APPENDIX C
INSTRUCTION SET DESCRIPTIONS

This appendix provides reference information for the 80C186 Modular Core family instruction set. Tables C-1 through C-3 define the variables used in Table C-4, which lists the instructions with their descriptions and operations.

Table C-1. Instruction Format Variables

Variable	Description
dest	A register or memory location that may contain data operated on by the instruction, and which receives (is replaced by) the result of the operation.
src	A register, memory location or immediate value that is used in the operation, but is not altered by the instruction
target	A label to which control is to be transferred directly, or a register or memory location whose content is the address of the location to which control is to be transferred indirectly.
disp8	A label to which control is to be conditionally transferred; must lie within –128 to +127 bytes of the first byte of the next instruction.
accum	Register AX for word transfers, AL for bytes.
port	An I/O port number; specified as an immediate value of 0–255, or register DX (which contains port number in range 0–64K).
src-string	Name of a string in memory that is addressed by register SI; used only to identify string as byte or word and specify segment override, if any. This string is used in the operation, but is not altered.
dest-string	Name of string in memory that is addressed by register DI; used only to identify string as byte or word. This string receives (is replaced by) the result of the operation.
count	Specifies number of bits to shift or rotate; written as immediate value 1 or register CL (which contains the count in the range 0–255).
interrupt-type	Immediate value of 0–255 identifying interrupt pointer number.
optional-pop-value	Number of bytes (0–64K, ordinarily an even number) to discard from the stack.
external-opcode	Immediate value (0–63) that is encoded in the instruction for use by an external processor.

Table C-2. Instruction Operands

Operand	Description
reg	An 8- or 16-bit general register.
reg16	An 16-bit general register.
seg-reg	A segment register.
accum	Register AX or AL
immed	A constant in the range 0–FFFFH.
immed8	A constant in the range 0–FFH.
mem	An 8- or 16-bit memory location.
mem16	A 16-bit memory location.
mem32	A 32-bit memory location.
src-table	Name of 256-byte translate table.
src-string	Name of string addressed by register SI.
dest-string	Name of string addressed by register DI.
short-label	A label within the –128 to +127 bytes of the end of the instruction.
near-label	A label in current code segment.
far-label	A label in another code segment.
near-proc	A procedure in current code segment.
far-proc	A procedure in another code segment.
memptr16	A word containing the offset of the location in the current code segment to which control is to be transferred.
memptr32	A doubleword containing the offset and the segment base address of the location in another code segment to which control is to be transferred.
regptr16	A 16-bit general register containing the offset of the location in the current code segment to which control is to be transferred.
repeat	A string instruction repeat prefix.

Table C-3. Flag Bit Functions

Name	Function
AF	Auxiliary Flag: Set on carry from or borrow to the low order four bits of AL; cleared otherwise.
CF	Carry Flag: Set on high-order bit carry or borrow; cleared otherwise.
DF	Direction Flag: Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
IF	Interrupt-enable Flag: When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
OF	Overflow Flag: Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise.
PF	Parity Flag: Set if low-order 8 bits of result contain an even number of 1 bits; cleared otherwise.
SF	Sign Flag: Set equal to high-order bit of result (0 if positive, 1 if negative).
TF	Single Step Flag: Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
ZF	Zero Flag: Set if result is zero; cleared otherwise.

Table C-4. Instruction Set

Name	Description	Operation	Flags Affected
AAA	**ASCII Adjust for Addition**: AAA Changes the contents of register AL to a valid unpacked decimal number; the high-order half-byte is zeroed. **Instruction Operands**: none	if ((AL) and 0FH) > 9 or (AF) = 1 then (AL) ← (AL) + 6 (AH) ← (AH) + 1 (AF) ← 1 (CF) ← (AF) (AL) ← (AL) and 0FH	AF ✓ CF ✓ DF – IF – OF ? PF ? SF ? TF – ZF ?
AAD	**ASCII Adjust for Division**: AAD Modifies the numerator in AL before dividing two valid unpacked decimal operands so that the quotient produced by the division will be a valid unpacked decimal number. AH must be zero for the subsequent DIV to produce the correct result. The quotient is returned in AL, and the remainder is returned in AH; both high-order half-bytes are zeroed. **Instruction Operands**: none	(AL) ← (AH) × 0AH + (AL) (AH) ← 0	AF ? CF ? DF – IF – OF ? PF ✓ SF ✓ TF – ZF ✓
AAM	**ASCII Adjust for Multiply**: AAM Corrects the result of a previous multiplication of two valid unpacked decimal operands. A valid 2-digit unpacked decimal number is derived from the content of AH and AL and is returned to AH and AL. The high-order half-bytes of the multiplied operands must have been 0H for AAM to produce a correct result. **Instruction Operands**: none	(AH) ← (AL) / 0AH (AL) ← (AL) % 0AH	AF ? CF ? DF – IF – OF ? PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
AAS	**ASCII Adjust for Subtraction**: AAS Corrects the result of a previous subtraction of two valid unpacked decimal operands (the destination operand must have been specified as register AL). Changes the content of AL to a valid unpacked decimal number; the high-order half-byte is zeroed. **Instruction Operands**: none	if ((AL) and 0FH) > 9 or (AF) = 1 then (AL) ← (AL) – 6 (AH) ← (AH) – 1 (AF) ← 1 (CF) ← (AF) (AL) ← (AL) and 0FH	AF ✓ CF ✓ DF – IF – OF ? PF ? SF ? TF – ZF ?
ADC	**Add with Carry**: ADC ***d**est, **s**rc* Sums the operands, which may be bytes or words, adds one if CF is set and replaces the destination operand with the result. Both operands may be signed or unsigned binary numbers (see AAA and DAA). Since ADC incorporates a carry from a previous operation, it can be used to write routines to add numbers longer than 16 bits. **Instruction Operands**: ADC reg, reg ADC reg, mem ADC mem, reg ADC reg, immed ADC mem, immed ADC accum, immed	if (CF) = 1 then (dest) ← (dest) + (src) + 1 else (dest) ← (dest) + (src)	AF ✓ CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected

ADD	**Addition**:
	ADD *dest, src*
	Sums two operands, which may be bytes or words, replaces the destination operand. Both operands may be signed or unsigned binary numbers (see AAA and DAA).
	Instruction Operands:
	ADD reg, reg
	ADD reg, mem
	ADD mem, reg
	ADD reg, immed
	ADD mem, immed
	ADD accum, immed
		(dest) ← (dest) + (src)	AF ✓
			CF ✓
			DF –
			IF –
			OF ✓
			PF ✓
			SF ✓
			TF –
			ZF ✓

