APPENDIX

H

8086 INSTRUCTION SET

Instructions	Interpretation	Comments
AAA	ASCII adjust [AL] after addition	This instruction has implied addressing mode; this instruction is used to adjust the content of AL after addition of two ASCII characters
AAD	ASCII adjust for division	This instruction has implied addressing mode; converts two unpacked BCD digits in AX into equivalent binary numbers in AL; AAD must be used before dividing two unpacked BCD digits by an unpacked BCD byte
AAM	ASCII adjust after multiplication	This instruction has implied addressing mode; after multiplying two unpacked BCD numbers, adjust the product in AX to become an unpacked BCD result; ZF, SF, and PF are affected
AAS	ASCII adjust [AL] after subtraction	This instruction has implied addressing mode used to adjust [AL] after subtraction of two ASCII characters
ADC mem/reg 1, mem/reg 2	[mem/reg 1] ← [mem/reg 1] + [mem/reg 2] + CF	Memory or register can be 8- or 16-bit; all flags are affected; no segment registers are allowed; no memory-to-memory ADC is permitted
ADC mem, data	[mem] ← [mem] + data + CF	Data can be 8- or 16-bit; mem uses DS as the segment register; all flags are affected
ADC reg, data	[reg] ←[reg] + data + CF	Data can be 8- or 16-bit; register cannot be segment register; all flags are affected
ADD mem/reg 1, mem/reg 2	[mem/reg 1] ← [mem/reg 2] + [mem/reg 1]	Add two 8- or 16-bit data; no memory-to-memory ADD is permitted; all flags are affected; mem uses DS as the segment register; reg 1 or reg 2 cannot be segment register
ADD mem, data	[mem] ← [mem] + data	Mem uses DS as the segment register; data can be 8-or 16-bit; all flags are affected
ADD reg, data	[reg] ← [reg] + data	Data can be 8- or 16-bit; no segment registers are allowed; all flags are affected
AND mem/reg 1, mem/reg 2	[mem/reg 1] ← [mem/reg 1] ^ [mem/reg 2]	This instruction logically ANDs 8- or 16-bit data in [mem/reg 1] with 8- or 16-bit data in [mem/reg 2]; all flags are affected; OF and CF are cleared to zero; no segment registers are allowed; no memory-to-memory operation is allowed; mem uses DS as the segment register
AND mem, data	[mem] ← [mem] ^ data	Data can be 8- or 16-bit; mem uses DS as the segment register; all flags are affected with OF and CF always cleared to zero
AND reg, data	[reg] ← [reg] + data	Data can be 8- or 16-bit; reg cannot be segment register; all flags are affected with OF and CF cleared to zero

Instructions	Interpretation	Comments
CALL PROC (NEAR)	Call a subroutine in the same segment with signed 16-bit displacement (to CALL a subroutine in ±32K)	NEAR in the statement BEGIN PROC NEAR indicates that the subroutine 'BEGIN' is in the same segment and BEGIN is 16-bit signed; CALL BEGIN instruction decrements SP by 2 and then pushes IP onto the stack and then adds the signed 16-bit value of BEGIN to IP and CS is unchanged; thus, a subroutine is called in the same segment (intrasegment direct)
CALL reg 16	CALL a subroutine in the same segment addressed by the contents of a 16-bit general register	The 8086 decrements SP by 2 and then pushes IP onto the stack, then specified 16-bit register contents (such as BX, SI, and DI) provide the new value for IP; CS is unchanged (intrasegment indirect)
CALL mem 16	CALL a subroutine addressed by the content of a memory location pointed to by 8086 16-bit register such as BX, SI, and DI	The 8086 decrements SP by 2 and pushes IP onto the stack; the 8086 then loads the contents of a memory location addressed by the content of a 16-bit register such as BX, SI, and DI into IP; [CS] is unchanged (intrasegment indirect)
