" of "Hello !"
                MOV AX,[VarW]       ; get value into accumulator
                ADD AX,[VarW2]      ; add VarW2 to AX
                MOV [VarW2],AX      ; store AX in VarW2
                MOV AX,4C00h        ; back to system
                INT 21h
                END main
```

Flags: | - | - | - | - | O | D | I | T | S | Z | - | A | - | P | - | C |

Control Flags (how instructions are carried out):

D: Direction	1 = string op's process down from high to low address
I: Interrupt	whether interrupts can occur. 1= enabled
T: Trap	single step for debugging

Status Flags (result of operations):

C: Carry	result of unsigned op. is too large or below zero. 1 = carry/borrow
O: Overflow	result of signed op. is too large or small. 1 = overflow/underflow
S: Sign	sign of result. Reasonable for Integer only. 1 = neg. / 0 = pos.
Z: Zero	result of operation is zero. 1 = zero
A: Aux. carry	similar to Carry but restricted to the low nibble only
P: Parity	1 = result has even number of set bits

SIGNED NUMBERS

⇒ BYTE range ⇒ -128:127 or -80:7F

⇒ WORD range ⇒ -32768:32767 or -8000:7FFF

⇒ SIGNED ⇒ OF, UNSIGNED ⇒ CF

CBW	Extend AL to be AX	AH:=00H or AH:=FFH According to the sign of AL	mov al,0FFH CBW
CWD	Extend AX to be DXAX	DX:=0000H or DX:=FFFFH According to the sign of AX	

IMUL Op (B*B)	Multiplies a byte (Op) by AL and stores in AX	AX:= AL*Op	mov al,7h
			mov bl,10010100B
			IMUL bl

IMUL Op (W*W)	Multiplies a word (Op) by AX and stores in DXAX	DXAX := AX*Op	mov BX,112AH IMUL BX
IMUL Op (W*B)	Extends AL then Multiplies a word by AX and stores in DX	AH:=00F or AH:=FFH DXAX := AX*Op	mov al,8h; CBW IMUL BX

IDIV Op (W/B)	Divides AX by a byte Op, stores quotient in AL and rem. in AH	AL:= AX/Op
		AH:=AX%Op
			mov ax,0FFFH
			mov bl,0FH
			IDIV bl

IDIV Op (B/B)	Extend AL then does the one before	AH:=00H or AH:=FFH
		AL:= AX/Op
		AH:=AX%Op
			mov al,0FFH
			mov bl,00FH
			CBW
			IDIV bl

IDIV Op (W/W)	Extend AX to DXAX, Divides DXAX by Op and stores quot in AX and rem in DX	DX:=0000H or DX:=FFFFH
		AX:=DXAX/Op
		DX:=DXAX%Op
			mov ax,1002H
			mov bx, 0FFFH
			CWD
			IDIV bx

SAR Reg,Op	Shifts a signed number Op times right filling with the sign bit. Every time	CF:= Reg[0] every time	mov al,10110011B SAR al,1 ; al becomes 11011001B and CF

	the LSB is stored in CF		becomes 1
SAL	Same as SHL	-	-
JG, JGE, JL, JLE, JE	Jumps for signed operations. G for greater, L for less, E for equal	-	cmp ax,bx JG LBL1
JA, JAE, JB, JBE, JE	Jumps for unsigned operations. A for above, B for below and E for equal		cmp ax,bx JA LBL1

STRING OPERATIONS :

⇒ We should set ES to DS before using these instruction like this :-

```
MOV AX,@DATA
MOV DS,AX
MOV ES,AX
```

⇒ Almost everything here works with **SI,DI** where SI lives in DS and DI lives in ES

⇒ DF or Direction Flag determines the direction of string operations, 0 means auto-increment and 1 means auto-decrement

CMPSB	Compares the byte at address DS:SI with the byte at the address ES:DI	Changes the flags	mov SI, offset str1 mov DI, offset str2 CMPSB JNE LBL2 LBL1: INC SI INC DI CMPSB JE LBL1 LBL2:....
CMPSW	Compares the word starting at address DS:SI with the word starting at the address ES:DI	Changes the flags	mov SI, offset str1 mov DI, offset str2 CMPSW JNE LBL2 LBL1: INC SI INC DI CMPSW JE LBL1 LBL2:....

STD	Sets the direction flag to auto-decrement	DF:=1	-
CLD	Clears the dir. Flag to auto-increment	DF:=0	-
REPE Comp_Operation	Repeats a comparison operation for number of times stored in CX as long as comparison operation returns equal	CX should contain no. of times Increments SI and DI everytime	mov cx,20h REPE CMPSB
REPNE	Same as above as long as **not equal**	CX should contain no. of times Increments SI and DI everytime	mov cx,20h REPNE CMPSB
REP	Repeats a comparison operation for number of times stored in CX **unconditionally**	CX should contain no. of times Increments SI and DI everytime	mov cx,20h REP CMPSB
MOVSB	Copies the byte at [DS:SI] to [ES:DI]	[ES:DI] = [DS:SI]	mov SI,offset str1 mov DI, offset str2 Mov cx,20d REP MOVSB
MOVSW	Copies the word at [DS:SI] to [ES:DI]	[ES:DI] = [DS:SI] [ES:DI+1] = [DS:SI+1]	mov SI,offset str1 mov DI, offset str2 Mov cx,10d REP MOVSW
LODSB	Loads the byte at [DS:SI] to AL **CAUTION:** **REP LODSB is not allowed !**	AL=[DS:SI]	-
LODSW	Loads the word starting at [DS:SI] to AX **CAUTION:** **REP LODSW is not**	AL=[DS:SI] AH=[DS:SI+1]	

	allowed !		