AND	**And Logical**: AND ***d**est, **s**rc* Performs the logical "and" of the two operands (byte or word) and returns the result to the destination operand. A bit in the result is set if both corresponding bits of the original operands are set; otherwise the bit is cleared. **Instruction Operands**: AND reg, reg AND reg, mem AND mem, reg AND reg, immed AND mem, immed AND accum, immed	(dest) ← (dest) and (src) (CF) ← 0 (OF) ← 0	AF ? CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
BOUND	**Detect Value Out of Range**: BOUND *dest, src* Provides array bounds checking in hardware. The calculated array index is placed in one of the general purpose registers, and the upper and lower bounds of the array are placed in two consecutive memory locations. The contents of the register are compared with the memory location values, and if the register value is less than the first location or greater than the second memory location, a trap type 5 is generated. **Instruction Operands**: BOUND reg, mem	if ((dest) < (src) or (dest) > ((src) + 2) then (SP) ← (SP) – 2 ((SP) + 1 : (SP)) ← FLAGS (IF) ← 0 (TF) ← 0 (SP) ← (SP) – 2 ((SP) + 1 : (SP)) ← (CS) (CS) ← (1EH) (SP) ← (SP) – 2 ((SP) + 1 : (SP)) ← (IP) (IP) ← (1CH)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
CALL	**Call Procedure**: CALL ***p**rocedure-name* Activates an out-of-line procedure, saving information on the stack to permit a RET (return) instruction in the procedure to transfer control back to the instruction following the CALL. The assembler generates a different type of CALL instruction depending on whether the programmer has defined the procedure name as NEAR or FAR. **Instruction Operands**: CALL near-proc CALL far-proc CALL memptr16 CALL regptr16 CALL memptr32	if Inter-segment then (SP) ← (SP) – 2 ((SP) +1:(SP)) ← (CS) (CS) ← SEG (SP) ← (SP) – 2 ((SP) +1:(SP)) ← (IP) (IP) ← dest	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
CBW	**Convert Byte to Word**: CBW Extends the sign of the byte in register AL throughout register AH. Use to produce a double-length (word) dividend from a byte prior to performing byte division. **Instruction Operands**: none	if (AL) < 80H then (AH) ← 0 else (AH) ← FFH	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
CLC	**Clear Carry flag**: CLC Zeroes the carry flag (CF) and affects no other flags. Useful in conjunction with the rotate through carry left (RCL) and the rotate through carry right (RCR) instructions. **Instruction Operands**: none	(CF) ← 0	AF – CF ✓ DF – IF – OF – PF – SF – TF – ZF –
CLD	**Clear Direction flag**: CLD Zeroes the direction flag (DF) causing the string instructions to auto-increment the source index (SI) and/or destination index (DI) registers. **Instruction Operands**: none	(DF) ← 0	AF – CF – DF ✓ IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
CLI	**Clear Interrupt-enable Flag**: CLI Zeroes the interrupt-enable flag (IF). When the interrupt-enable flag is cleared, the 8086 and 8088 do not recognize an external interrupt request that appears on the INTR line; in other words maskable interrupts are disabled. A non-maskable interrupt appearing on NMI line, however, is honored, as is a software interrupt. **Instruction Operands**: none	(IF) ← 0	AF – CF – DF – IF ✓ OF – PF – SF – TF – ZF –
CMC	**Complement Carry Flag**: CMC Toggles complement carry flag (CF) to its opposite state and affects no other flags. **Instruction Operands**: none	if (CF) = 0 then (CF) ← 1 else (CF) ← 0	AF – CF ✓ DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected

CMP	**Compare**:
	CMP *dest, src*
	Subtracts the source from the destination, which may be bytes or words, but does not return the result. The operands are unchanged, but the flags are updated and can be tested by a subsequent conditional jump instruction. The comparison reflected in the flags is that of the destination to the source. If a CMP instruction is followed by a JG (jump if greater) instruction, for example, the jump is taken if the destination operand is greater than the source operand.
	Instruction Operands:
	CMP reg, reg
	CMP reg, mem
	CMP mem, reg
	CMP reg, immed
	CMP mem, immed
	CMP accum, immed
		(dest) – (src)	AF ✓
			CF ✓
			DF –
			IF –
			OF ✓
			PF ✓
			SF ✓
			TF –
			ZF ✓

CMPS	**Compare String**:
	CMPS *dest-string, src-string*
	Subtracts the destination byte or word from the source byte or word. The destination byte or word is addressed by the destination index (DI) register and the source byte or word is addresses by the source index (SI) register. CMPS updates the flags to reflect the relationship of the destination element to the source element but does not alter either operand and updates SI and DI to point to the next string element.
	Instruction Operands:
	CMP dest-string, src-string
	CMP (repeat) dest-string, src-string
		(dest-string) – (src-string)
		if
		(DF) = 0
		then
		(SI) ← (SI) + DELTA
		(DI) ← (DI) + DELTA
		else
		(SI) ← (SI) – DELTA
		(DI) ← (DI) – DELTA
			AF ✓
			CF ✓
			DF –
			IF –
			OF ✓
			PF ✓
			SF ✓
			TF –
			ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
CWD	**Convert Word to Doubleword**: CWD Extends the sign of the word in register AX throughout register DX. Use to produce a double-length (doubleword) dividend from a word prior to performing word division. **Instruction Operands**: none	if (AX) < 8000H then (DX) ← 0 else (DX) ← FFFFH	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
DAA	**Decimal Adjust for Addition**: DAA Corrects the result of previously adding two valid packed decimal operands (the destination operand must have been register AL). Changes the content of AL to a pair of valid packed decimal digits. **Instruction Operands**: none	if ((AL) and 0FH) > 9 or (AF) = 1 then (AL) ← (AL) + 6 (AF) ← 1 if (AL) > 9FH or (CF) = 1 then (AL) ← (AL) + 60H (CF) ← 1	AF ✓ CF ✓ DF – IF – OF ? PF ✓ SF ✓ TF – ZF ✓
DAS	**Decimal Adjust for Subtraction**: DAS Corrects the result of a previous subtraction of two valid packed decimal operands (the destination operand must have been specified as register AL). Changes the content of AL to a pair of valid packed decimal digits. **Instruction Operands**: none	if ((AL) and 0FH) > 9 or (AF) = 1 then (AL) ← (AL) – 6 (AF) ← 1 if (AL) > 9FH or (CF) = 1 then (AL) ← (AL) – 60H (CF) ← 1	AF ✓ CF ✓ DF – IF – OF ? PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
DEC	**Decrement**: DEC *dest* Subtracts one from the destination operand. The operand may be a byte or a word and is treated as an unsigned binary number (see AAA and DAA). **Instruction Operands:** DEC reg DEC mem	(dest) ← (dest) – 1	AF ✓ CF – DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
DIV	**Divide**: DIV *src* Performs an unsigned division of the accumulator (and its extension) by the source operand. If the source operand is a byte, it is divided into the two-byte dividend assumed to be in registers AL and AH. The byte quotient is returned in AL, and the byte remainder is returned in AH. If the source operand is a word, it is divided into the two-word dividend in registers AX and DX. The word quotient is returned in AX, and the word remainder is returned in DX. If the quotient exceeds the capacity of its destination register (FFH for byte source, FFFFH for word source), as when division by zero is attempted, a type 0 interrupt is generated, and the quotient and remainder are undefined. Nonintegral quotients are truncated to integers. **Instruction Operands**: DIV reg DIV mem	**When Source Operand is a Byte**: (temp) ← (byte-src) if (temp) / (AX) > FFH then (type 0 interrupt is generated) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← FLAGS (IF) ← 0 (TF) ← 0 (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (CS) (CS) ← (2) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (IP) (IP) ← (0) else (AL) ← (temp) / (AX) (AH) ← (temp) % (AX) **When Source Operand is a Word**: (temp) ← (word-src) if (temp) / (DX:AX) > FFFFH then (type 0 interrupt is generated) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← FLAGS (IF) ← 0 (TF) ← 0 (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (CS) (CS) ← (2) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (IP) (IP) ← (0) else (AX) ← (temp) / (DX:AX) (DX) ← (temp) % (DX:AX)	AF ? CF ? DF – IF – OF ? PF ? SF ? TF – ZF ?