CALL subroutine in another segment	CALL a subroutine in another segment	FAR in the statement BEGIN PROC FAR indicates that the subroutine 'BEGIN' is in another segment and the value of BEGIN is 32 bit wide The 8086 decrements SP by 2 and pushes CS onto the stack and moves the low 16-bit value of the specified 32-bit number such as 'BEGIN' in CALL BEGIN into CS; SP is again decremented by 2; IP is pushed onto the stack; IP is then loaded with high 16-bit value of BEGIN; thus, this instruction CALLS a subroutine in another code segment (intersegment direct)
CALL DWORDPTR [reg 16]	CALL a subroutine in another segment	This instruction decrements SP by 2, and pushes CS onto the stack; CS is then loaded with the contents of memory locations addressed by [reg 16+2] and [reg 16 + 3] in DS; the SP is again decremented by 2; IP is pushed onto the stack; IP is then loaded with the contents of memory locations addressed by [reg 16] and [reg 16 + 1] in DS; typical 8086 registers used for reg 16 are BX, SI, and DI (intersegment indirect)
CBW	Convert a byte to a word	Extend the sign bit (bit 7) ofAL register into AH
CLC	CF ← 0	Clear carry to zero
CLD	DF ← 0	Clear direction flag to zero
CLI	IF ← 0	Clear interrupt enable flag to zero to disable maskable interrupts
CMC	CF ← $\overline{CF}$	One's complement carry
CMP mem/reg 1, mem/reg 2	[mem/reg 1] – [mem/reg 2], flags are affected	mem/reg can be 8- or 16-bit; no memory-to-memory comparison allowed; result of subtraction is not provided; all flags are affected
CMP mem/reg, data	[mem/reg] – data, flags are affected	Subtracts 8- or 16-bit data from [mem or reg] and affects flags; no result is provided
CMPS BYTE or CMPSB	FOR BYTE [[SI]] – [[DI]], flags are affected [SI] ← [SI] ± 1 [DI] ← [DI] ± 1	8- or 16-bit data addressed by [DI] in ES is subtracted from 8- or 16-bit data addressed by SI in DS and flags are affected without providing any result; if DF = 0, then SI and DI are incremented by one for byte and two for word; if DF = 1, then SI and DI are decremented by one for byte and two for word;
CMPS WORD or CPSW	FOR WORD [[SI]] – [[DI]], flags are affected [SI] ← [SI] ± 2 [DI] ← [DI] ± 2	the segment register ES in destination cannot be overridden
CWD	Convert a word to 32 bits	Extend the sign bit of AX (bit 15) into DX

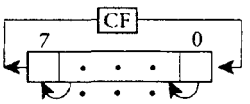
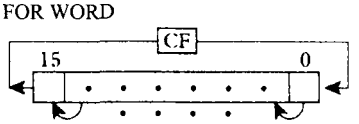
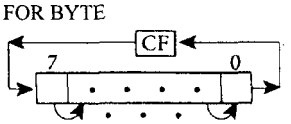
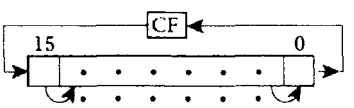
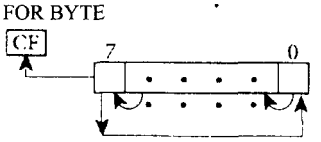
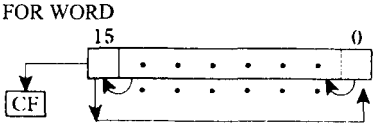
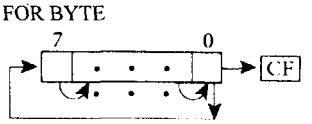
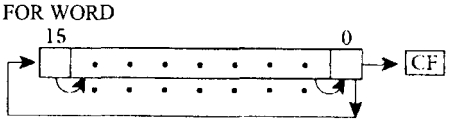
Instructions	Interpretation	Comments
DAA	Decimal adjust [AL] after addition	This instruction uses implied addressing mode; this instruction converts [AL] into BCD; DAA should be used after BCD addition
DAS	Decimal adjust [AL] after subtraction	This instruction uses implied addressing mode; converts [AL] into BCD; DAS should be used after BCD subtraction
DEC reg 16	[reg 16] ← [reg 16] – 1	This is a one-byte instruction; used to decrement a 16-bit register except segment register; does not affect the carry flag
DEC mem/reg 8	[mem] ← [mem] – 1 or [reg 8] ← [reg 8] – 1	Used to decrement a byte or a word in memory or an 8-bit register content; segment register cannot be decremented by this instruction; does not affect carry flag
DIV mem/reg	16/8 bit divide: [AX] / [mem8 / reg8] [AH] ← Remainder [AL] ← Quotient 32/16 bit divide: [DX] [AX] / [mem16 / reg16] [DX] ← Remainder, [AX] ← Quotient	Mem/reg is 8-bit for 16-bit by 8-bit divide and 16-bit for 32-bit by 16-bit divide; this is an unsigned division; no flags are affected; division by zero automatically generates an internal interrupt