STOSB	Stores AL in [ES:DI]	[ES:DI] = AL	
STOSW	Stores AX in [ES:DI] and [ES:DI+1]	[ES:DI] = AL [ES:DI+1] = AH	
INSB	Loads a byte from I/O address stored in DX and Stores it in ES address stored in DI then increments DI by 1	[ES:DI] = IN(DX) DI += 1	mov dx,34f8h INSB
INSW	Loads a byte from I/O address stored in DX and Stores it in ES address stored in DI then increments DI by 1	[ES:DI] = IN(DX) DI += 2	mov dx,34f8h INSW
OUTSB	Loads a byte from DS address stored in SI to I/O address stored in DX then increments SI by 1	OUT(DX) = [DS:SI] SI += 1	mov dx,34f8h OUTSB
OUTSW	Loads a word from DS address stored in SI to I/O address stored in DX then increments SI by 2	OUT(DX) = [DS:SI] SI += 2	mov dx,34f8h OUTSW

SCASB	Compares AL with [ES:DI]		mov al,41H
			mov cx,20 ;Search the first 20 bytes of DI for 'A' char
			REPNE SCASB

XLAT	When BX carries the offset of an ASCII Lookup table of hex numbers from 0 to F, then AX is translates to its corresponding ASCII	BX:= offset ASCII_TABLE AX:= BX[AX]	mov al,9 mov BX,offset ASC_TBL XLAT

Some notes on Instructions:

- DIV doesn't allow immediate mode (DIV 2)
- MUL doesn't allow immediate mode (MUL 10)
- MUL & DIV OPERANDS CAN BE A REGISTER OR MEMORY
- IN DIV & MUL IN [REGISTER INDIRECT MODE]--> OPERATION TYPE SHOULD BE SPECIFIED USING BYTE PTR OR WORD PTR
- MUL BYTE PTR[SI] --> MULTIPLY AL BY A BYTE FROM THE OFFSET POINTED TO BY SI
- SHL, SHR, SAL, SAR, ROL, ROR, RLC, RCR --> EITHER BY 1 OR QUANTITY SHOULD BE SPECIFIED BY CL
- MOV IMMEDIATE VALUE INTO SEGMENT REGISTERS [CS,DS,SS,ES] IS NOT ALLOWED
- MOV INTO FLAG REGISTER IS NOT ALLOWED
- MOV REGISTER SOURCE AND DESTINATION MUST BE OF SAME SIZE MOV AX, BX
- CS CANNOT BE MODIFIED DIRECTLY [MOV CS, AX FOR EXAMPLE IS NOT ALLOWED]
- MOV AX, [BX] + 3 = MOV AX, [BX + 3] = MOV AX, 3[BX]
- PUSH AL,AH,BL,BH,CL,CH,DL,DH IS NOT ALLOWED
- Division by 0 gives **Runtime Error.**
- **Carry & Overflow flags:**
    - Carry : look to the **unsigned** form of the operands/result.
        - Addition(ADD Op1,Op2): CF=1 when Op1+Op2 > 0FFFFH for 16-bit or 0FFH for 8-bit.
        - Subtraction(SUB Op1,Op2): CF=1 when Op1 < Op2
    - Overflow : look to the **signed** form of the operands/result
        - Addition (ADD Op1,Op2): OF=1 when Op1&Op2 have same sign opposite to the result sign
        - Subtraction (SUB Op1,Op2): Op1(+ve), Op2(-ve) but result(-ve) or Op1(-ve), Op2(+ve) but result is (+ve).

Interrupts and their appendices

INT#	option	Function	Input/Output	Example

10H Video modes	AH=00	Change video mode (modes are listed in page 26)	**Input:**
			AL = Video Mode No. (Look Page 26)
			Output:
			--
				`MOV AH,0`
				`MOV AL,13H ; This is 320*200 VGA Gfx Mode`
				`INT 10H`

	AH=02	Move *Cursor* to X,Y Position	**Input:**
			DL = X, DH = Y
			Output:
			--
				`MOV AH,2`
				`MOV DX,0B0AH ; X = 0AH, Y = 0BH`
				`INT 10H`

	AH=03	Get *Cursor* Position	**Input:**
			BH = Page no. (must be 0 in gfx modes)
			Output:
			DL = X, DH = Y
				`MOV AH,03`
				`MOV BH,0`
				`INT 10H`

	AH=06	Scroll-up (can be used for clearing the screen)	**Input:**
			AL = lines to scroll (0 in case of clearing)
			BH = Color attribute to color the scrolled part (look "color attributes" in page 27)
			CL,CH = X,Y of scroll starting point - (0,0) if top-left of the screen
			Dl, DH = X,Y of scroll ending point - (79,24) if bottom-rigth of a 80*25 screen
			Output:
			--
				`MOV AH,06H`
				`MOV AL,10 ;scroll up 10 lines`
				`MOV BH,07 ;Normal attr. (black back, white fore)`
				`MOV CX,0000 ;Starting at (0,0)`
				`MOV DX,184FH ;Ending at (79,24)`
				`INT 10H`

	AH=07	Scroll-down (can be used for clearing the screen)	**Input:**
			AL = lines to scroll (0 in case of clearing)
			BH = Color attribute to color the scrolled part (look "color attributes" in page 27)
			CL,CH = X,Y of scroll starting point - (0,0) if top-left of the screen
			Dl, DH = X,Y of scroll ending point - (79,24) if bottom-rigth of a 80*25 screen
			Output:
			--
				`MOV AH,07H`
				`MOV AL,10 ;scroll down 10 lines`
				`MOV BH,07 ;Normal attr. (black back, white fore)`
				`MOV CX,0000 ;Starting at (0,0)`
				`MOV DX,184FH ;Ending at (79,24)`
				`INT 10H`

	AH=08	Read char at *Cursor* position	**Input:**
			BH = Page No.
			Output:
			AL = char. ASCII (look ASCII codes in page 28)
			AH = char. color (look color attr. format in page 27)
				`MOV AH,08H`
				`MOV BH,00`
				`INT 10H`

	AH=09	Display a Char number of times in a certain color	**Input:**
			AL = Char. ASCII
			BH = Page No.
			BL = Color (look color attr. format in page 27)
			CX = No. of times
			Output:
			--
				`MOV AH,09H`
				`MOV AL,'A' ;Display char 'A'`
				`MOV BH,0 ;Page 0`
				`MOV BL,07 ;Normal color attr. (black back, white fore)`
				`MOV CX,010H ;16 Times`
				`INT 10H`

	AH=0CH	Draw Pixel in graphics mode given :
		1. No. of page
		2. Position
		3. Color
			Input:
			AL = color hex. (according to the graphics mode)
			BH = Page no. (should be 0)
			CX = X position of the pixel (col.)