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
ENTER	**Procedure Entry**: ENTER *locals, levels* Executes the calling sequence for a high-level language. It saves the current frame pointer in BP, copies the frame pointers from procedures below the current call (to allow access to local variables in these procedures) and allocates space on the stack for the local variables of the current procedure invocation. **Instruction Operands**: ENTER locals, level	(SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (BP) (FP) ← (SP) if level > 0 then repeat (level – 1) times (BP) ← (BP) – 2 (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (BP) end repeat (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (FP) end if (BP) ← (FP) (SP) ← (SP) – (locals)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
ESC	**Escape**: ESC Provides a mechanism by which other processors (coprocessors) may receive their instructions from the 8086 or 8088 instruction stream and make use of the 8086 or 8088 addressing modes. The CPU (8086 or 8088) does a no operation (NOP) for the ESC instruction other than to access a memory operand and place it on the bus. **Instruction Operands**: ESC immed, mem ESC immed, reg	if mod ≠ 11 then data bus ← (EA)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
HLT	**Halt**: HLT Causes the CPU to enter the halt state. The processor leaves the halt state upon activation of the RESET line, upon receipt of a non-maskable interrupt request on NMI, or upon receipt of a maskable interrupt request on INTR (if interrupts are enabled). **Instruction Operands**: none	None	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
IDIV	**Integer Divide**: IDIV *src* Performs a signed division of the accumulator (and its extension) by the source operand. If the source operand is a byte, it is divided into the double-length dividend assumed to be in registers AL and AH; the single-length quotient is returned in AL, and the single-length remainder is returned in AH. For byte integer division, the maximum positive quotient is +127 (7FH) and the minimum negative quotient is –127 (81H). If the source operand is a word, it is divided into the double-length dividend in registers AX and DX; the single-length quotient is returned in AX, and the single-length remainder is returned in DX. For word integer division, the maximum positive quotient is +32,767 (7FFFH) and the minimum negative quotient is –32,767 (8001H). If the quotient is positive and exceeds the maximum, or is negative and is less than the minimum, the quotient and remainder are undefined, and a type 0 interrupt is generated. In particular, this occurs if division by 0 is attempted. Nonintegral quotients are truncated (toward 0) to integers, and the remainder has the same sign as the dividend. **Instruction Operands:** IDIV reg IDIV mem	**When Source Operand is a Byte**: (temp) ← (byte-src) if (temp) / (AX) > 0 and (temp) / (AX) > 7FH or (temp) / (AX) < 0 and (temp) / (AX) < 0 – 7FH – 1 then (type 0 interrupt is generated) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← FLAGS (IF) ← 0 (TF) ← 0 (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (CS) (CS) ← (2) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (IP) (IP) ← (0) else (AL) ← (temp) / (AX) (AH) ← (temp) % (AX) **When Source Operand is a Word**: (temp) ← (word-src) if (temp) / (DX:AX) > 0 and (temp) / (DX:AX) > 7FFFH or (temp) / (DX:AX) < 0 and (temp) / (DX:AX) < 0 – 7FFFH – 1 then (type 0 interrupt is generated) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← FLAGS (IF) ← 0 (TF) ← 0 (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (CS) (CS) ← (2) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (IP) (IP) ← (0) else (AX) ← (temp) / (DX:AX) (DX) ← (temp) % (DX:AX)	AF ? CF ? DF – IF – OF ? PF ? SF ? TF – ZF ?