ESC external OP code, source	ESCAPE to external processes	This instruction is used to pass instructions to a coprocessor such as the 8087 floating point coprocessor which simultaneously monitors the system bus with the 8086; the coprocessor OP codes are 6-bit wide; the coprocessor treats normal 8086 instructions as NOP's; the 8086 fetches all instructions from memory; when the 8086 encounters an ESC instruction, it usually treats it as NOP; the coprocessor decodes this instruction and carries out the operation using the 6-bit OP code independent of the 8086; for ESC OP code, memory, the 8086 accesses data in memory for the coprocessor; for ESC data, register, the coprocessor operates on 8086 registers; the 8086 treats this as an NOP
HLT	HALT	Halt
IDIV mem/reg	Same as DIV mem/reg	Signed division. No flags are affected.
IMUL mem/reg	For 8 x 8 [AX] ← [AL] * [mem 8 / reg 8] For 16 x 16 [DX][AX]← [AX] * [mem 16 / reg 16]	Mem/reg can be 8- or 16-bit; only CF and OF are affected; signed multiplication
IN AL, DX	[AL] ← PORT [DX]	Input AL with the 8-bit content of a port addressed by DX; this is a one-byte instruction
IN AX, DX	[AX] ← PORT [DX]	Input AX with the 16-bit content of a port addressed by DX and DX + 1; this is a one-byte instruction
IN AL, PORT	[AL] ← [PORT]	Input AL with the 8-bit content of a port addressed by the second byte of the instruction
IN AX, PORT	[AX] ← [PORT]	Input AX with the 16-bit content of a port addressed by the 8-bit address in the second byte of the instruction
INC reg 16	[reg 16] ← [reg 16] + 1	This is a one-byte instruction; used to increment a 16-bit register except the segment register; does not affect the carry flag

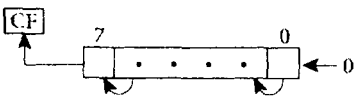
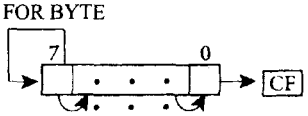
Instructions	Interpretation	Comments
INC mem/reg 8	[mem] ← [mem] + 1 or [reg 8] ← [reg 8] + 1	This is a two-byte instruction; can be used to increment a byte or word in memory or an 8-bit register content; segment registers cannot be incremented by this instruction; does not affect the carry flag
INT n (n can be zero thru 255)	[SP] ← [SP] – 2 ,[[SP]] ← Flags IF ← 0, TF ← 0 [SP] ← [SP] – 2, [[SP]] ← [CS] [CS] ← 4n + 2 [SP] ← [SP] – 2 [[SP]] ← [IP] [IP] ← 4n	Software interrupts can be used as supervisor calls; that is, request for service from an operating system; a different interrupt type can be used for each type of service that the operating system could supply for an application or program; software interrupt instructions can also be used for checking interrupt service routines written for hardware-initiated interrupts
INTO	Interrupt on Overflow	Generates an internal interrupt if OF = 1; executes INT 4; can be used after an arithmetic operation to activate a service routine if OF = 1; when INTO is executed and if OF = 1, operations similar to INT n take place
IRET	Interrupt Return	POPS IP, CS and Flags from stack; IRET is used as return instruction at the end of a service routine for both hardware and software interrupts
JA/JNBE disp 8	Jump if above/jump if not below or equal	Jump if above/jump if not below or equal with 8-bit signed displacement; that is, the displacement can be from -128_{10} to $+127_{10}$, zero being positive; JA and JNBE are the mnemonic which represent the same instruction; Jump if both CF and ZF are zero; used for unsigned comparison
JAE/JNB/JNC disp 8	Jump if above or equal/jump if not below/jump if no carry	Same as JA/JNBE except that the 8086 Jumps if CF = 0; used for unsigned comparison