			DX = Y position of the pixel (row)
			Output:
			--
				`MOV AH,0CH`
				`MOV AL,05 ;Pixel color`
				`MOV BH,0 ;Page 0`
				`MOV CX,0 ;X = 0`
				`MOV DX,10H ; Y = 16`
				`INT 10H`

	AH=0DH	Read color of a pixel in graphics mode	**Input:**
			BH = Page no.
			CX = X position of the pixel to read (col)
			DX = Y position of the pixel to read (row)
			Output:
			AL = Color
				`MOV AH,0DH`
				`MOV BH,0 ;Page 0`
				`MOV CX,0 ;X = 0`
				`MOV DX,10H ; Y = 16`
				`INT 10H`

	AH=0FH	Get Current mode data :- 1. Mode no. 2. No. of character coloumns 3. Active page no.	**Input: --** **Output:** AL = Video Mode no. (look p26) AH = no. of col.s BH = active page	`MOV AH,0FH` `INT 10H`

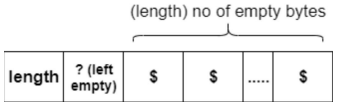
16H Keyboard	AH=0	Wait key pressed and return its scan and ASCII	**Input**: -- **Output:** AH = scan code (look page 29) AL = ASCII code (look page 28)	`MOV AH,00H` `INT 16H`

	AH=1	Check for a pressed key (no wait)	**Input : --**
			Output:
			ZF=1 if no key pressed
			AH,AL = scan,ASCII if key pressed
				`Check: MOV AH,1`
				`INT 16H`
				`JZ Check`

21H
String
	AH=2	Display Char	**Input:**
			DL = Char to print ('$' is printed also , 7 for beep sound)
			Output:--
				`MOV AH,02H`
				`MOV DL,'$'`
				`INT 21H`

	AH=7	Read ONE char from user without echo (printing in the console)	**Input:** AL = input char (ASCII) **Output**:--	`MOV AH,07H` `INT 21H`

	AH=9	Display string	**Input:**
			DX = offset (pointer) to the message in DS
			Output:--
				`Mes DB 'this is a message','$' ; should be ended w/ $`
				
				`MOV AH,09`
				`MOV DX,offset Mes`
				`INT 21H`

	AH=0AH	Read string from keyboard until the user presses ENTER	**Input:**
			DX = offset (pointer) to data buffer to read in
			 
			
			Output:
			Buffer[1] -> actual length
			Buffer[2] -> Actual Entred string
				`Buff DB '30','?', 30 DUP($)`
				`-- or--`
				`Buff LABEL BYTE ; pointer to the first byte of the buffer`
				`BuffMaxLen DB 30`
				`BuffActualLen DB ? ;carries the actual length after reading`
				`BuffData DB 30 DUP($) ;carries the actual entries`
				
				`MOV AH,0AH`
				`MOV DX,offset Buff`
				`INT 21H`

33H Mouse	AX=0	Check if there's a mouse attached	**Input** : -- **Output:** AX = 0FFFFH if mouse attached	`MOV AX,00H` `INT 33H`
	AX=1	Shows the mouse	**Input/Output:--**	`MOV AX,01H` `INT 33H`
	AX=2	Hides the mouse	**Input/Output:--**	`MOV AX,02H` `INT 33H`

	AX=3	Get mouse position and button status	**Input: --**
			Output:
			CX = Y of the mouse (row)
			DX = X of the mouse (col)
			BX = 1 if button clicked, 0 if not
				`MOV SI,200`
				`MOV AX,3`
				`MPos: INT 33H`
				`DEC SI`
				`JNZ MPos`

	AX=4	Set mouse position	**Input:**
			CX = Y of the mouse (row)
			DX = X of the mouse (col)
			Output:--
				`MOV AX,04H`
				`MOV CX,000AH`
				`MOV DX,000BH`
				`INT 33H`

Table E-2: Video Modes and Their Definition

AL	Pixels	Characters	Char box	Text/ graph	Colors	Aadpter	Max pages	Buffer start
00H	320x200	40 x 25	8x8	text	16 *	CGA	8	B8000h
