Intel 80x86 Instruction Set Summary

This document contains a description of all 80x86 instructions not including math coprocessor instructions. Each instruction is described briefly. All operand forms valid with each instruction are shown and some syntax examples are given. The flags affected by each instruction are shown in the upper right corner for the description for the instruction. The effect of instructions on the flags are indicated as follows:

-	No change
?	Unpredictable change
*	Predictable change
1	Set to 1
0	Set to 0

AAA

ASCII adjust AX after addition

O	D	I	T	S	Z	A	P	C
?	-	-	-	?	?	*	?	*

Description:
This instruction is used to adjust the value in AL into the correct range after an unpacked BCD addition has occurred. After executing an ADD or ADC instruction that leaves a single BCD or ASCII digit in AL, execute AAA to produce a valid BCD result.
If the value in AL indicates that a decimal overflow occurred, the BCD digit is forced into the legal range, and AH is incremented.

Example	***Function***
AAA	Corrects the result of an ASCII addition

AAD

ASCII adjust AX before division

O	D	I	T	S	Z	A	P	C
?	-	-	-	*	*	?	*	?

Description:
This instruction is used before BCD division. Before execution, the AL register should contain a single unpacked BCD digit. The AH register should hold the next higher order BCD digit. After executing the AAD instruction, AX contains the binary equivalent of the two BCD digits.

Example	***Function***
AAD	Corrects AX before an ASCII division

AAM

ASCII adjust AX after multiplication

O	D	I	T	S	Z	A	P	C
?	-	-	-	*	*	?	*	?

Description:
The AAM instruction converts the result of a single digit BCD multiplication in the AX register into two unpacked BCD digits. The high order digit will be in AH and the low order digit in AL.

Example	***Function***
AAM	Corrects AX after an ASCII multiplication

AAS	ASCII adjust AX after subtraction	`O D I T S Z A P C`
		`? - - - ? ? * ? *`

Description: This instruction ensures that a BCD subtraction results in a valid BCD digit. After executing a SUB or SBB instruction that leaves a single BCD digit in AL, execute AAS to produce a valid BCD result. If the value in AL produces a decimal borrow, the BCD is forced into the legal range and 1 is subtracted from AH.		
Example	***Function***	
AAD	Corrects AX after an ASCII subtraction	

ADC	Add with carry	`O D I T S Z A P C`
		`* - - - * * * * *`

Description: This instruction adds the contents of the source and destination together, increments the result if the carry flag is set and stores the result in the destination. The operands must be the same size. If the operands are signed integers, OF flag will indicate an invalid result. If the operands are unsigned, the CF will be set if a carry occurred out of the high bit of the result.		
General Forms	***Function***	
ADC *reg,idata* ADC *mem,idata* ADC *regd,regs* ADC *reg,mem* ADC *mem,reg*	Add *idata* with carry to *reg* Add *idata* with carry to memory location *mem* Add *regs* with carry to *regd* Add contents of memory location *mem* with carry to *reg* Add contents of *reg* with carry to memory location *mem*	
Examples		
ADC AL,BL ADC DATA1,AX ADC BL,[DI] ADC EAX,1 ADC BYTE PTR [BX],2	Adds BL with carry to AL Adds AX with carry to memory location DS:DATA1 Adds memory location DS:DI with carry to BL Adds 1 with carry to EAX	

ADD	Add	`O D I T S Z A P C`
		`* - - - * * * * *`

Description: This instruction adds the contents of the source and destination and stores the result in the destination. The operands must be the same size. If the operands are signed integers, the OF flag indicates an invalid result. If the operands are unsigned, the CF flag indicates a carry out of the high bit of the result. If the operands are BCD digits, the AF flag indicates a decimal carry.		
General Forms	***Function***	
ADD *reg,idata* ADD *mem,idata* ADD *regd,regs* ADD *reg,mem* ADD *mem,reg*	Add *idata* to *reg* Add *idata* to memory location *mem* Add *regs* to *regd* Add contents of memory location *mem* to *reg* Add contents of *reg* to memory location *mem*	
Examples	***Function***	
ADD CL,BL ADD DATA2,DL ADD CL,[SI] ADD ECX,1 ADD WORD PTR [BX],2	Adds BL to CL Adds DL to memory location DS:DATA2 Adds contents of memory location DS:SI to CL Adds 1 to ECX Adds 2 to word in memory location DS:BX	

AND	Logical AND	O D I T S Z A P C 0 - - - * * ? * 0

Description:
This instruction performs a bit by bit logical AND operation on the contents of the source and destination, and stores the result in the destination.

General Forms	***Function***
AND *reg,idata*	Logical AND *reg* with *idata*
AND *mem,idata*	Logical AND contents of memory location *mem* with *idata*
AND *regd,regs*	Logical AND *regd* with *regs*
AND *reg,mem*	Logical AND *reg* with contents of memory location *mem*
AND *mem,reg*	Logical AND contents of memory location *mem* with *reg*

Examples	
AND AL,07FH	Clears the high order bit of AL
AND DATA3,DX	Logical AND of word at memory location DS:DATA3 with DX
AND CL,ES:[DI+2]	Logical AND of byte at memory location ES:DI+2 with CL
AND BX,CX	Logical AND of BX with CX
AND AX,MASK[SI]	Logical AND of word at memory location DS:MASK+SI with AX

ARPL	Adjust requested privilege level (80286 or later)	O D I T S Z A P C - - - - - * - - -

Description:
This instruction is used to modify a selector’s requested privilege level. Both the source and destination operands must be valid selectors.
If the RPL of the destination operand is numerically less (higher privilege level) than that of the source operand, the destination selector’s RPL is changed to match that of the source operand, and ZF is set to 1. If the destination operand is numerically higher (less privileged), then it is not modified and ZF is set to 0.

General Forms	***Function***
ARPL *regd,regs*	Adjust RPL of *regd* down to agree with *regs*
ARPL *mem,reg*	Adjust RPL of selector in location *mem* down to agree with *reg*

Examples	
ARPL AX,BX	Privilege level of selector in AX adjusted to agree with BX
ARPL MEM,CX	Privilege level of selector in DS:MEM adjusted to agree with CX

BOUND	Check array bounds (80186 or later)	O D I T S Z A P C - - - - - - - - -

Description:
The source operand specifies the location of a memory table giving the array bounds (lower bound, followed by upper bound). The destination operand is an array index. If the source operand is not within the bounds specified by the destination operand, then an INT 5 is executed.

General Forms	***Function***
BOUND *reg,mem*	Check that array index in *reg* is within limits specified at *mem*

Example	
BOUND AX, BETS	If AX is not in bounds of DS:BETS issue INT 5 exception

BSF	Bit scan forward (80386 or later)	`O D I T S Z A P C`
		`- - - - - * - - -`

Description: This instruction scans the source operand starting at bit position 0. It writes the bit position of the first 1 bit found to the destination operand. If the source operand is 0, the zero flag is set and the contents of the destination operand are undefined.		
General Forms	***Function***	
BSF *regd,regs* BSF *reg,mem*	Scan *regd* for 1 bit. *Regs* gets index of first 1 bit Scan memory location *mem* for 1 bit. *Reg* gets index of first 1 bit	
Examples		
BSF AX,BX BSF EAX,DAN	Scans BX from bit 0, AX gets the position of the first 1 bit in BX, the Z flag set if no bits in BX are set Scans DWORD at DS:DAN from bit 0, EAX gets the position of the first 1 bit, Z flag set if no bits in DS:DAN are set.	