JB/JC/JNAE disp 8	Jump if below/jump if carry/jump if not above or equal	Same as JA/JNBE except that the jump is taken CF = 1, used for unsigned comparison
JBE/JNA disp 8	Jump if below or equal/jump if not above	Same as JA/JNBE except that the jump is taken if CF = 1 or ZF = 0; used for unsigned comparison
JCXZ disp 8	Jump if CX = 0	Jump if CX = 0; this instruction is useful at the beginning of a loop to bypass the loop if CX = 0
JE/JZ disp 8	Jump if equal/jump if zero	Same as JA/JNBE except that the jump is taken if ZF = 1; used for both signed and unsigned comparison
JG/JNLE disp 8	Jump if greater/jump if not less or equal	Same as JA/JNBE except that the jump is taken if ((SF ⊕ OF) or ZF) = 0; used for signed comparison
JGE/JNL disp 8	Jump if greater or equal/ jump if not less	Same as JA/JNBE except that the jump is taken if (SF ⊕ OF) = 0; used for signed comparison
JL/JNGE disp 8	Jump if less/Jump if not greater nor equal	Same as JA/JNBE except that the jump is taken if (SF ⊕ OF) = 1; used for signed comparison
JLE/JNG disp 8	Jump if less or equal/ jump if not greater	Same as JA/JNBE except that the jump is taken if ((SF ⊕ OF) or ZF) = 1; used for signed comparison

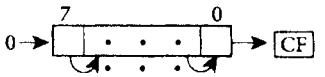
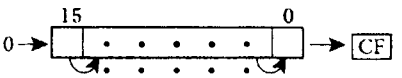
Instructions	Interpretation	Comments
JMP Label	Unconditional Jump with a signed 8-bit (SHORT) or signed 16-bit (NEAR) displacement in the same segment	The label START can be signed 8-bit (called SHORT jump) or signed 16-bit (called NEAR jump) displacement; the assembler usually determines the displacement value; if the assembler finds the displacement value to be signed 8-bit (–128 to +127, 0 being positive), then the assembler uses two bytes for the instruction: one byte for the OP code followed by a byte for the displacement; the assembler sign extends the 8-bit displacement and then adds it to IP; [CS] is unchanged; on the other hand, if the assembler finds the displacement to be signed 16-bit (±32 K), then the assembler uses three bytes for the instruction: one byte for the OP code followed by 2 bytes for the displacement; the assembler adds the signed 16-bit displacement to IP; [CS] is unchanged; therefore, this JMP provides a jump in the same segment (intrasegment direct jump)
JMP *reg16*	[IP] ← [reg 16]; [CS] is unchanged	Jump to an address specified by the contents of a 16-bit register such as BX, Sl, and DI in the same code segment; in the example JMP BX, [BX] is loaded into IP and [CS] is unchanged (intrasegment memory indirect jump)
JMP mem 16	[IP] ← [mem]; [CS] is unchanged	Jump to an address specified by the contents of a 16-bit memory location addressed by 16-bit register such as BX, SI, and DI; in the example, JMP [BX] copies the content of a memory location addressed by BX in DS into IP; CS is unchanged (intrasegment memory indirect jump)
JMP Label (to another segment)	Unconditionally jump to another segment	This is a 5-byte instruction: the first byte is the OP code followed by four bytes of 32-bit immediate data; bytes 2 and 3 are loaded into IP; bytes 4 and 5 are loaded into CS to JUMP unconditionally to another segment (intersegment direct)
JMP DWORDPTR [reg 16]	Unconditionally jump to another segment	This instruction loads the contents of memory locations addressed by [reg 16] and [reg 16 + 1] in DS into IP; it then loads the contents of memory locations addressed by [reg 16 + 2] and [reg 16 + 3] in DS into CS; typical 8086 registers used for reg 16 are BX, SI, and DI (intersegment indirect)