	320x350	40 x 25	8x14	text	16 *	EGA	8	B8000h
	360x400	40 x 25	9x16	text	16 *	VGA	8	B8000h
	320x400	40 x 25	8x16	text	16 *	MCGA	8	B8000h
01H	320x200	40 x 25	8x8	text	16	CGA	8	B8000h
	320x350	40 x 25	8x14	text	16	EGA	8	B8000h
	360x400	40 x 25	9x16	text	16	VGA	8	B8000h
	320x400	40 x 25	8x16	text	16	MCGA	8	B8000h
02H	640x200	80 x 25	8x8	text	16 *	CGA	8	B8000h
	640x350	80 x 25	8x14	text	16 *	EGA	8	B8000h
	720x400	80 x 25	9x16	text	16 *	VGA	8	B8000h
	640x400	80 x 25	8x16	text	16 *	MCGA	8	B8000h
03H	640x200	80 x 25	8x8	text	16	CGA	8	B8000h
	640x350	80 x 25	8x14	text	16	EGA	8	B8000h
	720x400	80 x 25	9x16	text	16	VGA	8	B8000h
	640x400	80 x 25	8x16	text	16	MCGA	8	B8000h
04H	320x200	40 x 25	8x8	graph	4	CGA	1	B8000h
	320x200	40 x 25	8x8	graph	4	EGA	1	B8000h
	320x200	40 x 25	8x8	graph	4	VGA	1	B8000h
	320x200	40 x 25	8x8	graph	4	MCGA	1	B8000h
05H	320x200	40 x 25	8x8	graph	4 *	CGA	1	B8000h
	320x200	40 x 25	8x8	graph	4 *	EGA	1	B8000h
	320x200	40 x 25	8x8	graph	4 *	VGA	1	B8000h
	320x200	40 x 25	8x8	graph	4 *	MCGA	1	B8000h
06H	640x200	80 x 25	8x8	graph	2	CGA	1	B8000h
	640x200	80 x 25	8x8	graph	2	EGA	1	B8000h
	640x200	80 x 25	8x8	graph	2	VGA	1	B8000h
	640x200	80 x 25	8x8	graph	2	MCGA	1	B8000h
07H	720x350	80 x 25	9x14	text	mono	MDA	8	B0000h
	720x350	80 x 25	9x14	text	mono	EGA	4	B0000h
	720x400	80 x 25	9x16	text	mono	VGA	8	B0000h
08H	reserved							
09H	reserved							
0AH	reserved							
0BH	reserved							
0CH	reserved							
0DH	320x200	40 x 25	8x8	graph	16	EGA	2/4	A0000h
	320x200	40 x 25	8x8	graph	16	VGA	8	A0000h
0EH	640x200	80 x 25	8x8	graph	16	EGA	1/2	A0000h
	640x200	80 x 25	8x8	graph	16	VGA	4	A0000h
0FH	640x350	80 x 25	9x14	graph	mono	EGA	1	A0000h
	640x350	80 x 25	8x14	graph	mono	VGA	2	A0000h
10H	640x350	80 x 25	8x14	graph	4	EGA	1/2	A0000h
	640x350	80 x 25	8x14	graph	16	VGA	2	A0000h
11H	640x480	80 x 30	8x16	graph	2	VGA	1	A0000h
	640x480	80 x 30	8x16	graph	2	MCGA	1	A0000h
12H	640x480	80 x 30	8x16	graph	16	VGA	1	A0000h
13H	320x200	40 x 25	8x8	graph	256	VGA	1	A0000h
	320x200	40 x 25	8x8	graph	256	MCGA	1	A0000h

* color burst off

Color attribute appendix

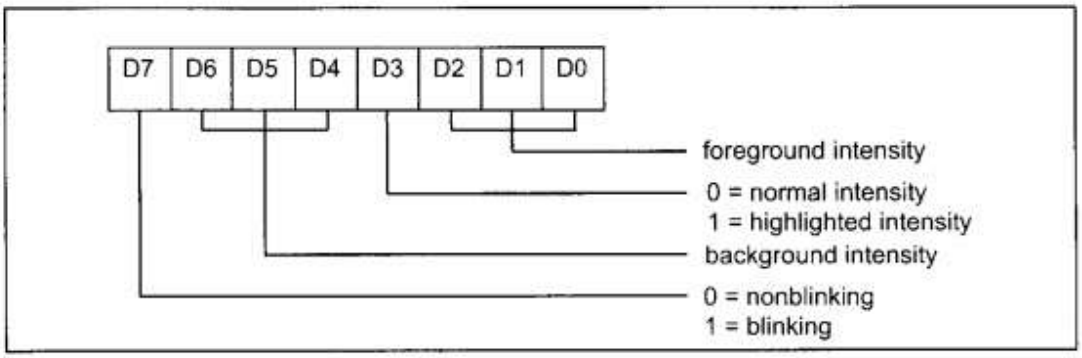


Figure 4-2. Attribute Byte for Monochrome Monitors

Table 4-1: The 16 Possible Colors

I	R	G	B	Color
0	0	0	0	black
0	0	0	1	blue
0	0	1	0	green
0	0	1	1	cyan
0	1	0	0	red
0	1	0	1	magenta
0	1	1	0	brown
0	1	1	1	white
1	0	0	0	gray
1	0	0	1	light blue
1	0	1	0	light green
1	0	1	1	light cyan
1	1	0	0	light red
1	1	0	1	light magenta
1	1	1	0	yellow
1	1	1	1	high intensity white

ASCII Cheatsheet

ASCII Control Characters

Dec	Hex	Char	Description
0	00	NUL	null
1	01	SOH	start of header
2	02	STX	start of text
3	03	ETX	end of text
4	04	EOT	end of transmission
5	05	ENQ	enquiry
6	06	ACK	acknowledge
7	07	BEL	bell
8	08	BS	backspace
9	09	HT	horizontal tab
10	0A	LF	line feed
11	0B	VT	vertical tab
12	0C	FF	form feed
13	0D	CR	enter/carriage-return
14	0E	SO	shift out
15	0F	SI	shift in
16	10	DLE	data link escape
17	11	DC1	device control 1
18	12	DC2	device control 2