BSR	Bit scan reverse (80386 or later)	`O D I T S Z A P C`
		`- - - - - * - - -`

Description: This instruction scans the source operand starting at the highest bit position. It writes the bit position of the first 1 bit found to the destination operand. If the source operand is 0, the zero flag is set and the contents of the destination operand are undefined.		
General Forms	***Function***	
BSR *regd,regs* BSR *reg,mem*	Scan *regd* for 1 bit. *Regs* gets index of first 1 bit. Scan memory location *mem* for 1 bit. *Reg* gets index of first 1 bit	
Examples		
BSR AX,BX BSR EAX,MEM	Scans BX from bit 15, AX gets the position of the first 1 bit in BX, the Z flag set if no bits in BX are set. Scans DWORD at DS:MEM from bit 31, EAX gets the position of the first 1 bit, the Z flag is set if no bits in DS:MEM are set.	

BSWAP	Byte swap (80486 or later)	`O D I T S Z A P C`
		`- - - - - - - - -`

Description: This instruction converts the value in the specified 32 bit register from little endian format to big endian format. Byte 0 and byte 3 are exchanged, and byte 1 and byte 2 are exchanged.		
General Forms	***Function***	
BSWAP reg32	Swap the byte order of the specified register	
Example		
BSWAP EAX	Converts EAX from little-endian to big-endian format	

BT	Bit test (80386 or later)	O D I T S Z A P C - - - - - - - - *
Description: This instruction tests the bit specified by the operands and places its value into the carry flag. The source operand contains an index into the bit array specified by the destination. The state of the specified bit is copied into the carry flag. This instruction does not accept BYTE operands. It works only on 16 or 32 bit values. Do not use this instruction with BYTE oriented memory mapped i/o registers.		
General Forms	***Function***	
BT *reg,idata* BT *mem,idata* BT *regd,regs* BT *mem,reg*	Copy the bit specified by *idata* from *reg* to the carry flag Copy the bit specified by *idata* from memory location *mem* to CF Copy the bit specified by *regs* from *regd* to the carry flag Copy the bit specified by *reg* from memory location *mem* to CF	
Examples		
BT EBX,4 BT MEM,1 BT EBX,ECX BT MEM,AX	Test bit 4 of EBX, C <= bit 4 Test bit 1 of memory location DS:MEM, C <= bit 1 Test bit ECX of EBX, C <= bit ECX Test bit AX of memory location DS:MEM, C <= bit AX	

BTC	Bit test and complement (80386 or later)	O D I T S Z A P C - - - - - - - - *
Description: This instruction tests the bit specified by the operands and places its value into the carry flag. The specified bit is then complemented. The source operand contains an index into the bit array specified by the destination operand. The state of the selected bit is copied to the carry flag, and the bit is complemented. The carry flag will contain the state of the bit before it is complemented. This instruction does not work on byte operands. It can only be used on 16 or 32 bit operands. Do not use this instruction with memory mapped i/o devices that are 8 bits wide.		
General Forms	***Function***	
BTC *reg,idata* BTC *mem,idata* BTC *regd,regs* BTC *mem,reg*		
Examples		
BTC EBX,4 BTC MEM,1 BTC EBX,ECX BTC MEM,AX	Test and complement bit 4 of EBX, C <= bit 4 Test and complement bit 1 or EBX, C <= bit 1 Test and complement bit ECX of EBX, C <= bit ECX Test and complement bit AX of location DS:MEM, C <= bit AX	

BTR	Bit test and reset (80386 or later)	O D I T S Z A P C - - - - - - - - *

Description:
This instruction tests the bit specified by the operands and places its value into the carry flag. The selected bit is then reset. The source operand contains an index into the bit array specified by the destination operand. The state of the selected bit is copied to the carry flag, and the bit is then reset. The carry flag will contain the state of the bit before it is reset. This instruction does not work on byte operands. It can only be used on 16 or 32 bit operands. Do not use this instruction with memory mapped i/o devices that are 8 bits wide.

General Forms	***Function***
BTR *reg,idata* BTR *mem,idata* BTR *regd,regs* BTR *mem,reg*	

Examples	
BTR EBX,4	Test and reset bit 4 of EBX, C <= bit 4
BTR MEM,1	Test and reset bit 1 or EBX, C <= bit 1
BTR EBX,ECX	Test and reset bit ECX of EBX, C <= bit ECX
BTR MEM,AX	Test and reset bit AX of location DS:MEM, C <= bit AX

BTS	Bit test and set (80386 or later)	O D I T S Z A P C - - - - - - - - *

Description:
This instruction tests the bit specified by the operands then places its value into the carry flag. The selected bit is then set. The source operand contains the index of the bit array specified by the destination operand. The state of the selected bit is copied to the carry flag, and the bit is then set. The carry flag will contain the state of the bit before it is set. This instruction does not work on byte operands. It can only be used on 16 or 32 bit operands. Do not use this instruction with memory mapped i/o devices that are 8 bits wide.

General Forms	***Function***
BTS *reg,idata* BTS *mem,idata* BTS *regd,regs* BTS *mem,reg*	

Examples	
BTS EBX,4	Test and set bit 4 of EBX, C <= bit 4
BTS MEM,1	Test and set bit 1 or EBX, C <= bit 1
BTS EBX,ECX	Test and set bit ECX of EBX, C <= bit ECX
BTS MEM,AX	Test and set bit AX of location DS:MEM, C <= bit AX

CALL — Call far procedure (subroutine)

O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

Description:
The far procedure call saves the current code segment selector and the address of the next instruction (IP or EIP) onto the stack. Control then transfers to the destination specified by the operand.

General Forms	***Function***
CALL idata	Push CS:IP and then load CS:IP with value specified in idata
CALLl mem	Push CS:IP and then load CS:IP with value contained in mem

Examples	
CALL SUBR1	Call procedure SUBR1
CALL FAR PTR JTAB[SI]	Call the procedure whose address is stored in memory location DS:[JTAB+SI]
CALL MEM	Call FAR to the procedure whose address is stored in memory location DS:MEM. This assumes that MEM is declared as a DWORD

CALL — Call near procedure (subroutine)

O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

Description:
The near procedure call pushes the address of the next instruction (IP or EIP) onto the stack and then transfers control to the location specified by the operand. If the operand is an immediate value, the destination is relative to the current location. If the operand is a memory address or register, the subroutine address is taken indirectly from the operand.

General Forms	***Function***
CALL offset	Push IP and then add offset to IP
CALL mem	Push IP and then load IP with the contents of mem
CALL reg	Push IP and then load IP with the contents of reg

Examples	
CALL SUBR1	Call procedure SUBR1
CALL CX	Call procedure whose address is in CX
CALL NEAR PTR JTAB[SI]	Call NEAR to the procedure whose address is stored in memory location DS:[JTAB+SI]
CALL MEM	Call NEAR to the procedure whose address is stored in memory location DS:MEM. This assumes that MEM is declared as a WORD.

CBW — Convert BYTE to WORD

O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

Description:
This instruction sign extends the byte in AL into AX

Example	***Function***
CBW	Sign extend AL into AX

CDQ — Convert DWORD to QWORD
(80386 or later)

O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

Description:
This instruction sign extends the 32 bit value in EAX into EDX.