JNE/JNZ disp 8	Jump if not equal/jump if not zero	Same as JA/JNBE except that the jump is taken if ZF = 0; used for both signed and unsigned comparison
JNO disp 8	Jump if not overflow	Same as JA/JNBE except that the jump is taken if OF = 0
JNP/JPO disp 8	Jump if no parity/jump if parity odd	Same as JA/JNBE except that the jump is taken if PF = 0
JNS disp 8	Jump if not sign	Same as JA/JNBE except that the jump is taken if SF = 0
JO disp 8	Jump if overflow	Same as JA/JNBE except that the jump is taken if OF = 1
JP/JPE disp 8	Jump if parity/jump if parity even	Same as JA/JNBE except that the jump is taken if PF = 1
JS disp 8	Jump if sign	Same as JA/JNBE except that the jump is taken if SF = 1
LAHF	[AH] ← Flag low-byte	This instruction has implied addressing mode; it loads AH with the low byte of the flag register; no flags are affected

Instructions	Interpretation	Comments
LDS reg, mem	[reg] ← [mem] [DS] ← [mem + 2]	Load a 16-bit register (AX, BX, CX, DX, SP, BP, SI, DI) with the content of specified memory and load DS with the content of the location that follows; no flags are affected; DS is used as the segment register for mem
LEA reg, mem	[reg] ← [offset portion of address]	LEA (load effective address) loads the value of the source operand rather than its content to register (such as SI, DI, BX) which are allowed to contain offset for accessing memory; no flags are affected
LES reg, mem	[reg] ← [mem] [ES] ← [mem+ 2]	DS is used as the segment register for mem; in the example LES DX, [BX], DX is loaded with 16-bit value from a memory location addressed by 20-bit physical address computed from DS and BX; the 16-bit content of the next memory is loaded into ES; no flags are affected
LOCK	LOCK bus during next instruction	Lock is a one-byte prefix that causes the 8086 (configured in maximum mode) to assert its bus LOCK signal while following instruction is executed; this signal is used in multiprocessing; the LOCK pin of the 8086 can be used to LOCK other processors off the system bus during execution of an instruction; in this way, the 8086 can be assured of uninterrupted access to common system resources such as shared RAM
LODS BYTE or LODSB LODS WORD or LODSW	FOR BYTE [AL] ← [[SI]] [SI] ← [SI] ± 1 FOR WORD [AX] ← [[SI]],[SI] ← [SI] ± 2	Load 8-bit data into AL or 16-bit data into AX from a memory location addressed by SI in segment DS; if DF = 0, then SI is incremented by 1 for byte or incremented by 2 for word after the load; if DF = 1, then SI is decremented by 1 for byte or decremented by 2 for word; LODS affects no flags
LOOP disp 8	Loop if CX not equal to zero	Decrement CX by one, without affecting flags and loop with signed 8-bit displacement (from –128 to +127, zero being positive) if CX is not equal to zero
LOOPE/I.OOPZ disp 8	Loop while equal/loop while zero	Decrement CX by one without affecting flags and loop with signed 8-bit displacement if CX is equal to zero, and if ZF = 1 which results from execution of the previous instruction
LOOPNE/ LOOPNZ disp 8	Loop while not equal/loop while not zero	Decrement CX by one without affecting flags and loop with signed 8-bit displacement if CX is not equal to zero and ZF = 0 which results from execution of previous instruction
MOV mem/reg 2, mem/reg 1	[mem/reg 2] ← [mem/reg 1]	mem uses DS as the segment register; no memory-to-memory operation allowed; that is, MOV mem, mem is not permitted; segment register cannot be specified as reg or reg; no flags are affected; not usually used to load or store 'A' from or to memory