19	13	DC3	device control 3
20	14	DC4	device control 4
21	15	NAK	negative ACK
22	16	SYN	synchronize
23	17	ETB	end of trans. block
24	18	CAN	cancel
25	19	EM	end of medium
26	1A	SUB	substitute
27	1B	ESC	escape
28	1C	FS	file separator
29	1D	GS	group separator
30	1E	RS	record separator
31	1F	US	unit separator
127	7F	DEL	delete

Printable ASCII Characters

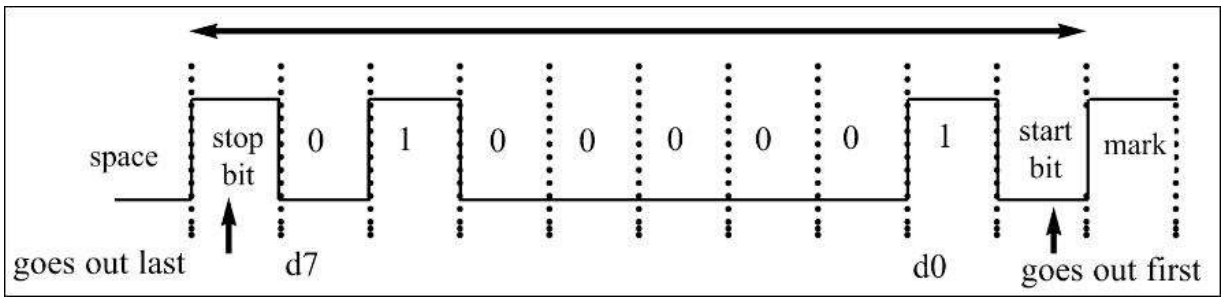
Dec	Hex	Char	Description
32	20	Space	space
33	21	!	exclamation mark
34	22	"	double quote
35	23	#	number
36	24	$	dollar
37	25	%	percent
38	26	&	ampersand
39	27	'	single quote
40	28	(	left parenthesis
41	29	)	right parenthesis
42	2A	*	asterisk
43	2B	+	plus
44	2C	,	comma
45	2D	-	minus
46	2E	.	period
47	2F	/	slash
48	30	0	zero
49	31	1	one
50	32	2	two
51	33	3	three
52	34	4	four
53	35	5	five
54	36	6	six
55	37	7	seven
56	38	8	eight
57	39	9	nine
58	3A	:	colon
59	3B	;	semicolon
60	3C	<	less than
61	3D	=	equality sign
62	3E	>	greater than
63	3F	?	question mark
64	40	@	"at" sign
65	41	A	
66	42	B	
67	43	C	
68	44	D	
69	45	E	
70	46	F	
71	47	G	
72	48	H	
73	49	I	
74	4A	J	
75	4B	K	
76	4C	L	
77	4D	M	
78	4E	N	
79	4F	O	
80	50	P	
81	51	Q	
82	52	R	
83	53	S	
84	54	T	
85	55	U	
86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5A	Z	
91	5B	[	left square bracket
92	5C	\	backslash
93	5D	]	right square bracket
94	5E	^	caret/circumflex
95	5F	_	underscore
96	60	`	grave/accent
97	61	a	
98	62	b	
99	63	c	
100	64	d	
101	65	e	
102	66	f	
103	67	g	
104	68	h	
105	69	i	
106	6A	j	
107	6B	k	
108	6C	l	
109	6D	m	
110	6E	n	
111	6F	o	
112	70	p	
113	71	q	
114	72	r	
115	73	s	
116	74	t	
117	75	u	
118	76	v	
119	77	w	
120	78	x	
121	79	y	
122	7A	z	
123	7B	{	left curly bracket
124	7C	|	vertical bar
125	7D	}	right curly bracket
126	7E	~	tilde
127	7F	DEL	delete

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2015) More cheatsheets at DCJTech.info

Hex Scan Code	Key	Hex Scan Code	Key	Hex Scan Code	Key	Hex Scan Code	Key	Hex Scan Code	Key	Hex Scan Code	Key
		20	D	40	F6	60	Ctrl F3	80	Alt 9	A0	Alt Dn Arrow
01	ESC	21	F	41	F7	61	Ctrl F4	81	Alt 0	A1	Alt PgDn
02	1	22	G	42	F8	62	Ctrl F5	82	Alt -	A2	Alt Ins
03	2	23	H	43	F9	63	Ctrl F6	82	Alt =	A3	Alt Del
04	3	24	J	44	F10	64	Ctrl F7	84	Ctrl PgUp	A4	Alt / (num)
05	4	25	K	45	Num Lk	65	Ctrl F8	85	F11	A5	Alt Tab
06	5	26	L	46	Scrl Lk	66	Ctrl F9	86	F12	A6	Alt Enter (num)
07	6	27	; :	47	Home	67	Ctrl F10	87	SH F11		
08	7	28	' "	48	Up Arrow	68	Alt F1	88	SH F12		
09	8	29	` ~	49	Pg Up	69	Alt F2	89	Ctrl F11		
0A	9	2A	L SH	4A	- (num)	6A	Alt F3	8A	Ctrl F12		
0B	0	2B	\ |	4B	4 Left Arrow	6B	Alt F4	8B	Alt F11		
0C	- _	2C	Z	4C	5 (num)	6C	Alt F5	8C	Alt F12		
0D	= +	2D	X	4D	6 Rt Arrow	6D	Alt F6	8C	Ctrl Up Arrow		
0E	BKSP	2E	C	4E	+ (num)	6E	Alt F7	8E	Ctrl - (num)		
0F	Tab	2F	V	4F	1 End	6F	Alt F8	8F	Ctrl 5 (num)		
10	Q	30	B	50	2 Dn Arrow	70	Alt F9	90	Ctrl + (num)		
11	W	31	N	51	3 Pg Dn	71	Alt F10	91	Ctrl Dn Arrow		
12	E	32	M	52	0 Ins	72	Ctrl PtScr	92	Ctrl Ins		
13	R	33	, <	53	Del .	73	Ctrl L Arrow	93	Ctrl Del		
14	T	34	. >	54	SH F1	74	Ctrl R Arrow	94	Ctrl Tab		