Example	***Function***
CDQ	Sign extend EAX into EDX

CLC	Clear carry flag	`O D I T S Z A P C`
		`- - - - - - - - 0`

Description: This instruction will set the carry flag to 0.		
Example	***Function***	
CLC	Set carry flag to 0	

CLD	Clear direction flag	`O D I T S Z A P C`
		`- 0 - - - - - - -`

Description: This instruction will set the direction flag to 0. This will cause string instructions to increment the pointer registers.		
Example	***Function***	
CLD	Set direction flag to 0 (string instructions increment index registers)	

CLI	Clear interrupt flag	`O D I T S Z A P C`
		`- - 0 - - - - - -`

Description: This instruction will set the interrupt enable flag to 0. This causes interrupts to be disabled.		
Example	***Function***	
CLI	Interrupt flag set to 0 (interrupts disabled)	

CLTS	Clear the task-switched flag
	(80286 or later)
		`O D I T S Z A P C`
		`- - - - - - - - -`

Description: This instruction sets the task switched bit (TS) in the MSW (80286) or CR0 register (80386 or later) to 0.		
Example	***Function***	
CLTS	Task-switched flag in MSW or CR0 set to 0	

CMC	Complement carry	`O D I T S Z A P C`
		`- - - - - - - - *`

Description: This instruction complements the state of the carry flag in the flags register. If the flag is 1, it will be set to 0. If the flag is 0, it will be set to 1		
Example	***Function***	
CMC	Complements (inverts) the carry flag	

CMP

Compare operands

```
O D I T S Z A P C
* - - - * * * * *
```

Description:
This instruction compares the two operands. The contents of the source operand is subtracted from the contents of the destination operand and the flags are set to correspond to the result of the subtraction. The result of the subtraction is not stored.

General Forms	***Function***
CMP *reg,idata*	Subtract *idata* from *reg* and set the flags accordingly
CMP *mem,idata*	Subtract *idata* from the contents of *mem* and set the flags accordingly
CMP *regd,regs*	Subtract *regs* from *regd* and set the flags accordingly
CMP *reg,mem*	Subtract the contents of *mem* from *reg* and set the flags accordingly
CMP *mem,reg*	Subtract *reg* from the contents of *mem* and set the flags accordingly

Examples	
CMP BL,CL	Compare BL with CL
CMP MEM,AX	Compare word at memory location DS:MEM with AX
CMP ECX,DAY1	Compare ECX with the DWORD at memory location DS:DAY1
CMP AL,2	Compare AL with the constant 2
CMP DATA2,1	Compare the contents of memory location DS:DATA2 with 1

CMPS

Compare strings

```
O D I T S Z A P C
* - - - * * * * *
```

Description:
This instruction subtracts the memory location specified by DS:SI (or DS:ESI) from the operand specified by ES:DI (or ES:EDI), setting the flags and discarding the result, was with the CMP instruction. The size of the operand can be either a BYTE, WORD, or DWORD. Following the comparison, SI (ESI) and DI (EDI) will be either incremented or decremented, depending on the state of the direction flag, by an amount appropriate to the size of the operands.

Example	***Function***
CMPSB	Compare memory byte at DS:SI with memory byte ES:DI
CMPSW	Compare memory word at DS:SI with memory word at ES:DI
CMPSD	Compare memory dword at DS:SI with memory dword at ES:DI

CMPXCHG

Compare and exchange
(80486 and later)

```
O D I T S Z A P C
* - - - * * * * *
```

Description:

Example	***Function***
CMPXCHG CX,BX	Compare CX with AX, if equal CX<=BX else AX<=BX
CMPXCHG MEM,DX	Compare DS:MEM with AX, if equal MEM<=DX else AX<=DX

CMPXCHG8B

Compare and exchange 8 bytes
(80486 and later)

```
O D I T S Z A P C
- - - - - * - - -
```

Description:

Example	***Function***
CMPXCHG8B MEM	Compare DS:MEM with EDX:EAX, if equal MEM<=ECX:EBX else MEM<=EDX:EAX

CPUID	Get CPU identification
	(Pentium or later)
		`O D I T S Z A P C`
		`- - - - - * - - -`

Description:		
Example	***Function***	
CPUID	EAX <= CPU identification information	

CWD	Convert WORD to DWORD	`O D I T S Z A P C`
		`- - - - - - - - -`

Description: This instruction sign extends the word in AX into DX:AX		
Example	***Function***	
CWD	Sign extend AX into DX:AX	

CWDE	Convert WORD to DWORD
	(80386 or later)
		`O D I T S Z A P C`
		`- - - - - - - - -`

Description: This instruction will sign extend the word in AX into EAX.		
Example	***Function***	
CWDE	Sign extend AX into EAX	

DAA	Decimal adjust after addition	`O D I T S Z A P C`
		`? - - - * * * * *`

Description: This instruction is used following an addition on packed decimal data to ensure that the value in AL contains a correct decimal result.		
Example	***Function***	
DAA	Adjust the contents of AL after BCD addition	

DAS	Decimal adjust after subtraction	`O D I T S Z A P C`
		`? - - - * * * * *`

Description: This instruction is used following a subtraction of packed decimal data to ensure that the value in AL contains a correct decimal result.		
Example	***Function***	
DAS	Adjust the contents of AL after BCD subtraction	

DEC — Decrement

```
O D I T S Z A P C
* - - - * * * * -
```

Description:
This instruction subtracts 1 from the specified operand. This instruction does not affect the carry flag, but affects all other condition code flags.

General Forms	***Function***
DEC *reg*	Subtract 1 from *reg*
DEC *mem*	Subtract 1 from *mem*

Example	
DEC BH	Subtract 1 from BH
DEC CX	Subtract 1 from CX
DEC MEM[BX]	Subtract 1 from the contents of memory location DS:MEM+BX
DEC EDX	Subtract 1 from EDX

DIV — Divide (unsigned)

```
O D I T S Z A P C
? - - - ? ? ? ? ?
```

Description:
This instruction performs an unsigned division of the value in the accumulator register or register pair by the specified operand, storing the quotient in the low part of the accumulator and the remainder in the high part of the accumulator. For BYTE operands, the accumulator is AX, with the resulting quotient in AL and the remainder in AH. For WORD operands, the accumulator is DX:AX, with the resulting quotient in AX and the remainder in DX. For DWORD operands, the accumulator is EDX:EAX, with the resulting quotient in EAX and the remainder in EDX.

General Forms	***Function***
DIV BH	Divide AX by BH, AH<=remainder, AL<=quotient
DIV CX	Divide DX:AX by CX; DX<=remainder, AX<=quotient
DIV ESI	Divide EDX:EAX by ESI; EDX<=remainder, EAX<=quotient

Example	
DIV BH	Divide AX by BH, AH<=remainder, AL<=quotient
DIV CX	Divide DX:AX by CX; DX<=remainder, AX<=quotient
DIV ESI	Divide EDX:EAX by ESI; EDX<=remainder, EAX<=quotient

ENTER — Create a stack frame
(80186 or later)

```
O D I T S Z A P C
- - - - - - - - -
```

Description:
This instruction will set up a stack frame reserving space for local variables for the procedure. When the second operand is greater than 0, the pointers to previous stack frames are pushed onto the stack to allow addressing of stack resident variables whose scopes contain the scope of the current procedure. The ENTER *n*,0 instruction is equivalent to this instruction sequence:

```
    PUSH    BP
    MOV      BP,SP
    SUB      SP,n
```

Examples	***Function***
ENTER 16,0	Create a stack frame of 16 bytes for level 0
ENTER 32,1	Create a stack frame of 32 bytes for level 1

HLT — Halt

O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

Description:
This instruction stops the processor. No other instructions will execute until the processor is brought out of the halt state by a reset or an interrupt. An NMI or reset will always bring the processor out of the halt state. If the halt state is entered with maskable interrupts disabled (IF = 0) , then these interrupts will not be acknowledged or bring the processor out of the halt state. Execution will resume at the instruction following the HLT instruction after the interrupt service routine is completed.

Example	***Function***
HLT	Halts all processing until a reset or interrupt occurs

IDIV — Divide (signed)

O	D	I	T	S	Z	A	P	C
?	-	-	-	?	?	?	?	?