MOV mem, data	[mem] ← data	mem uses DS as the segment register; 8- or 16-bit data specifies whether memory location is 8- or 16-bit; no flags are affected
MOV reg, data	[reg] ← data	Segment register cannot be specified as reg; data can be 8- or 16-bit; no flags are affected
MOV segreg, mem/reg	[segreg] ← [mem/reg]	mem uses DS as segment register; used for initializing CS, DS, ES, and SS; no flags are affected
MOV mem/reg, segreg	[mem/reg] ← [segreg]	mem uses DS as segment register; no flags are affected

Instructions	Interpretation	Comments
MOVS BYTE or MOVSB	FOR BYTE [[DI]]← [[SI]] [SI] ← [SI] ± 1	Move 8-bit or 16-bit data from the memory location addressed by SI in segment DS location addressed by DI in ES; segment DS can be overridden by a prefix but destination segment must be ES and cannot be overridden; if DF = 0, then SI is incremented by one for byte or incremented by two for word; if DF = 1, then SI is decremented by one for byte or by two for word
MOVS WORD or MOVSW	FOR WORD [[DI]]← [[SI]] [SI] ← [SI] ± 2	
MUL mem/reg	FOR 8 × 8 [AX] ← [AL] * [mem/reg] FOR 16 × 16 [DX] [AX] ← [AX] * [mem/reg]	mem/reg can be 8- or 16-bit; only CF and OF are affected; unsigned multiplication
NEG mem/reg	[mem/reg] ← $\overline{[mem/reg]}$ + 1	mem/reg can be 8- or 16-bit; performs two's complement subtraction of the specified operand from zero, that is, two's complement of a number is formed; all flags are affected except CF = 0 if [mem/reg] is zero; otherwise CF = 1
NOP	No Operation	8086 does nothing
NOT reg	[reg] ← $\overline{[reg]}$	mem and reg can be 8- or 16-bit; segment registers are not allowed; no flags are affected; ones complement reg
NOT mem	[mem] ← $\overline{[mem]}$	mem uses DS as the segment register; no flags are affected; ones complement mem
OR Mem/reg 1, Mem/reg 2	[mem/reg 1] ← [mem/reg 1] v [mem/reg 2]	No memory-to-memory operation is allowed; [mem] or [reg 1] or [reg 2] can be 8- or 16-bit; all flags are affected with OF and CF cleared to zero; no segment registers are allowed; mem uses DS as segment register
OR mem, data	[mem] ← [mem] v data	mem and data can be 8- or 16-bit; mem uses DS as segment register; all flags are affected with CF and OF cleared to zero
OR reg, data	[reg] ← [reg] v data	reg and data can be 8- or 16-bit; no segment registers are allowed; all flags are affected with CF and OF cleared to zero
OUT DX, AL	PORT [DX] ← [AL]	Output the 8-bit contents of AL into an I/O Port addressed by the 16-bit content of DX; this is a one-byte instruction
OUT DX, AX	PORT [DX] ← [AX]	Output the 16-bit contents of AX into an I/O Port addressed by the 16-bit content of DX; this is a one-byte instruction
OUT PORT, AL	PORT ← [AL]	Output the 8-bit contents of AL into the Port specified in the second byte of the instruction
OUT PORT, AX	PORT ← [AX]	Output the 16-bit contents of AX into the Port specified in the second byte of the instruction
POP mem	[mem] ← [[SP]],[SP] ← [SP] + 2	mem uses DS as the segment register; no flags are affected
POP reg	[reg]← [[SP]] ,[SP] ← [SP] + 2	Cannot be used to POP segment registers or flag register
POP segreg	[segreg] ← [[SP]] [SP] ← [SP] + 2	POP CS is illegal
POPF	[Flags] ← [[SP]] [SP] ← [SP] + 2	This instruction pops the top two stack bytes in the16-bit flag register
PUSH mem	[SP] ← [SP] – 2 [[SP]] ← [mem]	mem uses DS as segment register; no flags are affected; pushes 16-bit memory contents

Instructions	Interpretation	Comments
PUSH reg	[SP] ← [SP] – 2 [[SP]] ← [reg]	reg must be a 16-bit register; cannot be used to PUSH segment register or Flag register
PUSH segreg	[SP] ← [SP] – 2 [[SP]] ← [segreg]	PUSH CS is illegal
PUSHF	[SP] ← [SP] – 2 [[SP]] ← [Flags]	This instruction pushes the 16-bit Flag register onto the stack
RCL mem/reg, 1	ROTATE through carry left once byte or word in mem/reg	FOR BYTE  FOR WORD 
RCL mem/reg, CL	ROTATE through carry left byte or word in mem/reg by [CL]	Operation same as RCL mem/reg, 1 except the number of rotates is specified in CL for rotates up to 255; zero or negative rotates are illegal
RCR mem/reg, 1	ROTATE through carry right once byte or word in mem/reg	FOR BYTE  FOR WORD 
RCR mem/reg, CL	ROTATE through carry right byte or word in mem/reg by [CL]	Operation same as RCR mem/reg, 1 except the number of rotates is specified in CL for rotates up to 255; zero or negative rotates are illegal
ROL mem/reg, 1	ROTATE left once byte or word in mem/reg	FOR BYTE  FOR WORD 
ROL mem/reg, CL	ROTATE left byte or word by the content of CL	[CL] contains rotate count up to 255; zero and negative shifts are illegal; CL is used to rotate count when the rotate is greater than once; mem uses DS as the segment register
ROR mem/reg, 1	ROTATE right once byte or word in mem/reg	FOR BYTE  FOR WORD 

Instructions	Interpretation	Comments
ROR mem/reg, CL	ROTATE right byte or word in mem/reg by [CL]	Operation same as ROR mem/reg, 1; [CL] specifics the number of rotates for up to 255; zero and negative rotates are illegal; mem uses DS as the segment register
SAHF	[Flags, low-byte] ← [AH]	This instruction stores the contents of the AH register in the low-byte of the flag register; OF, DF, IF, and TF flags are not affected.
SAL mem/reg, 1	Shift arithmetic left once byte or word in mem or reg	FOR BYTE  FOR WORD  Mem uses DS as the segment register; reg cannot be segment registers; OF and CF are affected; if sign bit is changed during or after shifting, the OF is set to one
SAL mem/reg, CL	Shift arithmetic left byte or word by shift count on CL	Operation same as SAL mem/reg, 1; CL contains shift count for up to 255; zero and negative shifts are illegal; [CL] is used as shift count when shift is greater than one; OF and SF are affected; if sign bit of [mem] is changed during or after shifting, the OF is set to one; mem uses DS as segment register
SAR mem/reg, 1	SHIFT arithmetic right once byte or word in mem/reg	FOR BYTE  FOR WORD 
SAR mem/reg, CL	SHIFT arithmetic right byte or word in mem/reg by [CL]	Operation same as SAR mem/reg, 1; however, shift count is specified in CL for shifts up to 255; zero and negative shifts are illegal
SBB mem/reg 1, mem/reg 2	[mem/reg 1] ← [mem/reg 1] - [mem/reg 2] – CF	Same as SUB mem/reg 1, mem/reg 2 except this is a subtraction with borrow
SBB mem, data	[mem] ← [mem] - data - CF	Same as SUB mem, data except this is a subtraction with borrow
SBB reg, data	[reg] ←[reg] – data – CF	Same as SUB reg, data except this is a subtraction with borrow
SCAS BYTE or SCASB	FOR BYTE [AL] – [[DI]], flags are affected,[DI] ← [DI] ± 1	8- or 16-bit data addressed by [DI] in ES is subtracted from 8- or 16-bit data in AL or AX and flags are affected without affecting [AL] or [AX] or string data; ES cannot be overridden; if DF = 0, then DI is incremented by one for byte and two for word; if DF = 1, then DI is decremented by one for byte or decremented by two for word
SCAS WORD or SCASW	FOR WORD[AX] - [[DI]], flags are affected,[DI] ← [DI] ± 2	
SHL mem/reg, 1	SHIFT logical left once byte or word in mem/reg	Same as SAL mem/reg, 1
SHL mem/reg, CL	SHIFT logical left byte or word in mem/reg by the shift count in CL	Same as SAL mem/reg, CL except overflow is cleared to zero

Instructions	Interpretation	Comments
SHR mem/reg, 1	SHIFT right logical once byte or word in mem/reg	FOR BYTE  FOR WORD 