15	Y	35	/ ?	55	SH F2	75	Ctrl End	95	Ctrl / (num)		
16	U	36	R SH	56	SH F3	76	Ctrl PgDn	96	Ctrl * (num)		
17	I	37	PtScr	57	SH F4	77	Ctrl Home	97	Alt Home		
18	O	38	Alt	58	SH F5	78	Alt 1	98	Alt Up Arrow		
19	P	39	Spc	59	SH F6	79	Alt 2	99	Alt PgUp		
1A	[{	3A	CpsLk	5A	SH F7	7A	Alt 3	9A			
1B	] }	3B	F1	5B	SH F8	7B	Alt 4	9B	Alt Left Arrow		
1C	Enter	3C	F2	5C	SH F9	7C	Alt 5	9C			
1D	Ctrl	3D	F3	5D	SH F10	7D	Alt 6	9D	Alt Rt Arrow		
1E	A	3E	F4	5E	Ctrl F1	7E	Alt 7	9E			
1F	S	3F	F5	5F	Ctrl F2	7F	Alt 8	9F	Alt End		

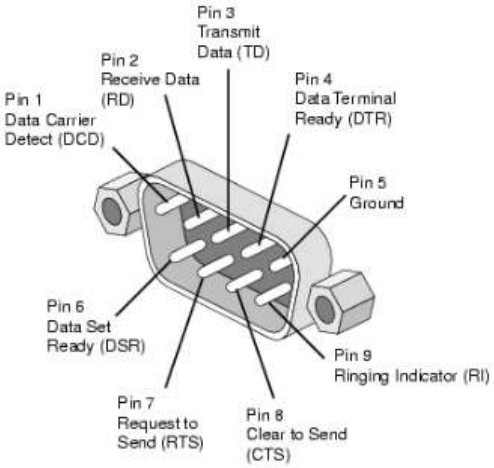
Port Programming and Serial Communication

- **Asynchronous Serial Comm. - Byte Framing**



- **Serial Cable RS232 pins**

Pin	SIG.	Signal Name	DTE (PC)
1	DCD	Data Carrier Detect	in
2	RXD	Receive Data	in
3	TXD	Transmit Data	out
4	DTR	Data Terminal Ready	out
5	GND	Signal Ground	-
6	DSR	Data Set Ready	in
7	RTS	Request to Send	out
8	CTS	Clear to Send	in
9	RI	Ring Indicator	in



- **Minimum bit-rate/baud-rate for different cable lengths by ft.**

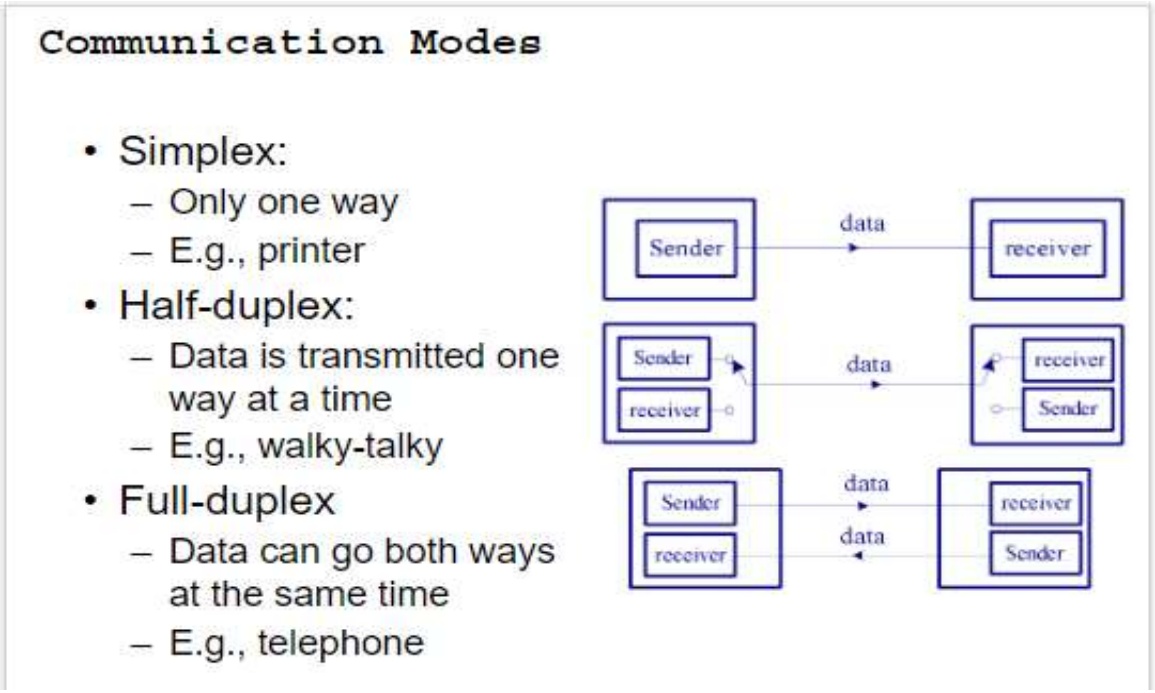
Baud rate	Maximum range / cable length
19200	50ft
9600	500ft
4800	1000ft
2400	3000ft

- Abbreviations :-
  - **PISO :** Parallel-in Serial-out
  - **SIPO :** Serial-in Parallel-out
  - **UART :** universal asynchronous receiver-transmitter
  - **USART** : universal synchronous-asynchronous receiver-transmitter
  - **ASCII :** American Standard Code for Information Interchange
  - **MSB:** Most significant bit
  - **LSB** : Least Significant bit
  - **STX :** Start of Text
  - **ETX:** End of Text
  - **TxD:** Transmit data out
  - **RxD:** Receive data in
  - **TxC:** Transmit data clock
  - **RxC:** Receive data clock
  - **Bps:** bit per second
  - **EIA :** Electronics Industries Association
  - **RS232 :** Recommended Standard 232
  - **TTL :** Transistor-Transistor Logic
  - **Modem:** Modulator/Demodulator
  - **DTE:** Data Terminal Equipment, usually a computer.
  - **DCE:** Data Communication Equipment, usually a modem.
  - **SG:** signal ground
  - **PC:** Personal Computer - in case you forgot :”
  - **I/O:** Input/Output - in case you forgot :”
  - **POST:** power-on self-test
  - For RS232 pins, look the RS232 Diagram
  - **COM:** Communication امال حضرتك فاكر ايه يعني
- **Serial vs. Parallel**

Serial	**Parallel**
Long distances	Short distances
One line only	Multiple lines
slower	faster
Cheap	Expensive
Data cannot be distorted by length	Data can be distorted by length

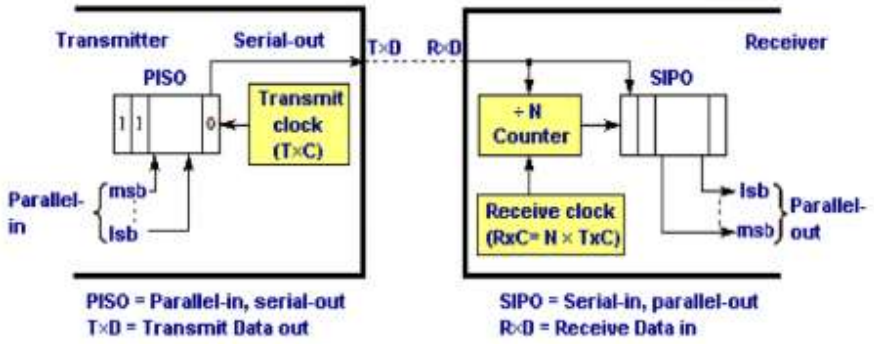
- **Async vs Sync**

Async	**Sync**
Suitable for communication that both terminals shouldn't be sync. with each other.	Suitable for real-time communication - like phone calls.
Character-oriented comm. Like in printer (data is sent char-by-char)	Block-oriented comm.
One-byte at a time	Chunk of data at a time



- **Some Extra notes on communication**
  - The x86 COM port uses the UART.
  - Asynchronous peripheral chips and modems can be programmed for data 5, 6, 7, or 8 bits wide and the number of stop bits, 1 or 2.