Description:
This instruction performs a signed division of the value in the accumulator register or register pair by the specified operand, storing the quotient in the low part of the accumulator and the remainder in the high part of the accumulator. For BYTE operands, the accumulator is AX, with the resulting quotient in AL and the remainder in AH. For WORD operands, the accumulator is DX:AX, with the resulting quotient in AX and the remainder in DX. For DWORD operands, the accumulator is EDX:EAX, with the resulting quotient in EAX and the remainder in EDX.

General Forms	***Function***
IDIV BH	Divide AX by BH, AH<=remainder, AL<=quotient
IDIV CX	Divide DX:AX by CX; DX<=remainder, AX<=quotient
IDIV ESI	Divide EDX:EAX by ESI; EDX<=remainder, EAX<=quotient
Example	
IDIV BH	Divide AX by BH; AH<=remainder, AL<=quotient
IDIV CX	Divide DX:AX by CX; DX<=remainder, AX<=quotient
IDIV ESI	Divide EDX:EAX by ESI; EDX<=remainder, EAX<=quotient

IMUL — Multiply (signed)

O	D	I	T	S	Z	A	P	C
*	-	-	-	?	?	?	?	*

Description:
This instruction performs a signed multiply. The flags are left in an indeterminate state except for OF and CF, which are cleared to 0 if the result of the multiplication is the same size as the multiplicand.
In the single operand form of the instruction, the result is placed in AX if the operands are BYTE, DX:AX for WORD operands, and EDX:EAX for DWORD operands. The multiple operand forms of the instruction only exist on 80386 and later processors.

General Forms	***Function***
IMUL *reg*	*acc <- acc * reg*
IMUL *mem*	*acc <- acc * mem*
IMUL *regd*,*regs*	*regd <- regd * regs*
IMUL *regd*,*mem*	*regd <- regd * mem*
IMUL *regd*,idata	*regd <- regd * idata*
IMUL *regd,regs,idata*	*regd <- regs * idata*
IMUL *regd*,*mem,idata*	*regd <- mem * idata*
Example	
IMUL CL	Multiply CL times AL; product replaces AX
IMUL CX	Multiply CX times AX; product replaces DX:AX
IMUL ECX	Multiply ECX times EAX; product replaces EDX:EAX
IMUL DX,AX,2	Multiply AX times 2; product replaces DX
IMUL MEM	Multiply AX times contents of memory location DS:MEM; product replaces DX:AX

IN	Read data from input port	O D I T S Z A P C - - - - - - - - -

Description:
This instruction reads a BYTE, WORD or DWORD into the accumulator from an I/O port. The immediate form of the instruction only allows a BYTE sized operand, and thus restricts access to the first 256 I/O ports. Placing the 16 bit port address in DX allows access to all I/O ports. The accumulator is either AL, AX or EAX.

General Forms	***Function***
IN *acc*,*idata* IN *acc*,DX	
Example	
IN AL,20H IN AX,DX	Input data from port 20H to AL Input data from port in DX to AX

INC	Increment	O D I T S Z A P C * - - - * * * * -

Description:
This instruction will add 1 to the value in the specified operand. This instruction does not affect the carry flag, but affects all other condition code flags.

General Forms	***Function***
INC *reg* INC *mem*	Add 1 to *reg* Add 1 to *mem*
Example	
INC DH INC MEM INC EDX	Add 1 to DH Add 1 to contents of memory location DS:MEM Add 1 to EDX

INS	Input string (80186 or later)	O D I T S Z A P C - - - - - - - - -

Description:
This instruction will read a value from the input port specified by DX and place the result in the memory location specified by ES:DI (or ES:EDI). The DI (or EDI) register will then be incremented or decremented, depending on the state of the direction flag, by an amount appropriate to the size of the operand. (1 for BYTE, 2 for WORD, 4 for DWORD).

Example	***Function***
INSB INSW INSD	Input byte sized data from port DX into memory at ES:DI Input word sized data from port DX into memory at ES:DI Input dword sized data from port DX into memory at ES:DI

INT	Software interrupt	O D I T S Z A P C - - - - - - - - -

Description:
This instruction saves the current flags and execution location on the stack. Control is then transferred to the location specified by the interrupt vector.

General Form	***Function***
INT *vector*	Software interrupt using *vector*.
Example	
INT 3 INT 21H	Software interrupt using vector 3. This is a special on byte instruction used for debugger breakpoint Software interrupt using vector 21H, Two byte instruction

INTO	Interrupt on overflow	O D I T S Z A P C - - - - - - - - -
Description: This instruction will test the state of the overflow flag and signal an exception if it is set by executing an INT 4.		
Example	*Function*	
INTO	Interrupt using vector 4 if overflow flag = 1	

INVD	Invalidate cache (80486 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
Example	*Function*	
INVD	Data in the internal cache is invalidated or erased	

INVLPG	Invalidate TLB (80486 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
Example	*Function*	
INVLPG	Clears translation look-aside buffer	

IRET/IRETD	Return from interrupt	O D I T S Z A P C ? ? ? ? ? ? ? ? ?
Description: This instruction is used to perform a return from an interrupt service routine. This instruction will pop the IP, CS, and Flags from the stack to return control to the location interrupted by either a hardware interrupt of a software interrupt.		
Example	***Function***	
IRET IRETD	16 bit FAR return from interrupt, pops FLAGS, CS, IP 32 bit FAR return from interrupt, pops EFLAGS, CS, EIP	

J*cc* Conditional jump

```
O D I T S Z A P C
- - - - - - - - -
```

Description:
The J*cc* instructions test the conditions described for each mnemonic. If the condition is met, the processor branches to the specified location within the current code segment. If the condition is false, execution continues with the instruction following the jump. On the 80286 and earlier processors, the target of the branch is specified with an 8 bit IP relative displacement. This limits the maximum distance for the jump to +/- 127 bytes approximately. On the 80386 and later processors, a 32 bit displacement is allowed, allowing the target of the jump to be anywhere within the current segment.

General Form	***Function***
J*cc offset*	Jump if condition is true

Examples

JA LOC	Jump to LOC if above (unsigned x>y) (CF=0 & ZF=0)
JAE LOC	Jump to LOC if above or equal (CF=0)
JB LOC	Jump to LOC if below (unsigned x<y) (CF=1)
JBE LOC	Jump to LOC if below or equal (CF=1 | ZF=1)
JC LOC	Jump to LOC if carry (CF=1)
JCXZ LOC	Jump to LOC if CX=0
JECXZ LOC	Jump to LOC if ECX=0
JE LOC	Jump to LOC if equal (ZF=1)
JG LOC	Jump to LOC if greater (signed x>y) (CF=0F & ZF=0)
JGE LOC	Jump to LOC if greater or equal (SF=OF)
JL LOC	Jump to LOC if less (signed x<y) (SF!=OF & ZF=0)
JLE LOC	Jump to LOC if less or equal (SF!=OF)
JNA LOC	Jump to LOC if not above (same as JBE)
JNAE LOC	Jump to LOC if not above or equal (same as JB)
JNB LOC	Jump to LOC if not below (same as JAE)
JNBE LOC	Jump to LOC if not below or equal (same as JA)
JNC LOC	Jump to LOC if carry not set (CF=0)
JNE LOC	Jump to LOC if not equal (ZF=0)
JNG LOC	Jump to LOC if not greater (SF!=OF & ZF=1)
JNGE LOC	Jump to LOC if not greater or equal (same as JL)
JNL LOC	Jump to LOC if not less than (same as JGE)
JNLE LOC	Jump to LOC if not less than or equal (same as JG)
JNO LOC	Jump to LOC if not overflow (OF=0)
JNP LOC	Jump to LOC if no parity (PF=0) (odd parity)
JNS LOC	Jump to LOC no sign (SF=0) (positive number)
JNZ LOC	Jump to LOC if not zero (ZF=0)
JO LOC	Jump to LOC if overflow (OF=1)
JP LOC	Jump to LOC if parity (PF=1) (even parity)
JPE LOC	Jump to LOC if parity even (PF=1)
JPO LOC	Jump to LOC if parity odd (PF=0)
JS LOC	Jump to LOC if sign (SF=1) (negative number)
JZ LOC	Jump to LOC if zero (ZF=1)

JMP — Near Jump

```
O D I T S Z A P C
- - - - - - - - -
```

Description:
This instruction transfers execution of the program to a new location. A new value is loaded into IP (or EIP) to perform the transfer of control. For the JMP *offset* form of the instruction, the target address is specified as a signed displacement that is added to the current contents of IP (or EIP). For the other forms of the instruction, the operand value replaces the current value of IP (or EIP).