SHR mem/reg, CL	SHIFT right logical byte or word in mem/reg by [CL]	Operation same as SHR mem/reg, 1; however, shift count is specified in CL for shifts up to 255; zero and negative shifts are illegal
STC	CF ← 1	Set carry to one
STD	DF ← 1	Set direction flag to one
STI	IF ← 1	Set interrupt enable flag to one to enable maskable interrupts
STOS BYTE or STOSB	FOR BYTE [[DI]] ← [AL] [DI] ← [DI] ± 1	Store 8-bit data from AL or 16-bit data from AX into a memory location addressed by DI in segment ES; segment register ES cannot be overridden; if DF = 0, then DI is incremented by one for byte or incremented by two for word after the store
STOS WORD or STOSW	FOR WORD [[DI]] ← [AX],[DI] ← [DI] ± 2	
SUB mem/reg 1, mem/reg 2	[mem/reg 1] ← [mem/reg 1] - [mem/reg 2]	No memory-to-memory SUB permitted; all flags are affected; mem uses DS as the segment register
SUB mem, data	[mem] ← [mem] – data	Data can be 8- or 16-bit; mem uses DS as the segment register; all flags are affected
SUB reg, data	[reg] ← [reg] – data	Data can be 8- or 16-bit; all flags are affected
TEST mem/reg 1, mem/reg 2	[mem/reg 1]- [mem/reg 2], no result; flags are affected	No memory-to-memory TEST is allowed; no result is provided; all flags are affected with CF and OF cleared to zero; [mem], [reg 1] or [reg 2] can be 8-or 16-bit; no segment registers are allowed; mem uses DS as the segment register
TEST mem, data	[mem] - data, no result; flags are affected	Mem and data can be 8- or 16-bit; no result is provid ed;flagsareaffected with CF and OF cleared to zero; mem uses DS as the segment register
TEST reg, data	[reg]- data, no result; flags are affected	Reg and data can be 8- or 16-bit; no result is provided; all flags are affected with CF and OF cleared to zero; reg cannot be segment register;
WAIT	8086 enters wait state	Causes CPU to enter wait state if the 8086 TEST pin is high; while in wait state, the 8086 continues to check TEST pin for low; if TEST pin goes back to zero, the 8086 executes the next instruction; this feature can be used to synchronize the operation of 8086 to an event in external hardware
XCHG mem/reg, mem/reg	[mem] ↔ [reg]	reg and mem can be both 8- or 16-bit; mem uses DS as the segment register; reg cannot be segment register; no flags are affected; no mem to mem .
XCHG reg,reg	[reg] ↔ [reg]	reg can be 8-or 16-bit; reg cannot be segment register; no flags are affected

Instructions	Interpretation	Comments
XLAT	[AL] ← [AL] + [BX]	This instruction is useful for translating characters from one code such as ASCII to another such as EBCDIC; this is a no-operand instruction and is called an instruction with implied addressing mode; the instruction loads AL with the contents of a 20-bit physical address computed from DS, BX, and AL; this instruction can be used to read the elements in a table where BX can be loaded with a 16-bit value to point to the starting address (offset from DS) and AL can be loaded with the element number (0 being the first element number); no flags are affected; the XLAT instruction is equivalent to MOV AL, [AL] [BX]
XOR mem/reg 1, mem/reg 2	[mem/reg 1] ← [mem/reg 1] ⊕ [mem/reg 2]	No memory-to-memory operation is allowed; [mem] or [reg 1] or [reg 2] can be 8- or 16-bit; all flags are affected with CF and OF cleared to zero; mem uses DS as the segment register
XOR mem, data	[reg] ← [mem] ⊕ data	Data and mem can be 8- or 16-bit; mem uses DS as the segment register; mem cannot be segment register; all flags are affected with CF and OF cleared to zero
XOR reg, data	[reg] ← [reg] ⊕ data	Same as XOR mem, data.