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
IMUL	**Integer Multiply**: IMUL *src* Performs a signed multiplication of the source operand and the accumulator. If the source is a byte, then it is multiplied by register AL, and the double-length result is returned in AH and AL. If the source is a word, then it is multiplied by register AX, and the double-length result is returned in registers DX and AX. If the upper half of the result (AH for byte source, DX for word source) is not the sign extension of the lower half of the result, CF and OF are set; otherwise they are cleared. When CF and OF are set, they indicate that AH or DX contains significant digits of the result. **Instruction Operands**: IMUL reg IMUL mem IMUL immed	**When Source Operand is a Byte**: (AX) ← (byte-src) × (AL) if (AH) = sign-extension of (AL) then (CF) ← 0 else (CF) ← 1 (OF) ← (CF) **When Source Operand is a Word**: (DX:AX) ← (word-src) × (AX) if (DX) = sign-extension of (AX) then (CF) ← 0 else (CF) ← 1 (OF) ← (CF)	AF ? CF ✓ DF – IF – OF ✓ PF ? SF ? TF – ZF ?
IN	**Input Byte or Word**: IN *accum, port* Transfers a byte or a word from an input port to the AL register or the AX register, respectively. The port number may be specified either with an immediate byte constant, allowing access to ports numbered 0 through 255, or with a number previously placed in the DX register, allowing variable access (by changing the value in DX) to ports numbered from 0 through 65,535. **Instruction Operands**: IN AL, immed8 IN AX, DX	**When Source Operand is a Byte**: (AL) ← (port) **When Source Operand is a Word**: (AX) ← (port)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
INC	**Increment**: INC *dest* Adds one to the destination operand. The operand may be byte or a word and is treated as an unsigned binary number (see AAA and DAA). **Instruction Operands**: INC reg INC mem	(dest) ← (dest) + 1	AF ✓ CF – DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓
INS	**In String**: INS *dest-string, port* Performs block input from an I/O port to memory. The port address is placed in the DX register. The memory address is placed in the DI register. This instruction uses the ES register (which cannot be overridden). After the data transfer takes place, the DI register increments or decrements, depending on the value of the direction flag (DF). The DI register changes by 1 for byte transfers or 2 for word transfers. **Instruction Operands**: INS dest-string, port INS (repeat) dest-string, port	(dest) ← (src)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
INT	**Interrupt**: INT *interrupt-type* Activates the interrupt procedure specified by the interrupt-type operand. Decrements the stack pointer by two, pushes the flags onto the stack, and clears the trap (TF) and interrupt-enable (IF) flags to disable single-step and maskable interrupts. The flags are stored in the format used by the PUSHF instruction. SP is decremented again by two, and the CS register is pushed onto the stack. The address of the interrupt pointer is calculated by multiplying interrupt-type by four; the second word of the interrupt pointer replaces CS. SP again is decremented by two, and IP is pushed onto the stack and is replaced by the first word of the interrupt pointer. If interrupt-type = 3, the assembler generates a short (1 byte) form of the instruction, known as the breakpoint interrupt. **Instruction Operands**: INT immed8	(SP) ← (SP) – 2 ((SP) + 1:(SP)) ← FLAGS (IF) ← 0 (TF) ← 0 (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (CS) (CS) ← (interrupt-type × 4 + 2) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (IP) (IP) ← (interrupt-type × 4)	AF – CF – DF – IF ✓ OF – PF – SF – TF ✓ ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
INTO	**Interrupt on Overflow**: INTO Generates a software interrupt if the overflow flag (OF) is set; otherwise control proceeds to the following instruction without activating an interrupt procedure. INTO addresses the target interrupt procedure (its type is 4) through the interrupt pointer at location 10H; it clears the TF and IF flags and otherwise operates like INT. INTO may be written following an arithmetic or logical operation to activate an interrupt procedure if overflow occurs. **Instruction Operands**: none	if (OF) = 1 then (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← FLAGS (IF) ← 0 (TF) ← 0 (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (CS) (CS) ← (12H) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (IP) (IP) ← (10H)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
IRET	**Interrupt Return**: IRET Transfers control back to the point of interruption by popping IP, CS, and the flags from the stack. IRET thus affects all flags by restoring them to previously saved values. IRET is used to exit any interrupt procedure, whether activated by hardware or software. **Instruction Operands**: none	(IP) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 (CS) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 FLAGS ← ((SP) + 1:(SP)) (SP) ← (SP) + 2	AF ✓ CF ✓ DF ✓ IF ✓ OF ✓ PF ✓ SF ✓ TF ✓ ZF ✓
JA JNBE	**Jump on Above**: **Jump on Not Below or Equal**: JA *disp8* JNBE *disp8* Transfers control to the target location if the tested condition ((CF=0) or (ZF=0)) is true. **Instruction Operands**: JA short-label JNBE short-label	if ((CF) = 0) or ((ZF) = 0) then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
JAE JNB	**Jump on Above or Equal**: **Jump on Not Below**: JAE *disp8* JNB *disp8* Transfers control to the target location if the tested condition (CF = 0) is true. **Instruction Operands**: JAE short-label JNB short-label	if (CF) = 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JB JNAE	**Jump on Below**: **Jump on Not Above or Equal**: JB *disp8* JNAE *disp8* Transfers control to the target location if the tested condition (CF = 1) is true. **Instruction Operands**: JB short-label JNAE short-label	if (CF) = 1 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JBE JNA	**Jump on Below or Equal**: **Jump on Not Above**: JBE *disp8* JNA *disp8* Transfers control to the target location if the tested condition ((C =1) or (ZF=1)) is true. **Instruction Operands**: JBE short-label JNA short-label	if ((CF) = 1) or ((ZF) = 1) then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JC	**Jump on Carry**: JC *disp8* Transfers control to the target location if the tested condition (CF=1) is true. **Instruction Operands**: JC short-label	if (CF) = 1 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
JCXZ	**Jump if CX Zero**: JCXZ *disp8* Transfers control to the target location if CX is 0. Useful at the beginning of a loop to bypass the loop if CX has a zero value, i.e., to execute the loop zero times. **Instruction Operands**: JCXZ short-label	if (CX) = 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JE JZ	**Jump on Equal**: **Jump on Zero**: JE *disp8* JZ *disp8* Transfers control to the target location if the condition tested (ZF = 1) is true. **Instruction Operands**: JE short-label JZ short-label	if (ZF) = 1 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JG JNLE	**Jump on Greater Than**: **Jump on Not Less Than or Equal**: JG *disp8* JNLE *disp8* Transfers control to the target location if the condition tested (SF = OF) and (ZF=0) is true. **Instruction Operands**: JG short-label JNLE short-label	if ((SF) = (OF)) and ((ZF) = 0) then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JGE JNL	**Jump on Greater Than or Equal**: **Jump on Not Less Than**: JGE *disp8* JNL *disp8* Transfers control to the target location if the condition tested (SF=OF) is true. **Instruction Operands**: JGE short-label JNL short-label	if (SF) = (OF) then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
JL JNGE	**Jump on Less Than**: **Jump on Not Greater Than or Equal**: JL *disp8* JNGE *disp8* Transfers control to the target location if the condition tested (SF≠OF) is true. **Instruction Operands**: JL short-label JNGE short-label	if (SF) ≠ (OF) then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JLE JNG	**Jump on Less Than or Equal**: **Jump on Not Greater Than**: JGE *disp8* JNL *disp8* Transfers control to the target location If the condition tested ((SF≠OF) or (ZF=0)) is true. **Instruction Operands**: JGE short-label JNL short-label	if ((SF) ≠ (OF)) or ((ZF) = 1) then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JMP	**Jump Unconditionally**: JMP *target* Transfers control to the target location. **Instruction Operands**: JMP short-label JMP near-label JMP far-label JMP memptr JMP regptr	if Inter-segment then (CS) ← SEG (IP) ← dest	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JNC	**Jump on Not Carry**: JNC *disp8* Transfers control to the target location if the tested condition (CF=0) is true. **Instruction Operands**: JNC short-label	if (CF) = 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
JNE JNZ	**Jump on Not Equal**: **Jump on Not Zero**: JNE *disp8* JNZ *disp8* Transfers control to the target location if the tested condition (ZF = 0) is true. **Instruction Operands**: JNE short-label JNZ short-label	if (ZF) = 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JNO	**Jump on Not Overflow**: JNO *disp8* Transfers control to the target location if the tested condition (OF = 0) is true. **Instruction Operands**: JNO short-label	if (OF) = 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JNS	**Jump on Not Sign**: JNS *disp8* Transfers control to the target location if the tested condition (SF = 0) is true. **Instruction Operands**: JNS short-label	if (SF) = 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JNP JPO	**Jump on Not Parity:** **Jump on Parity Odd**: JNO *disp8* JPO *disp8* Transfers control to the target location if the tested condition (PF=0) is true. **Instruction Operands**: JNO short-label JPO short-label	if (PF) = 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
JO	**Jump on Overflow**: JO *disp8* Transfers control to the target location if the tested condition (OF = 1) is true. **Instruction Operands**: JO short-label	if (OF) = 1 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JP JPE	**Jump on Parity**: **Jump on Parity Equal**: JP *disp8* JPE *disp8* Transfers control to the target location if the tested condition (PF = 1) is true. **Instruction Format**: JP short-label JPE short-label	if (PF) = 1 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
JS	**Jump on Sign**: JS *disp8* Transfers control to the target location if the tested condition (SF = 1) is true. **Instruction Format**: JS short-label	if (SF) = 1 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
LAHF	**Load Register AH From Flags**: LAHF Copies SF, ZF, AF, PF and CF (the 8080/8085 flags) into bits 7, 6, 4, 2 and 0, respectively, of register AH. The content of bits 5, 3, and 1 are undefined. LAHF is provided primarily for converting 8080/8085 assembly language programs to run on an 8086 or 8088. **Instruction Operands**: none	(AH) ← (SF):(ZF):X:(AF):X:(PF):X:(CF)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected

LDS	**Load Pointer Using DS**:
	LDS *dest, src*
	Transfers a 32-bit pointer variable from the source operand, which must be a memory operand, to the destination operand and register DS. The offset word of the pointer is transferred to the destination operand, which may be any 16-bit general register. The segment word of the pointer is transferred to register DS.
	Instruction Operands:
	LDS reg16, mem32
		(dest) ← (EA)
		(DS) ← (EA + 2)
			AF –
			CF –
			DF –
			IF –
			OF –
			PF –
			SF –
			TF –
			ZF –

LEA	**Load Effective Address**: LEA *dest, src* Transfers the offset of the source operand (rather than its value) to the destination operand. **Instruction Operands**: LEA reg16, mem16	(dest) ← EA	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
LEAVE	**Leave**: LEAVE Reverses the action of the most recent ENTER instruction. Collapses the last stack frame created. First, LEAVE copies the current BP to the stack pointer releasing the stack space allocated to the current procedure. Second, LEAVE pops the old value of BP from the stack, to return to the calling procedure's stack frame. A return (RET) instruction will remove arguments stacked by the calling procedure for use by the called procedure. **Instruction Operands**: none	(SP) ← (BP) (BP) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
LES	**Load Pointer Using ES**: LES *dest, src* Transfers a 32-bit pointer variable from the source operand to the destination operand and register ES. The offset word of the pointer is transferred to the destination operand. The segment word of the pointer is transferred to register ES. **Instruction Operands**: LES reg16, mem32	(dest) ← (EA) (ES) ← (EA + 2)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
LOCK	**Lock the Bus**: LOCK Causes the 8088 (configured in maximum mode) to assert its bus LOCK signal while the following instruction executes. The instruction most useful in this context is an exchange register with memory. The LOCK prefix may be combined with the segment override and/or REP prefixes. **Instruction Operands**: none	none	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
LODS	**Load String (Byte or Word)**: LODS *src-string* Transfers the byte or word string element addressed by SI to register AL or AX and updates SI to point to the next element in the string. This instruction is not ordinarily repeated since the accumulator would be overwritten by each repetition, and only the last element would be retained. **Instruction Operands**: LODS src-string LODS (repeat) src-string	**When Source Operand is a Byte**: (AL) ← (src-string) if (DF) = 0 then (SI) ← (SI) + DELTA else (SI) ← (SI) – DELTA **When Source Operand is a Word**: (AX) ← (src-string) if (DF) = 0 then (SI) ← (SI) + DELTA else (SI) ← (SI) – DELTA	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
LOOP	**Loop**: LOOP *disp8* Decrements CX by 1 and transfers control to the target location if CX is not 0; otherwise the instruction following LOOP is executed. **Instruction Operands**: LOOP short-label	(CX) ← (CX) – 1 if (CX) ≠ 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
LOOPE LOOPZ	**Loop While Equal**: **Loop While Zero**: LOOPE *disp8* LOOPZ *disp8* Decrements CX by 1 and transfers control is to the target location if CX is not 0 and if ZF is set; otherwise the next sequential instruction is executed. **Instruction Operands**: LOOPE short-label LOOPZ short-label	(CX) ← (CX) – 1 if (ZF) = 1 and (CX) ≠ 0 then (IP)←(IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
LOOPNE LOOPNZ	**Loop While Not Equal**: **Loop While Not Zero**: LOOPNE *disp8* LOOPNZ *disp8* Decrements CX by 1 and transfers control to the target location if CX is not 0 and if ZF is clear; otherwise the next sequential instruction is executed. **Instruction Operands**: LOOPNE short-label LOOPNZ short-label	(CX) ← (CX) – 1 if (ZF) = 0 and (CX) ≠ 0 then (IP) ← (IP) + disp8 (sign-ext to 16 bits)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

MOV	**Move (Byte or Word)**:
	MOV *dest, src*
	Transfers a byte or a word from the source operand to the destination operand.
	Instruction Operands:
	MOV mem, accum
	MOV accum, mem
	MOV reg, reg
	MOV reg, mem
	MOV mem, reg
	MOV reg, immed
	MOV mem, immed
	MOV seg-reg, reg16
	MOV seg-reg, mem16
	MOV reg16, seg-reg
	MOV mem16, seg-reg
		(dest)←(src)	AF –
			CF –
			DF –
			IF –
			OF –
			PF –
			SF –
			TF –
			ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected

MOVS	**Move String**:
	MOVS *dest-string, src-string*
	Transfers a byte or a word from the source string (addressed by SI) to the destination string (addressed by DI) and updates SI and DI to point to the next string element. When used in conjunction with REP, MOVS performs a memory-to-memory block transfer.
	Instruction Operands:
	MOVS dest-string, src-string
	MOVS (repeat) dest-string, src-string
		(dest-string) ← (src-string)	AF –
			CF –
			DF –
			IF –
			OF –
			PF –
			SF –
			TF –
			ZF –

MUL	**Multiply**: MUL *src* Performs an unsigned multiplication of the source operand and the accumulator. If the source is a byte, then it is multiplied by register AL, and the double-length result is returned in AH and AL. If the source operand is a word, then it is multiplied by register AX, and the double-length result is returned in registers DX and AX. The operands are treated as unsigned binary numbers (see AAM). If the upper half of the result (AH for byte source, DX for word source) is non-zero, CF and OF are set; otherwise they are cleared. **Instruction Operands**: MUL reg MUL mem	**When Source Operand is a Byte**: (AX) ← (AL) × (src) if (AH) = 0 then (CF) ← 0 else (CF) ← 1 (OF) ← (CF) **When Source Operand is a Word**: (DX:AX) ← (AX) × (src) if (DX) = 0 then (CF) ← 0 else (CF) ← 1 (OF) ← (CF)	AF ? CF ✓ DF – IF – OF ✓ PF ? SF ? TF – ZF ?