General Forms	***Function***
JMP *offset*	Add *offset* to the current value in IP
JMP *reg*	Replace the contents of IP with the contents of *reg*
JMP *mem*	Replace the contents of IP with the contents of *mem*

Example	
JMP LOC	Jump to LOC.
JMP DX	Jump to address in DX
JMP NEAR PTR MEM	Jump NEAR to the address whose offset is in the WORD at memory location DS:MEM
JMP FAR PTR MEM	Jump FAR to the address contained in the DWORD at DS:MEM

JMP — Far Jump

```
O D I T S Z A P C
- - - - - - - - -
```

Description:
This instruction transfers execution of the program to a new location. The contents of the specified operand are loaded into IP (or EIP) and CS.

General Forms	***Function***
JMP *idata*	Replace CS:IP with *idata*.
JMP *mem*	Replace CS:IP with the contents of *mem*.

Example	
JMP LOC	Jump far to LOC.
JMP TABLE[SI]	Load CS:IP from the contents at the indicated memory location.
JMP FAR PTR MEM	Jump FAR to the address contained in the DWORD at DS:MEM

LAHF — Load AH from the FLAGS

```
O D I T S Z A P C
- - - - - - - - -
```

Description:
This instruction moves the contents of the low byte of the FLAGS register into AH.

Example	***Function***
LAHF	The low byte of the flags register is copied to AH

LAR — Load access rights

(80286 or later)

```
O D I T S Z A P C
- - - - - * - - -
```

Description:

Example	***Function***
LAR AX,BX	The access rights are loaded to AX from BX

LDS/LES/LFS/ LGS/LSS — Load far pointer

```
O D I T S Z A P C
- - - - - - - - -
```

Description:
This instruction will load a far pointer into a segment register plus the other specified register. The specified memory location contains the offset which will be loaded into a general purpose register, and the following location contains a segment value which will be loaded into the specified segment register.

General Forms	***Function***
L*seg reg*,*mem*	Load *reg* with the value at *mem* and load segment register *seg* with the contents of *mem*+2 (or *mem*+4 for 32 bit operations)

Example	
LDS DI,MEM	Load DS and DI from the DWORD at DS:MEM
LES AX,MEM	Load ES and AX from the DWORD at DS:MEM
LDS ESI,MEM	Load DS and ESI from the FWORD at DS:MEM
LES BX,ES:MEM	Load ES and BX from the DWORD at ES:MEM
LSS SP,MEM	Load SS and SP from the DWORD at DS:MEM

LEA — Load effective address

```
O D I T S Z A P C
- - - - - - - - -
```

Description:
This instruction loads the address specified by the memory operand into the specified register. The effective address calculation specified by the addressing mode of the memory operand is performed, and the resulting address (offset) is loaded into the register.

General Form	***Function***
LEA *reg*,*mem*	Load the *reg* with the effective address of *mem*.

Example	
LEA BX,MEM	Load the offset of MEM to BX
LEA DX,MEM[SI][BX]	Load DX with the offset of MEM+SI+BX
LEA SI,[DI+4]	Load SI with the offset of DI+4

LEAVE — Leave procedure (80186 or later)

```
O D I T S Z A P C
- - - - - - - - -
```

Description:
This instruction is the inverse of the ENTER instruction. LEAVE is used immediately before return from a procedure call to remove the stack frame created by ENTER. This instruction is the equivalent of the following instructions:

```
    MOV    SP,BP
    POP    BP
```

Example	***Function***
LEAVE	Reverses the action of ENTER

LGDT — Load global descriptor table register (80286 or later)

```
O D I T S Z A P C
- - - - - - - - -
```

Description:

Example	*Function*
LGDT MEM64	Loads the global descriptor table register from the 8 byte structure at memory location DS:MEM64

LIDT	Load interrupt descriptor table register (80286 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
Example	*Function*	
LIDT MEM64	Loads the interrupt descriptor table register from the 8 byte structure at memory location DS:MEM64	

LLDT	Load local descriptor table register (80286 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
Example	*Function*	
LLDT AX	Loads the local descriptor table register with the selector in AX	
LLDT MEM[SI]	Loads the local descriptor table register with the selector stored in memory at location DS:MEM+SI	

LMSW	Load machine status word (80286 or later)	O D I T S Z A P C - - - - - - - - -
Description: This instruction copies the contents of the machine status word into the AX register. If executed on an 80386 or later processor, it will move the contents of the low 16 bits of CR0 to the AX register.		
Example	*Function*	
LMSW AX	Copies the contents of AX into the machine status word (CR0). This instruction should only be used on an 80286 processor, as it has been superceded by the MOV CR0,EAX instruction on 80386 and later processors.	

LODS	Load string	O D I T S Z A P C - - - - - - - - -
Description: This instruction will load the BYTE, WORD, or DWORD at DS:SI (or DS:ESI) into the accumulator. Following the load, SI (or ESI) will be incremented or decremented, depending on the state of the direction flag, by an amount appropriate to the size of the operand.		
Example	***Function***	
LODSB	Load AL from the BYTE at memory location DS:SI	
LODSW	Load AX from the WORD at memory location DS:SI	
LODSD	Load EAX from the DWORD at memory location DS:ESI	

LOOP*cc*	Loop control. Decrement CX and Branch	O D I T S Z A P C - - - - - - - - -

Description:
These instructions perform a decrement and branch operation. The CX (or ECX) register is decremented by 1. If the result is 0, the branch is not taken. If the result of the decrement is not 0, the branch will be taken. For all variants other than LOOP, in addition to the decrement of CX, a test of the zero flag, ZF, is performed to determine if the branch should be taken.

General Forms	***Function***
LOOPcc *off*	
Example	
LOOP LOC	Decrement CX, if CX not zero, jump to location CS:LOC
LOOPD LOC	Decrement ECX, if ECX not zero, jump to location CS:LOC
LOOPZ LOC	Decrement CX, if CX not zero and ZF=1, jump to location CS:LOC
LOOPNZ LOC	Decrement CX, if CX not zero and ZF=0, jump to location CS:LOC
LOOPE LOC	Same as LOOPZ
LOOPNE LOC	Same as LOOPNZ

LSL	Load segment limit (80286 or later)	O D I T S Z A P C - - - - - * - - -

Description:

Example	*Function*
LSL AX,BX	Load AX with the segment limit from the selector in BX

LTR	Load task register (80286 or later)	O D I T S Z A P C - - - - - - - - -

Description:

Example	*Function*
LTR AX	Loads the selector in AX into the task register

MOV	Move data	O D I T S Z A P C - - - - - - - - -

Description:
This instruction copies the contents of the source operand into the destination operand.