  - **Parity-bit:**
    - Comes after the MSB and before Stop bit.
    - Even parity ⇒ no. of 1's of data bits sent + parity bit is EVEN
    - Odd parity ⇒ no. of 1's of data bits sent + parity bit is ODD

Principle of operation and Timing:



- The rate of data transfer in serial data communication is stated in bps (bits per second); also baud rate. Baud and bps rates are not necessarily equal. Baud rate is **modem terminology**, defined as number of signal changes per second. **As far as the conductor wire is concerned, baud rate and bps are the same.**
- Data transfer rate depends on communication ports incorporated into the system, with today's x86 transferring data at rates as high as 115,200 bps. In asynchronous serial data communication, baud rate is generally limited to **less than 100,000 bps** using a telephone line & modem.
- RS232 input/output voltage is not TTL (Transistor-Transistor Logic)compatible:
  - Bit 1 is –3 to –25 V; Bit 0 +3 to +25 V.
  - +3 to –3 is undefined.
- To connect RS232 to a TTL-level chip requires voltage converters to convert the TTL logic levels to the RS232 voltage level, and vice versa. Commonly called ***line drivers***.
- Data transferred on the telephone line must be converted from 0s and 1s to audio tones (Sinosoidal-shaped signals). This conversion is performed by a modem.
- **RS232 Pins coordination**
  - **DTR (data terminal ready) -** when a terminal or PC COM port is turned on, after self-test, it sends signal DTR to indicate it is ready for communication. **DTE ⇒ ⇒ Modem.**

        - **DSR (data set ready) -** when a DCE (modem) is turned on and has gone through self-test, it asserts DSR to indicate that it is ready to communicate.(An active-low signal)
        - **RTS (request to send)** - when the DTE has a byte to transmit, it asserts RTS to signal the modem that it has a byte of data to transmit. (An active-low output, DTE ⇒ ⇒ modem)
        - **CTS (clear to send)** - when the modem has room for storing the data it is to receive, it sends signal CTS to DTE (PC) to indicate it can receive the data.
        - **CD (carrier detect, or DCD, data carrier detect)** - the modem asserts signal CDC to inform DTE (PC) that a valid carrier has been detected and that contact between it and the other modem is established.
        - **RI (ring indicator)** - an output from modem (DCE) and input to a PC (DTE), that indicates that the telephone is ringing. ON and OFF in synchronization with the ringing sound.
    - RTS and CTS are also referred to as hardware flow control signals.
    - In the x86 PC, as many as four COM ports can be installed, numbered 1, 2, 3, and 4, but BIOS numbers them as 0, 1, 2, and 3.
    - The PC POST (power-on self-test) tests the USART chip for each of the four COM ports. If they are installed, their I/O port addresses are written to memory locations 0040:0000–0040:0007.

SECTION E.4: INT 14H -- ASYNCHRONOUS COMMUNICATION

AH	Function
00	**Initialize COM Port**

Additional Call Registers	Result Registers
AL = parameter (see below)	AH = port status (see below)
DX = port number (0 if COM1, 1 if COM2, etc.)	AL = modem status (see below)

Note 1: The parameter byte in AL is defined as follows

7 6 5 4 3 2 1 0	Indicates
x x x	Baud rate (000=110, 001=150, 010=300, 011=600, 100=1200, 101=2400, 110=4800, 111=9600)
x x	Parity (01=odd, 11=even, x0=none)
x	Stop bits (0 = 1, 1 = 2)
x x	Word length (10=7 bits, 11=8 bits)

Note 2: The port status returned in AH is defined as follows

7 6 5 4 3 2 1 0	Indicates
1	Timed-out
1	Transmit shift register empty
1	Transmit holding register empty
1	Break detected
1	Framing error detected
1	Parity error detected
1	Overrun error detected
1	Received data ready

Note 3: The modem status returned in AL is defined as follows

7 6 5 4 3 2 1 0	Indicates
1	Received line signal detect
1	Ring indicator
1	DSR (data set ready)
1	CTS (clear to send)
1	Change in receive line signal detect
1	Trailing edge ring indicator
1	Change in DSR status
1	Change in CTS status

AH	Function

01 Write character to COM Port

Additional Call Registers	Result Registers
AL = character	AH bit 7 =0 if successful, 1 if not
DX = port number (0 if COM1,	AH bits 0 - 6 = status if successful
1 if COM2, etc.)	AL = character

Note: The status byte in AH, bits 0 - 6, after the call is as follows

6 5 4 3 2 1 0	Indicates
1	Transmit shift register empty
1	Transmit holding register empty
1	Break detected
1	Framing error detected
1	Parity error detected
1	Overrun error detected
1	Receive data ready

02 Read character from COM Port

Additional Call Registers	Result Registers
DX = port number (0 if COM1,	AH bit 7 =0 if successful, 1 if not
1 if COM2, etc.)	AH bits 0 - 6 = status if successful
	AL = character read

Note: The status byte in AH, bits 1 - 4, after the call is as follows

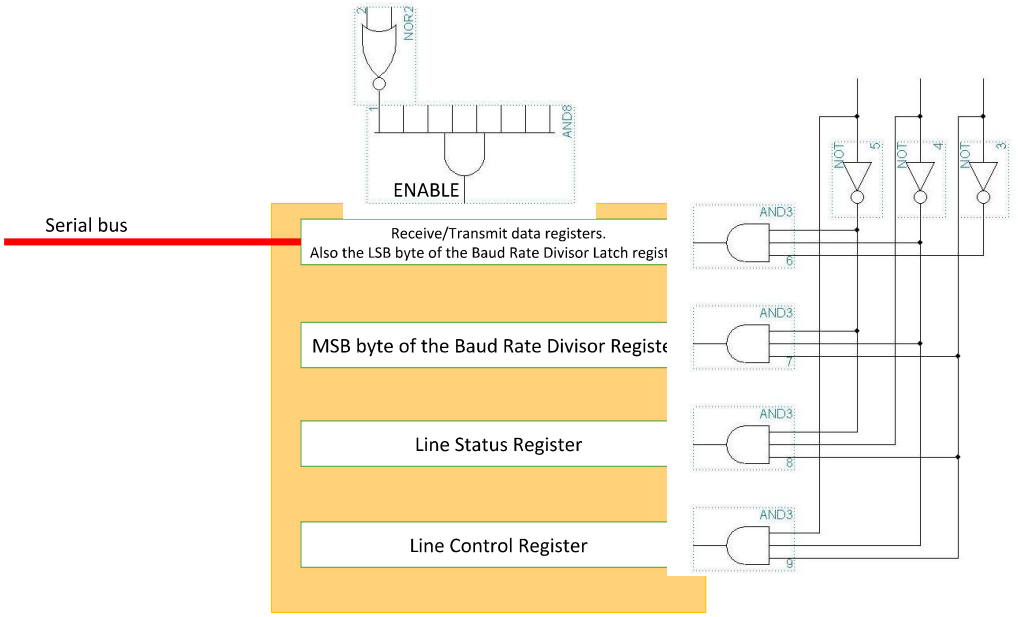
4 3 2 1	Indicates
1	Break detected
1	Framing error detected
1	Parity error detected
1	Overrun error detected

03 Read COM Port Status

Additional Call Registers	Result Registers
DX = port number (0 if COM1,	AH =port status
1 if COM2, etc.)	AL =modem status

Note: The port status and modem status returned in AH and AL are the same format as INT 14H function 00H, described above.

Lecture 9



IN/OUT

Instruction	Data Width	Function
IN AL, VALUE	8	A byte is input into AL from port (VALUE)
IN AX, VALUE	16	A word is input into AX from port (VALUE)
IN AL, DX	8	A byte is input into AL from the port addressed by DX
IN AX, DX	16	A word is input into AX from the port addressed by DX
INSB	8	A byte is input from the port addressed by DX and stored into the extra segment memory location addressed by DI, then DI = DI ± 1
INSW	16	A word is input from the port addressed by DX and stored into the extra segment memory location addressed by DI, then DI = DI ± 2
OUT VALUE, AL	8	A byte is output from AL into port (VALUE)
OUT VALUE, AX	16	A word is output from AL into port (VALUE)
OUT DX, AL	8	A byte is output from AL into the port addressed by DX
OUT DX, AX	16	A word is output from AX into the port addressed by DX
OUTSB	8	A byte is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 1
OUTSW	16	A word is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 2

Line Control Register

Bit 7	1	Divisor Latch Access Bit		
	0	Access to Receiver buffer, Transmitter buffer		
Bit 6	Set Break Enable			
Bits 3, 4 And 5	Bit 5	Bit 4	Bit 3	Parity Select
	X	X	0	No Parity
	0	0	1	Odd Parity
	0	1	1	Even Parity
	1	0	1	High Parity (Sticky)
	1	1	1	Low Parity (Sticky)
Bit 2	Length of Stop Bit			
	0	One Stop Bit		
	1	2 Stop bits for words of length 6,7 or 8 bits or 1.5 Stop Bits for Word lengths of 5 bits.		
Bits 0 And 1	Bit 1	Bit 0	Word Length	
	0	0	5 Bits	
	0	1	6 Bits	
	1	0	7 Bits	
	1	1	8 Bits	

Baud Rate Divisor Register Values

Bits Per Second	3F9/3F9 Value	3F8/2F8 Value
110	4	17h
300	1	80h
600	0	C0h
1200	0	60h
1800	0	40h
2400	0	30h
3600	0	20h
4800	0	18h
9600	0	0Ch
19.2K	0	6
38.4K	0	3
56K	0	1

$$Divisor\ value = \frac{X_{in}\ clock\ frequency}{baud\ rate \times 16}$$

Line Status Register

Bit	Notes
Bit 7	Error in Received FIFO
Bit 6	Transmitter shift Register is Empty
Bit 5	**Transmitter Holding Register is Empty**
Bit 4	Break Interrupt
Bit 3	Framing Error
Bit 2	Parity Error
Bit 1	Overrun Error
Bit 0	**Data Ready**

Port initialization

- **Set Divisor Latch Access Bit**

```
mov dx,3fbh          ; Line Control Register
mov al,10000000b     ;Set Divisor Latch Access Bit
out dx,al            ;Out it
```

Port initialization

- **;Set LSB byte of the Baud Rate Divisor Latch register.**

```
mov dx,3f8h
mov al,0ch
out dx,al
```

;Set MSB byte of the Baud Rate Divisor Latch register.

```
mov dx,3f9h
mov al,00h
out dx,al
```

Port initialization

•Set port configuration

```
mov dx,3fbh
mov al,00011011b
```

- •0:Access to Receiver buffer, Transmitter buffer
- •0:Set Break disabled
- •011:Even Parity
- •0:One Stop Bit
- •11:8bits

```
out dx,al
```

Port initialization

- Set Divisor Latch Access Bit
  - mov dx,3fbh ; Line Control Register
  - mov al,10000000b ;Set Divisor Latch Access Bit
  - out dx,al ;Out it
- ;Set LSB byte of the Baud Rate Divisor Latch register.
  - mov dx,3f8h
  - mov al,0ch
  - out dx,al
- ;Set MSB byte of the Baud Rate Divisor Latch register.
  - mov dx,3f9h
  - mov al,00h
  - out dx,al
- Set port configuration
  - mov dx,3fbh
  - mov al,00011011b
    - 0:Access to Receiver buffer, Transmitter buffer
    - 0:Set Break disabled
    - 011:Even Parity
    - 0:One Stop Bit
    - 11:8bits
  - out dx,al

Sending a value

```
;Check that Transmitter Holding Register is Empty
        mov dx , 3FDH      ; Line Status Register
AGAIN:  In al , dx         ;Read Line Status
        AND al , 00100000b
        JZ AGAIN

;If empty put the VALUE in Transmit data register
        mov dx , 3F8H      ; Transmit data register
        mov  al,VALUE
        out dx , al
```

Receiving a value

```
;Check that Data Ready
        mov dx , 3FDH      ; Line Status Register
    CHK: in al , dx
        AND al , 1
        JZ CHK

;If Ready read the VALUE in Receive data register
        mov dx , 03F8H
        in al , dx
        mov VALUE , al
```