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
NEG	**Negate**: NEG *dest* Subtracts the destination operand, which may be a byte or a word, from 0 and returns the result to the destination. This forms the two's complement of the number, effectively reversing the sign of an integer. If the operand is zero, its sign is not changed. Attempting to negate a byte containing –128 or a word containing –32,768 causes no change to the operand and sets OF. **Instruction Operands**: NEG reg NEG mem	**When Source Operand is a Byte**: (dest) ← FFH – (dest) (dest) ← (dest) + 1 (affecting flags) **When Source Operand is a Word**: (dest) ← FFFFH – (dest) (dest) ← (dest) + 1 (affecting flags)	AF ✓ CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓
NOP	**No Operation**: NOP Causes the CPU to do nothing. **Instruction Operands**: none	None	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
NOT	**Logical Not**: NOT *dest* Inverts the bits (forms the one's complement) of the byte or word operand. **Instruction Operands**: NOT reg NOT mem	**When Source Operand is a Byte**: (dest) ← FFH – (dest) **When Source Operand is a Word**: (dest) ← FFFFH – (dest)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
OR	**Logical OR**: OR *dest,src* Performs the logical "inclusive or" of the two operands (bytes or words) and returns the result to the destination operand. A bit in the result is set if either or both corresponding bits in the original operands are set; otherwise the result bit is cleared. **Instruction Operands**: OR reg, reg OR reg, mem OR mem, reg OR accum, immed OR reg, immed OR mem, immed	(dest) ← (dest) or (src) (CF) ← 0 (OF) ← 0	AF ? CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓
OUT	**Output**: OUT *port, accumulator* Transfers a byte or a word from the AL register or the AX register, respectively, to an output port. The port number may be specified either with an immediate byte constant, allowing access to ports numbered 0 through 255, or with a number previously placed in register DX, allowing variable access (by changing the value in DX) to ports numbered from 0 through 65,535. **Instruction Operands**: OUT immed8, AL OUT DX, AX	(dest) ← (src)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
OUTS	**Out String**: OUTS ***port, src_string*** Performs block output from memory to an I/O port. The port address is placed in the DX register. The memory address is placed in the SI register. This instruction uses the DS segment register, but this may be changed with a segment override instruction. After the data transfer takes place, the pointer register (SI) increments or decrements, depending on the value of the direction flag (DF). The pointer register changes by 1 for byte transfers or 2 for word transfers. **Instruction Operands**: OUTS port, src_string OUTS (repeat) port, src_string	(dst) ← (src)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
POP	**Pop**: POP ***dest*** Transfers the word at the current top of stack (pointed to by SP) to the destination operand and then increments SP by two to point to the new top of stack. **Instruction Operands**: POP reg POP seg-reg (CS illegal) POP mem	(dest) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
POPA	**Pop All**: POPA Pops all data, pointer, and index registers off of the stack. The SP value popped is discarded. **Instruction Operands**: none	(DI) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 (SI) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 (BP) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 (BX) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 (DX) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 (CX) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 (AX) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
POPF	**Pop Flags**: POPF Transfers specific bits from the word at the current top of stack (pointed to by register SP) into the 8086/8088 flags, replacing whatever values the flags previously contained. SP is then incremented by two to point to the new top of stack. **Instruction Operands**: none	Flags ← ((SP) + 1:(SP)) (SP) ← (SP) + 2	AF ✓ CF ✓ DF ✓ IF ✓ OF ✓ PF ✓ SF ✓ TF ✓ ZF ✓
PUSH	**Push**: PUSH ***src*** Decrements SP by two and then transfers a word from the source operand to the top of stack now pointed to by SP. **Instruction Operands**: PUSH reg PUSH seg-reg (CS legal) PUSH mem	(SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (src)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
PUSHA	**Push All**: PUSHA Pushes all data, pointer, and index registers onto the stack. The order in which the registers are saved is: AX, CX, DX, BX, SP, BP, SI, and DI. The SP value pushed is the SP value before the first register (AX) is pushed. **Instruction Operands**: none	temp ← (SP) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (AX) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (CX) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (DX) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (BX) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (temp) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (BP) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (SI) (SP) ← (SP) – 2 ((SP) + 1:(SP)) ← (DI)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
PUSHF	**Push Flags**: PUSHF Decrements SP by two and then transfers all flags to the word at the top of stack pointed to by SP. **Instruction Operands**: none	(SP) ← (SP) – 2 ((SP) + 1:(SP)) ← Flags	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
RCL	**Rotate Through Carry Left**: RCL *dest, count* Rotates the bits in the byte or word destination operand to the left by the number of bits specified in the count operand. The carry flag (CF) is treated as "part of" the destination operand; that is, its value is rotated into the low-order bit of the destination, and itself is replaced by the high-order bit of the destination. **Instruction Operands**: RCL reg, n RCL mem, n RCL reg, CL RCL mem, CL	(temp) ← count do while (temp) ≠ 0 (tmpcf) ← (CF) (CF) ← high-order bit of (dest) (dest) ← (dest) × 2 + (tmpcf) (temp) ← (temp) – 1 if count = 1 then if high-order bit of (dest) ≠ (CF) then (OF) ← 1 else (OF) ← 0 else (OF) undefined	AF – CF ✓ DF – IF – OF ✓ PF – SF – TF – ZF –
RCR	**Rotate Through Carry Right**: RCR *dest, count* Operates exactly like RCL except that the bits are rotated right instead of left. **Instruction Operands**: RCR reg, n RCR mem, n RCR reg, CL RCR mem, CL	(temp) ← count do while (temp) ≠ 0 (tmpcf) ← (CF) (CF) ← low-order bit of (dest) (dest) ← (dest) / 2 high-order bit of (dest) ← (tmpcf) (temp) ← (temp) – 1 if count = 1 then if high-order bit of (dest) ≠ next-to-high-order bit of (dest) then (OF) ← 1 else (OF) ← 0 else (OF) undefined	AF – CF ✓ DF – IF – OF ✓ PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
REP REPE REPZ REPNE REPNZ	**Repeat**: **Repeat While Equal**: **Repeat While Zero**: **Repeat While Not Equal**: **Repeat While Not Zero**: Controls subsequent string instruction repetition. The different mnemonics are provided to improve program clarity. REP is used in conjunction with the MOVS (Move String) and STOS (Store String) instructions and is interpreted as "repeat while not end-of-string" (CX not 0). REPE and REPZ operate identically and are physically the same prefix byte as REP. These instructions are used with the CMPS (Compare String) and SCAS (Scan String) instructions and require ZF (posted by these instructions) to be set before initiating the next repetition. REPNE and REPNZ are mnemonics for the same prefix byte. These instructions function the same as REPE and REPZ except that the zero flag must be cleared or the repetition is terminated. ZF does not need to be initialized before executing the repeated string instruction. **Instruction Operands**: none	do while (CX) ≠ 0 service pending interrupts (if any) execute primitive string Operation in succeeding byte (CX) ← (CX) – 1 if primitive operation is CMPB, CMPW, SCAB, or SCAW and (ZF) ≠ 0 then exit from while loop	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
RET	**Return**: RET *optional-pop-value* Transfers control from a procedure back to the instruction following the CALL that activated the procedure. The assembler generates an intra-segment RET if the programmer has defined the procedure near, or an intersegment RET if the procedure has been defined as far. RET pops the word at the top of the stack (pointed to by register SP) into the instruction pointer and increments SP by two. If RET is intersegment, the word at the new top of stack is popped into the CS register, and SP is again incremented by two. If an optional pop value has been specified, RET adds that value to SP. **Instruction Operands**: RET immed8	(IP) ← ((SP) = 1:(SP)) (SP) ← (SP) + 2 if inter-segment then (CS) ← ((SP) + 1:(SP)) (SP) ← (SP) + 2 if add immed8 to SP then (SP) ← (SP) + data	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
ROL	**Rotate Left**: ROL *dest, count* Rotates the destination byte or word left by the number of bits specified in the count operand. **Instruction Operands**: ROL reg, n ROL mem, n ROL reg, CL ROL mem CL	(temp) ← count do while (temp) ≠ 0 (CF) ← high-order bit of (dest) (dest) ← (dest) × 2 + (CF) (temp) ← (temp) – 1 if count = 1 then if high-order bit of (dest) ≠ (CF) then (OF) ← 1 else (OF) ← 0 else (OF) undefined	AF – CF ✓ DF – IF – OF ✓ PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
ROR	**Rotate Right**: ROR *dest, count* Operates similar to ROL except that the bits in the destination byte or word are rotated right instead of left. **Instruction Operands**: ROR reg, n ROR mem, n ROR reg, CL ROR mem, CL	(temp) ← count do while (temp) ≠ 0 (CF) ← low-order bit of (dest) (dest) ← (dest) / 2 high-order bit of (dest) ← (CF) (temp) ← (temp) – 1 if count = 1 then if high-order bit of (dest) ≠ next-to-high-order bit of (dest) then (OF) ← 1 else (OF) ← 0 else (OF) undefined	AF – CF ✓ DF – IF – OF ✓ PF – SF – TF – ZF –
SAHF	**Store Register AH Into Flags**: SAHF Transfers bits 7, 6, 4, 2, and 0 from register AH into SF, ZF, AF, PF, and CF, respectively, replacing whatever values these flags previously had. **Instruction Operands**: none	(SF):(ZF):X:(AF):X:(PF):X:(CF) ← (AH)	AF ✓ CF ✓ DF – IF – OF – PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
SHL SAL	**Shift Logical Left**: **Shift Arithmetic Left**: SHL *dest, count* SAL *dest, count* Shifts the destination byte or word left by the number of bits specified in the count operand. Zeros are shifted in on the right. If the sign bit retains its original value, then OF is cleared. **Instruction Operands**: SHL reg, n SAL reg, n SHL mem, n SAL mem, n SHL reg, CL SAL reg, CL SHL mem, CL SAL mem, CL	(temp) ← count do while (temp) ≠ 0 (CF) ← high-order bit of (dest) (dest) ← (dest) × 2 (temp) ← (temp) – 1 if count = 1 then if high-order bit of (dest) ≠ (CE) then (OF) ← 1 else (OF) ← 0 else (OF) undefined	AF ? CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓
SAR	**Shift Arithmetic Right**: SAR *dest, count* Shifts the bits in the destination operand (byte or word) to the right by the number of bits specified in the count operand. Bits equal to the original high-order (sign) bit are shifted in on the left, preserving the sign of the original value. Note that SAR does not produce the same result as the dividend of an "equivalent" IDIV instruction if the destination operand is negative and 1 bits are shifted out. For example, shifting –5 right by one bit yields –3, while integer division –5 by 2 yields –2. The difference in the instructions is that IDIV truncates all numbers toward zero, while SAR truncates positive numbers toward zero and negative numbers toward negative infinity. **Instruction Operands**: SAR reg, n SAR mem, n SAR reg, CL SAR mem, CL	(temp) ← count do while (temp) ≠ 0 (CF) ← low-order bit of (dest) (dest) ← (dest) / 2 (temp) ← (temp) – 1 if count = 1 then if high-order bit of (dest) ≠ next-to-high-order bit of (dest) then (OF) ← 1 else (OF) ← 0 else (OF) ← 0	AF ? CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓ the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
SBB	**Subtract With Borrow**: SBB *dest, src* Subtracts the source from the destination, subtracts one if CF is set, and returns the result to the destination operand. Both operands may be bytes or words. Both operands may be signed or unsigned binary numbers (see AAS and DAS) **Instruction Operands**: SBB reg, reg SBB reg, mem SBB mem, reg SBB accum, immed SBB reg, immed SBB mem, immed	if (CF) = 1 then (dest) = (dest) – (src) – 1 else (dest) ← (dest) – (src)	AF ✓ CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
SCAS	**Scan String**: SCAS *dest-string* Subtracts the destination string element (byte or word) addressed by DI from the content of AL (byte string) or AX (word string) and updates the flags, but does not alter the destination string or the accumulator. SCAS also updates DI to point to the next string element and AF, CF, OF, PF, SF and ZF to reflect the relationship of the scan value in AL/AX to the string element. If SCAS is prefixed with REPE or REPZ, the operation is interpreted as "scan while not end-of-string (CX not 0) and string-element = scan-value (ZF = 1)." This form may be used to scan for departure from a given value. If SCAS is prefixed with REPNE or REPNZ, the operation is interpreted as "scan while not end-of-string (CX not 0) and string-element is not equal to scan-value (ZF = 0)." **Instruction Operands**: SCAS dest-string SCAS (repeat) dest-string	**When Source Operand is a Byte**: (AL) – (byte-string) if (DF) = 0 then (DI) ← (DI) + DELTA else (DI) ← (DI) – DELTA **When Source Operand is a Word**: (AX) – (word-string) if (DF) = 0 then (DI) ← (DI) + DELTA else (DI) ← (DI) – DELTA	AF ✓ CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
SHR	**Shift Logical Right**: SHR *dest, src* Shifts the bits in the destination operand (byte or word) to the right by the number of bits specified in the count operand. Zeros are shifted in on the left. If the sign bit retains its original value, then OF is cleared. **Instruction Operands**: SHR reg, n SHR mem, n SHR reg, CL SHR mem, CL	(temp) ← count do while (temp) ≠ 0 (CF) ← low-order bit of (dest) (dest) ← (dest) / 2 (temp) ← (temp) – 1 if count = 1 then if high-order bit of (dest) ≠ next-to-high-order bit of (dest) then (OF) ← 1 else (OF) ← 0 else (OF) undefined	AF ? CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓
STC	**Set Carry Flag**: STC Sets CF to 1. **Instruction Operands**: none	(CF) ← 1	AF – CF ✓ DF – IF – OF – PF – SF – TF – ZF –
STD	**Set Direction Flag**: STD Sets DF to 1 causing the string instructions to auto-decrement the SI and/or DI index registers. **Instruction Operands**: none	(DF) ← 1	AF – CF – DF ✓ IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
STI	**Set Interrupt-enable Flag**: STI Sets IF to 1, enabling processor recognition of maskable interrupt requests appearing on the INTR line. Note however, that a pending interrupt will not actually be recognized until the instruction following STI has executed. **Instruction Operands**: none	(IF) ← 1	AF – CF – DF – IF ✓ OF – PF – SF – TF – ZF –
STOS	**Store (Byte or Word) String**: STOS *dest-string* Transfers a byte or word from register AL or AX to the string element addressed by DI and updates DI to point to the next location in the string. As a repeated operation. **Instruction Operands**: STOS dest-string STOS (repeat) dest-string	**When Source Operand is a Byte**: (DEST) ← (AL) if (DF) = 0 then (DI) ← (DI) + DELTA else (DI) ← (DI) – DELTA **When Source Operand is a Word**: (DEST) ← (AX) if (DF) = 0 then (DI) ← (DI) + DELTA else (DI) ← (DI) – DELTA	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected

SUB	**Subtract**:
	SUB *dest, src*
	The source operand is subtracted from the destination operand, and the result replaces the destination operand. The operands may be bytes or words. Both operands may be signed or unsigned binary numbers (see AAS and DAS).
	Instruction Operands:
	SUB reg, reg
	SUB reg, mem
	SUB mem, reg
	SUB accum, immed
	SUB reg, immed
	SUB mem, immed
		(dest) ← (dest) – (src)	AF ✓
			CF ✓
			DF –
			IF –
			OF ✓
			PF ✓
			SF ✓
			TF –
			ZF ✓

TEST	**Test**: TEST dest, src Performs the logical "and" of the two operands (bytes or words), updates the flags, but does not return the result, i.e., neither operand is changed. If a TEST instruction is followed by a JNZ (jump if not zero) instruction, the jump will be taken if there are any corresponding one bits in both operands. **Instruction Operands**: TEST reg, reg TEST reg, mem TEST accum, immed TEST reg, immed TEST mem, immed	(dest) and (src) (CF) ← 0 (OF) ← 0	AF ? CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
WAIT	**Wait**: WAIT Causes the CPU to enter the wait state while its test line is not active. **Instruction Operands**: none	None	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
XCHG	**Exchange**: XCHG ***d****est, src* Switches the contents of the source and destination operands (bytes or **w**ords). When used in conjunction with the LOCK prefix, XCHG can test and set a semaphore that controls access to a resource shared by multiple processors. **Instruction Operands**: XCHG accum, reg XCHG mem, reg XCHG reg, reg	(temp) ← (dest) (dest) ← (src) (src) ← (temp)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Table C-4. Instruction Set (Continued)

Name	Description	Operation	Flags Affected
XLAT	**Translate**: XLAT ***t****ranslate-table* Replaces a byte in the AL register with a byte from a 256-byte, user-coded translation table. Register BX is assumed to point to the beginning of the table. The byte in AL is used as an index into the table and is replaced by the byte at the offset in the table corresponding to AL's binary value. The first byte in the table has an offset of 0. For example, if AL contains 5H, and the sixth element of the translation table contains 33H, then AL will contain 33H following the instruction. XLAT is useful for translating characters from one code to another, the classic example being ASCII to EBCDIC or the reverse. **Instruction Operands**: XLAT src-table	AL ← ((BX) + (AL))	AF – CF – DF – IF – OF – PF – SF – TF – ZF –
XOR	**Exclusive Or**: XOR ***d****est, src* Performs the logical "exclusive or" of the two operands and returns the result to the destination operand. A bit in the result is set if the corresponding bits of the original operands contain opposite values (one is set, the other is cleared); otherwise the result bit is cleared. **Instruction Operands**: XOR reg, reg XOR reg, mem XOR mem, reg XOR accum, immed XOR reg, immed XOR mem, immed	(dest) ← (dest) xor (src) (CF) ← 0 (OF) ← 0	AF ? CF ✓ DF – IF – OF ✓ PF ✓ SF ✓ TF – ZF ✓

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed