8086 Instruction set

Two processing units- BIU and EU, 8-Byte prefetch queue architecture

Registers (16-bit)

general purpose registers

8-bit- AL, AH, BL,BH, CL,CH, DL,DH ;

16-bit- AX,BX,CX,DX

base registers- BX, BP,SP

index registers- SI,DI

instruction pointer- IP

segment registers- CS,DS,ES,SS

flag register bits- UUUU OF DF IF TF SF ZF U AF U PF U CF

(U undefined, OF overflow, DF direction, IF interrupt dis/enable, TF trap or single step)

20-bit address – segment register:base/index

CS:IP , DS:BX , ES:DI, SS:SP

(physical address - segment register value * 16+ base/index)

segment override (other than default segment)

Default segments Instruction-CS, stack-SS, Data -DS, string source -DS,string destination-ES,

interrupt- hardware maskable / non maskable interrupt , internal/external

- software

interrupt operation: TOS < flags; TF < 0; IF < 0; TOS < CS; TOS < IP; IP < M[N]; CS < M[N+2] where N is 4 × (Interrupt Type)

memory reference- (base [and] [or])index [+/-] displacement 8/16 bits)

memory address: no displacement – [SI] , [DI], [BX],[BX+SI],[BX+DI], [BP+SI],[BP+DI]

: 8-bit displacement- [SI+d8] , [DI+d8], [BX+d8],[BX+SI+d8],[BX+DI+d8], [BP+d8],[BP+SI+d8],[BP+DI+d8]

: 16-bit displacement- [SI+d16] , [DI+d16], [BX+d16],[BX+SI+d16],[BX+DI+d16], [BP+d16],[BP+SI+d16],[BP+DI+d16]

Instructions

operands – op1 , or op2 – can be Immediate(***Imd***),register(***Reg***),Memory(***Mem***) , Segment register(***Sreg***)

Data movement- MOV, LEA, LDS,LES,XCHG,PUSH, POP, PUSHF, POPF,XLAT, IN, OUT, MOVS, LODS, STOS,

MOV op1,op2- ; Reg,<Imd, Mem<Imd, Reg<Reg, Reg<Mem, Mem<Reg, Reg<Sreg, Sreg<Mem, Mem<Sreg, Reg<Sreg, Sreg<Reg ; Move data from op2 to op1

LEA Reg,Mem - ;Load Effective Address, address of memory in Reg

LDS Reg,Mem - ;Load memory double word into word register and DS(ES), low word in Reg, High word in Sreg(ES or DS) (same for LES)

XCHG op1,op2- Reg<>Reg, Reg<>Mem, Mem<>Reg; Exchange values of two operands

PUSH op1 -write on stack [TOS]< Reg ,[TOS]<Sreg,[TOS]< Mem; TOS=TOS-2 before Store 16 bit value in the stack

POP op1 - read on stack reg<[TOS] ,Sreg<[TOS],mem<[TOS], TOS=TOS+2 after Retrieve 16 bit value from the stack

PUSHF - ; [TOS]<Flag ;Put flags register from the stack.

POPF - ; Flag<[TOS]Get flags register from the stack.

IN reg, port - AL<[byte],AX<[byte],AX<[DX],AX<[DX]; Input from port to AL or AX . Second operand is port number. port number over 255 – use DX reg

OUT port,reg - [byte]<AL,[byte]<AX,[DX]<AL,[DX]<AX; Output from AL or AX to port. First operand is a port number. .....

XLAT - ;Translate (copy) byte from table (memory) at DS:[BX + unsigned AL] to AL register.

[REP] MOVS[B/W] - ; Copy byte/word at DS:[SI] to ES:[DI]. Update SI and DI.CX contains number of transfer when REP is used.

[REP] LODS[B/W] -;Load byte/word at DS:[SI] into AX. Update SI. CX contains number of transfer when REP is used.

[REP] STOS[B/W] - ; Store byte/word in AL into ES:[DI]. Update DI CX contains number of transfer when REP is used.

Data processing (affects the flags) ADD,ADC,SUB,SBB,MUL,IMUL,DIV,IDIV,CMP,NOT,DEC,INC,AND,XOR,OR,TEST,NOT,SHL, SHR, SAR, RCR,RCL,ROL,ROR, AAA,AAS,AAM,AAD,DAA,DAB,

ADD op1,op2 - addition op1=op1+op2; Reg=Reg+Mem,Mem=Mem+Reg,Reg=Reg+Reg,Mem=Mem+Imd,Reg=Reg+Imd,

ADC op1,op2 – addition with Carry

SUB op1,op2 - subtraction

SBB op1,op2 – subtraction with Borrow

AND op1,op2 – logical AND

OR op1,op2 - logical OR
XOR op1,op2 – logical XOR
CMP op1,op2 - subtraction but result is not stored only flags affected
TEST op1,op2 – logical AND but result is not stored only flags affected
INC op1 - Reg , Mem; Increment
DEC op1 - Reg , Mem; decrement
NEG op1 - Reg , Mem; Negate. Makes operand negative (two's complement).
NOT op1 - Reg , Mem; Invert each bit of the operand. (one's complement).
MUL op1- Unsigned multiply. when operand is a byte: AX = AL * operand. When operand is a word: (DX :AX) = AX * operand. When Flag CF=OF=0 result fits into operand of MUL else extended to DX:AX
IMUL op1- Signed multiply. As MUL
DIV op1- Reg , Mem;Unsigned divide. When operand is a byte: AL = AX / operand ;AH = remainder (modulus). When operand is a word: AX = (DX :AX) / operand ;DX = remainder (modulus)
IDIV op1- Reg , Mem; Signed divide. As DIV
AAA ; ASCII Adjust after Addition. Corrects result in AH and AL after addition when working with BCD values.
AAS ; ASCII Adjust after Subtraction. Corrects result in AH and AL after subtraction when working with BCD values.
AAD ; ASCII Adjust before Division. (Prepares two BCD values(byte) for division.
AAM ; ASCII Adjust after Multiplication.
DAA ; Decimal adjust After Addition.
DAS ; Decimal adjust After Subtraction.
SHL op1,op2- Reg <1/CL,Mem<1/CL, ; Shift Arithmetic op1 Left. The number of shifts is set by op2.(Op2 is immediate is shift count is 1 , else shift count in CL). The bit that goes off goes to CF, OF =1 if sign changed)(SAL)
SHR op1,op2 ; Shift logical op1 Right.
SAR op1,op2 ; Shift Arithmetic op1 Right (retains msb)The bit that goes off goes to CF, OF =1 if sign changed)
ROL op1,op2 ; Rotate op1 left. The number of rotates is set by op2. Shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position. OF =1 if sign changed)
ROR op1,op2 ; Rotate op1 right.
RCL op1,op2 ; Rotate op1 left through Carry Flag
RCR op1,op2 ; Rotate op1 right through Carry Flag
[REPE/NE] CMPS[B/W] ;Compare bytes: ES:[DI] from DS:[SI] Flag=[SI]-[DI],repeat until CX=0 or equal/unequal
[REPE/NE] SCAS[B/W] ; Compare bytes: AL/AX from ES:[DI]. Flag=AL/AX-[DI]

Control Instructions (Flow change)– LOOP, REP,CALL,RET,IRET,JMP,Jcc INT
LOOP label ; Decrease CX, jump to label if CX not zero. CX = CX - 1 , if CX <> 0 then jump else no jump, continue
LOOP[E/Z] label; if (CX <> 0) and (ZF = 1) then jump
LOOP[NE/NZ] label; if (CX <> 0) and (ZF =0) then jump
REP ; Repeat following MOVS, LODS, STOS, instructions CX times
REP[E/Z] ; Repeat following CMPS,SCAS instructions CX times with ZF=1
REP[NE/NZ] ; Repeat following CMPS,SCAS instructions CX times with ZF=0
CALL Imd,Reg or Mem; Direct (Imd), indirect using Reg/mem,Transfer control to near or far procedure , return address in stack
JMP Reg or Mem; Unconditional near Jump address in Reg or Mem, far jump 4 byte Imd or mem
RET ; Return from near procedure. IP=[TOS]
RET n; Return from near procedure , IP=[TOS] ,SP=SP+n
RETF ;Return from Far procedure. IP=[TOS] CS=[TOS+2]
RETF n; Return from Far procedure., IP=[TOS] CS=[TOS+2], SP=SP+n
INTO ; Interrupt caused by overflow flag
INT n ; software interrupt number n (n is Byte byte)
IRET ; Return from Interrupt procedure. IP=[TOS] CS=[TOS+2],Flag=[TOS+4]
Jcc label; conditional short jump;flag based JO,JNO,JS,JNS,JC,JNC,JNP/JPO,JP/JPE,JE/JZ,JNE/JNZ
unsigned conditions- JA/JNBE,JAE/JNB,JB/JNAE,JBE/JNA unsigned conditions- JE/JZ,JG/JNLE,JGE/JNL,JL/JNGE,JLE/JNG,
JCXZ label; jump when CX=0
(JMP and CALL – Near or Far using direct/ indirect addressing). In Loop /Jcc label is always direct and short

Special Instructions-

CBW ; convert byte to signed word ; AL to AX
CWD ; Convert word to signed double word; AX to DX:AX
CLC ; clear Carry
CMC ; complement Carry
STC ; set carry
CLD ; clear Direction (auto increment)
STD ;set Direction (auto decrement)
CLI ; clear interrupt Flag
STI ; set interrupt Flag
NOP ; no operation

HLT ; Wait until RESET, NMI, or Enable Interrupt
ESC ; Fetch operand if memory operand, then NOP, for Coprocessor Instruction
WAIT ; wait until TEST line goes low, for slow memory
LOCK ; prefix byte in multiprocessors board for memory access