General Forms	***Function***
MOV *reg,idata*	Move immediate value *idata* into register *reg*
MOV *mem,idata*	Move immediate value *idata* into memory location *mem*
MOV *regd,regs*	Move the contents of *regs* into *regd*
MOV *reg,mem*	Move the contents of memory location *mem* into *reg*
MOV *mem,reg*	Move the contents of *reg* into memory location *mem*
Examples	
MOV CX,BX	Move the contents of BX to CX
MOV MEM,AL	Move the contents of AL to the byte at memory location DS:MEM
MOV ECX,MEM	Move the contents of the DWORD at DS:MEM to ECX
MOV MEM,3	Move the immediate value 3 to the memory location at DS:MEM
MOV DX,MEM[SI+4]	Move the contents of memory location DS:MEM+SI+4 to DX

MOV	Move selector/segment	O D I T S Z A P C - - - - - - - - -

Description:
This instruction copies the contents of the source operand into the destination segment register.

General Forms	***Function***
MOV sreg,reg	Move the contents of reg into segment register sreg
MOV sreg,mem	Move the contents of memory location mem into segment register sreg
MOV reg,sreg	Move the contents of segment register sreg into register reg
MOV mem,sreg	Move the contents of segment register sreg into memory location mem

Examples

MOV DS,AX	Move the contents of AX into DS
MOV ES,ES:[BX+2]	Move the contents of the WORD at memory location ES:BX+2 to ES
MOV DX,SS	Move the contents of SS into DX
MOV MEM,DS	Move the contents of DS into memory location DS:MEM

MOV	Move special (80386 or later)	O D I T S Z A P C - - - - - - - - -

Description:

Example	*Function*
MOV CR0,EAX	Move the contents of EAX to Control Register 0
MOV EAX,DR1	Move the contents of DR1 to EAX

MOVS	Move string	O D I T S Z A P C - - - - - - - - -

Description:
This instruction copies the memory operand specified by DS:SI (or DS:ESI) to the memory location specified by ES:DI (or ES:EDI). Following the memory copy, SI and DI (or ESI and EDI) will be incremented or decremented, depending on the state of the direction flag, by an amount corresponding to the size of the operand transferred.

Example	*Function*
MOVSB	Move the byte at memory location DS:SI to memory location ES:DI
MOVSW	Move the word at memory location DS:SI to memory location ES:DI
MOVSD	Move the dword at memory location DS:ESI to location ES:EDI

MOVSX	Move and sign extend (80386 or later)	O D I T S Z A P C - - - - - - - - -

Description:
This instruction is used to convert a signed 8 bit value into a signed 16 bit value, or a signed 16 bit value into a signed 32 bit value. The sign bit of the source operand will be replicated through the high byte (for 8 -> 16 extension) or word (for 16 -> 32 bit extension) of the destination

Example	*Function*
MOVSX AX,AL	Sign extend AL into AX
MOVSX EDX,DX	Sign extend DX into EDX
MOVSZ ECX,MEM	Sign extend the word at memory location DS:MEM into ECX

MOVZX	Move and zero extend (80386 or later)	O D I T S Z A P C - - - - - - - - -
Description: This instruction is used to convert an unsigned 8 bit value into an unsigned 16 bit value or an unsigned 16 bit value into an unsigned 32 bit value. The high byte (for 8 -> 16 bit extension) or word (for 16 -> 32 bit extension) will be filled with 0.		
Example	*Function*	
MOVZX AX,AL MOVZX EBX,AX MOVZX EDX,MEM	Zero extend AL into AX Zero extend AX into EBX Zero extend the word at memory location DS:MEM into EDX	

MUL	Unsigned multiplication	O D I T S Z A P C * - - - ? ? ? ? *
Description: This instruction performs an unsigned multiply. The flags are left in an indeterminate state except for OF and CF, which are cleared to 0 if the result of the multiplication is the same size as the multiplicand. In the single operand form of the instruction, the result is placed in AX if the operands are BYTE, DX:AX for WORD operands, and EDX:EAX for DWORD operands. The multiple operand forms of the instruction only exist on 80386 and later processors		
General Forms	***Function***	
MUL *reg* MUL *mem* MUL *regd*,*regs* MUL *regd*,*mem* MUL *regd*,idata MUL *regd*,*regs*,*idata* MUL *regd*,*mem*,*idata*	*acc <- acc * reg* *acc <- acc * mem* *regd <- regd * regs* *regd <- regd * mem* *regd <- regd * idata* *regd <- regs * idata* *regd <- mem * idata*	
Example		
MUL CL	Multiply CL times AL, the product replaces AX	
MUL CX	Multiply CX times AX, the product replaces DX:AX	

NEG	Negate	O D I T S Z A P C * - - - * * * * *
Description: This instruction subtracts its operand from 0. This results in the 2's complement negation of the operand.		
General Form	***Function***	
NEG *reg*	Negate the contents or *reg*	
NEG *mem*	Negate the contents of *mem*.	
Example		
NEG CX NEG ARRAY[SI+2]	Negate the contents of CX Negate the contents of the specified memory location	

NOP	No operation	O D I T S Z A P C - - - - - - - - -
Description: This instruction performs no operation.		
Example		
NOP	No operation	

NOT	Complement or logical negation	`O D I T S Z A P C`
		`- - - - - - - - -`

Description: This instruction performs the logical, bitwise complement of its operand. Each bit of the operand is inverted.		
General Form	***Function***	
NOT *reg* NOT *mem*	Invert the bits of *reg* Invert the bits of the memory location *mem*	
Example		
NOT AX NOT VAR NOT ARRAY[DI]	Invert the bits of AX Invert the bits of memory location VAR Invert the bits of memory location ARRAY+DI	

OR	Logical inclusive OR	`O D I T S Z A P C`
		`0 - - - * * ? * 0`

Description: This instruction performs a logical OR operation between each bit of the source operand and each bit of the destination operand. The result is stored in the destination.		
General Form	*Function*	
OR *reg,idata* OR *mem,idata* OR *regd,regs* OR *reg,mem* OR *mem,reg*	Logical OR *reg* with *idata* Logical OR contents of memory location *mem* with *idata* Logical OR *regd* with *regs* Logical OR *reg* with contents of memory location *mem* Logical OR contents of memory location *mem* with *reg*	
Examples		
OR AL,07FH OR DATA3,DX OR CL,ES:[DI+2] OR BX,CX OR AX,MASK[SI]	Sets all but the high bit of AL Logical OR of word at memory location DS:DATA3 with DX Logical OR of byte at memory location ES:DI+2 with CL Logical OR of BX with CX Logical OR of word at memory location DS:MASK+SI with AX	

OUT	Write data to output port	`O D I T S Z A P C`
		`- - - - - - - - -`

Description: This instruction writes the value in the accumulator (AL or AX) to the specified data port. Using an immediate value as the port address allows access to ports 0-255 (0-0FFh). In order to access any output port address (0-FFFF) it is necessary to use the out dx,ax form of the instruction		
General Form	*Function*	
OUT *idata*,*acc* OUT DX,*acc*		
Examples		
OUT 27h,AL OUT DX,AX	Write the value in AL to port 27h Write the value in AX to port at address in DX	

OUTS	Output string (80186 or later)	O D I T S Z A P C - - - - - - - - -
Description: This instruction will write the byte, word, or dword at location DS:SI (DS:ESI for 32 bit operation) to the output port whose address is in DX. (EDX for 32 bit operation). The SI (ESI) register will then be adjusted according to the size of the operand and the setting of the direction flag. The OUTS instruction can be prefixed with a REP prefix, in which case, CX (ECX) contains the number of times the OUTS instruction is to be repeated.		
General Form	*Function*	
OUTSB OUTSW OUTSD	Output byte to port Output word to port Output dword to port	
Example		
OUTSB	Write byte at DS:SI to output port whose address is in DX	

POP	Pop data from stack	O D I T S Z A P C - - - - - - - - -
Description: This instruction pops the current value from the top of the stack, stores it in the destination, and adjusts the stack pointer.		
General Form	*Function*	
POP *reg* POP *mem*	POP top of stack into *reg* POP top of stack into memory location *mem*	
Example		
POP CX POP VAR1	POP top of stack into CX POP top of stack into memory location VAR1	

POP	Pop segment register from stack	O D I T S Z A P C - - - - - - - - -
Description: This instruction will pop the current value from the top of the stack into the indicated segment register and adjust the stack pointer. The CS register is not a valid destination, but any other segment register may be used.		
General Form	*Function*	
POP *sreg*	POP top of stack into segment register *sreg*	
Example		
POP DS	POP the top of the stack into DS	

POPA/POPAD	Pop all general registers (80186 or later)	O D I T S Z A P C - - - - - - - - -

Description:
This instruction will pop all general purpose registers (16 bit for POPA, 32 bit for POPAD) from the top of the stack and adjust the stack pointer.

The registers are popped in the following order:
DI, SI, BP, SP, BX, DX, CX, AX

This instruction was introduced with the 80186, and does not exist in earlier processors.

General Form	*Function*
POPA POPAD	POP 16 bit general registers from stack POP 32 bit general registers from stack
Example	
POPA	

POPF/POPFD	Pop flags from stack	O D I T S Z A P C ? ? ? ? ? ? ? ? ?

Description:
This instruction pops the FLAGS register (EFLAGS for POPFD) from the top of the stack and adjusts the stack pointer.

General Form	*Function*
POPF POPFD	POP flags from top of stack to FLAGS register POP flags from top of stack to EFLAGS register
Example	
POPF	

PUSH	Push data onto stack	O D I T S Z A P C - - - - - - - - -

Description:
This instruction pushes the operand onto the stack, and adjusts the stack pointer. The operand pushed becomes the new top of the stack.

General Form	*Function*
PUSH *idata* PUSH *reg* PUSH *mem*	PUSH immediate value onto the stack PUSH contents of *reg* onto the stack PUSH contents of memory location *mem* onto the stack
Example	
PUSH 12 PUSH DX PUSH TABLE[BX+2]	PUSH the value 12 onto the stack PUSH the contents of register DX onto the stack PUSH the contents of the memory location onto the stack

PUSH	Push segment register onto stack	O D I T S Z A P C - - - - - - - - -

Description:
This instruction will push the contents of the specified segment register onto the stack and adjust the stack pointer. The operand pushed becomes the new top of the stack.

General Form	*Function*
PUSH *sreg*	PUSH *sreg* onto the stack
Example	
PUSH ES	PUSH the contents of ES onto the stack

PUSHA/ PUSHAD	Push all general registers onto stack (80186 or later)	O D I T S Z A P C - - - - - - - - -
Description: This instruction will push the contents of all of the general purpose registers (16 bit for PUSHA, 32 bit for PUSHAD) onto the stack and adjust the stack pointer.		
General Form	*Function*	
PUSHA PUSHAD	PUSH 16 bit general registers onto the stack PUSH 32 bit general registers onto the stack	
Example		
PUSHA		

PUSHF/ PUSHFD	Push flags onto stack	O D I T S Z A P C - - - - - - - - -
Description: This instruction pushes the FLAGS register (EFLAGS for PUSHFD) onto the top of the stack and adjusts the stack pointer.		
General Form	*Function*	
PUSHF PUSHFD	PUSH 16 bit flags onto the stack PUSH 32 bit eflags onto the stack	
Example		
PUSHF		

RCL	Rotate left through carry	O D I T S Z A P C * - - - - - - - *
Description: This instruction concatenates the carry flag with the specified operand and rotates the result left by the specified number of bit positions. For each bit position of rotation, the current contents of the carry flag goes to the low bit position of the operand, and the high bit of the operand goes to the carry flag. (Note: on processors prior to the 80386, the only valid value for *idata* is 1)		
General Form	*Function*	
RCL *reg,idata* RCL *mem,idata* RCL *reg*,CL RCL *mem*,CL	Rotate register *reg* left through carry by *idata* bit positions Rotate memory location *mem* left by *idata* bit positions Rotate register *reg* left by the number of bit positions in CL Rotate memory location mem left by the number of bit posn's in CL	
Example		
RCL BX,1 RCL VAR,CL RCL DL,CL	Rotate register BX left through carry by 1 bit position Rotate memory location VAR left through carry by CL bit positions Rotate register DL left through carry by CL bit positions	

RCR — Rotate right through carry

O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

Description:
This instruction concatenates the carry flag with the specified operand and rotates the result right by the specified number of bit positions. For each bit position of rotation, the current contents of the carry flag goes to the low bit position of the operand, and the high bit of the operand goes to the carry flag. (Note: on processors prior to the 80386, the only valid value for *idata* is 1)

General Form	*Function*
RCR *reg,idata*	Rotate register *reg* right through carry by *idata* bit positions
RCR *mem,idata*	Rotate memory location *mem* right by *idata* bit positions
RCR *reg*,CL	Rotate register *reg* right by the number of bit positions in CL
RCR *mem*,CL	Rotate memory location mem right by the number of bit posn's in CL
Example	
RCR BX,1	Rotate register BX right through carry by 1 bit position
RCR VAR,CL	Rotate memory location VAR right through carry by CL bit positions
RCR DL,CL	Rotate register DL right through carry by CL bit positions

REP*cc* — Repeat string prefix

O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

Description:
The repeat prefix may be applied to any string instruction. When used with a string instruction, the contents of the CX register (ECX for 32 bit operation) will be decremented and the string instruction repeated until CX goes to 0. If a REPcc form of the prefix is used, then the state of ZF is also tested when using CMPS or SCAS instructions.

General Form	*Function*
REP	Repeat while CX (ECX) is not 0
REPE	Repeat while CX (ECS) is not 0 and ZF is set
REPZ	Repeat while CX (ECX) is not 0 and ZF is set. (same as REPE)
REPNE	Repeat while CX (ECX) is not 0 and ZF is clear
REPNZ	Repeat while CX (ECX) is not 0 and ZF is clear (same as REPNE)
Example	
REP MOVSB	Repeat MOVSB while CX is not 0
REPZ SCASW	Repeat SCSAW while CX is not 0 and ZF is set
REPNE CMPSB	Repeat CMPSB while CX is not 0 and ZF is clear

RET — Near return from procedure

O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

Description:
This instruction restores the IP register (EIP for 32 bit operation) to the value it held before the last CALL instruction. The previous value of IP (EIP) is popped from the stack. If the optional *idata* operand is present, the *idata* value is added to SP (ESP) after the return address is popped from the stack.

General Form	*Function*
RET	Return from near subroutine call
RET *idata*	Return from near subroutine call and adjust stack by *idata*
Example	
RET	Return from subroutine
RET 4	Return from subroutine and then add 4 to SP (ESP)

RETF	Far return from procedure	`O D I T S Z A P C`
		`- - - - - - - - -`

Description: This instruction restores the CS and IP registers (CS and EIP for 32 bit operation) to the values held before the last CALL instruction. The previous values of IP (EIP) and CS are popped from the stack. If the optional *idata* operand is present, the *idata* value is added to SP (ESP) after the return address is popped from the stack.		
General Form	*Function*	
RETF RETF *idata*	Return from far subroutine call Return from far subroutine call and adjust stack by idata	
Example		
RETF RETF 8	Return from subroutine call Return from subroutine call and add 8 to SP (ESP)	

ROL	Rotate left	`O D I T S Z A P C`
		`* - - - - - - - *`

Description: This instruction rotates the destination operand left by the specified number of bit positions. For each bit position of rotation, the high bit position of the destination goes to the low bit position and also to the carry flag. (Note: on processors prior to the 80386, the only valid value for *idata* is 1)		
General Form	*Function*	
ROL *reg,idata* ROL *mem,idata* ROL *reg*,CL ROL *mem*,CL	Rotate register *reg* left by *idata* bit positions Rotate memory location *mem* left by *idata* bit positions Rotate register *reg* left by the number of bit positions in CL Rotate memory location mem left by the number of bit posn's in CL	
Example		
ROL BX,1 ROL VAR,CL ROL DL,CL	Rotate register BX left by 1 bit position Rotate memory location VAR left by CL bit positions Rotate register DL left by CL bit positions	

ROR	Rotate right	`O D I T S Z A P C`
		`* - - - - - - - *`

Description: This instruction rotates the destination operand right by the specified number of bit positions. For each bit position of rotation, the low bit position of the destination operand goes to the high bit position and also to the carry flag. (Note: on processors prior to the 80386, the only valid value for *idata* is 1)		
General Form	*Function*	
ROR *reg,idata* ROR *mem,idata* ROR *reg*,CL ROR *mem*,CL	Rotate register *reg* right by *idata* bit positions Rotate memory location *mem* right by *idata* bit positions Rotate register *reg* right by the number of bit positions in CL Rotate memory location mem right by the number of bit posn's in CL	
Example		
ROR BX,1 ROR VAR,CL ROR DL,CL	Rotate register BX right by 1 bit position Rotate memory location VAR right by CL bit positions Rotate register DL right by CL bit positions	

SAHF — Store AH to flags

O	D	I	T	S	Z	A	P	C
-	-	-	-	*	*	*	*	*

Description:
This instruction transfers the contents of the AH register to the low 8 bit positions of the FLAGS register (EFLAGS for 32 bit operation).

General Form	*Function*
SAHF	Set flags from AH

Example	
SAHF	Set flags from AH

SAL — Shift left arithmetic

O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	?	*	*

Description:
This instruction shifts the destination operand left arithmetically by the specified number of bit positions. The low order bit positions of the destination are set to 0. The high order bits shifted out of the destination are lost. The arithmetic shift left (SAL) and logical shift left (SHL) are equivalent operations. (Note: on processors prior to the 80386, the only valid value for *idata* is 1)

General Form	*Function*
SAL *reg*,*idata*	Shift register *reg* left arithmetically by *idata* bit positions
SAL *mem*,*idata*	Shift memory location *mem* left arithmetically by *idata* bit positions
SAL *reg*,CL	Shift register *reg* left arithmetically by CL bit positions
SAL *mem*,CL	Shift memory location *mem* left arithmetically by CL bit positions

Example	
SAL BL,1	Shift BL left arithmetically by 1 bit position
SAL VAR,1	Shift memory location VAR left arithmetically by 1 bit position
SAL DX,CL	Shift DX left arithmetically by CL bit positions

SAR — Shift right arithmetic

O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	?	*	*

Description:
This instruction shifts the destination operand right arithmetically by the specified number of bit positions. The value of the sign bit is replicated to the next lower bit positions, and the low order bits of the destination value are lost. (Note: on processors prior to the 80386, the only valid value for *idata* is 1)

General Form	*Function*
SAR *reg*,*idata*	Shift register *reg* right arithmetically by *idata* bit positions
SAR *mem*,*idata*	Shift memory location *mem* right arithmetically by *idata* bit positions
SAR *reg*,CL	Shift register *reg* right arithmetically by CL bit positions
SAR *mem*,CL	Shift memory location *mem* right arithmetically by CL bit positions

Example	
SAR BL,1	Shift BL right arithmetically by 1 bit position
SAR VAR,1	Shift memory location VAR right arithmetically by 1 bit position
SAR DX,CL	Shift DX right arithmetically by CL bit positions

SBB	Subtract with borrow	O D I T S Z A P C * - - - * * * * *
Description: This instruction subtracts the source operand and the current value of the carry flag from the destination operand. This instruction treats the carry flag as a borrow flag from a previous subtraction.		
General Form	*Function*	
SBB *reg*,*idata* SBB *mem*,*idata* SBB *regd*,*regs* SBB *reg*,*mem* SBB *mem*,*reg*	Subtract *idata* with borrow from register *reg* Subtract *idata* with borrow from memory location *mem* Subtract register *regd* with borrow from register *regs* Subtract memory location *mem* from register *reg* Subtract register *reg* from memory location *mem*	
Example		
SBB AX,CX SBB VAR,DX SBB BL,VAR	Subtract with borrow CX from AX Subtract with borrow DX from memory location VAR Subtract with borrow memory location VAR from BL	

SCAS	Scan string	O D I T S Z A P C * - - - * * * * *
Description: This instruction compares the value in the accumulator (AL, AX or EAX) with the contents of the memory location specified by ES:DI (or ES:EDI). The flags are set according to the results of the comparison and the contents of the DI (EDI) register is adjusted by the size of the operand. The size of the operand is added to DI (EDI) if the direction flag is clear and subtracted from DI (EDI) if the direction flag is set. A repeat prefix (REP, REPE, REPZ, REPNE, REPNZ) can be used with this instruction to cause it to be repeated.		
General Form	*Function*	
SCASB SCASW SCASD	Scan string byte Scan string word Scan string double word (80386 or later)	
Example		
SCASB SCASW	Compare AL with the byte at ES:DI, set flags, adjust DI by 1 Compare AX with the word at ES:DI, set flags adjust DI by 2	

SET*cc*	Set byte on *condition* (80386 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
General Form	*Function*	
Example		

SGDT	Store global descriptor table register (80286 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
General Form	*Function*	
Example		

SHL	Shift left logical	O D I T S Z A P C * - - - * * ? * *
Description: This instruction shifts the destination operand left logically by the specified number of bit positions. The low order bit positions of the destination are set to 0. The high order bits shifted out of the destination are lost. The arithmetic shift left (SAL) and logical shift left (SHL) are equivalent operations. (Note: on processors prior to the 80386, the only valid value for *idata* is 1)		
General Form	*Function*	
SHL *reg*,*idata*	Shift register *reg* by *idata* bit positions	
SHL *mem*,*idata*	Shift memory location *mem* left by *idata* bit positions	
SHL *reg*,CL	Shift register *reg* left by CL bit positions	
SHL *mem*,CL	Shift memory location *mem* left by CL bit positions	
Example		
SHL BL,1	Shift BL left by 1 bit position	
SHL VAR,1	Shift memory location VAR left by 1 bit position	
SHL DX,CL	Shift DX left by CL bit positions	

SHLD	Shift double operand left logical (80386 or later)	O D I T S Z A P C * - - - * * ? * *
Description:		
General Form	*Function*	
Example		

SHR	Shift right logical	O D I T S Z A P C * - - - * * ? * *
Description: This instruction shifts the destination operand right logically by the specified number of bit positions. The high order bit positions of the destination are set to 0. The low order bits shifted out of the destination are lost. (Note: on processors prior to the 80386, the only valid value for *idata* is 1)		
General Form	*Function*	
SHR *reg*,*idata*	Shift register *reg* right by *idata* bit positions	
SHR *mem*,*idata*	Shift memory location *mem* right by *idata* bit positions	
SHR *reg*,CL	Shift register *reg* right by CL bit positions	
SHR *mem*,CL	Shift memory location *mem* right by CL bit positions	
Example		
SHR BL,1	Shift BL right by 1 bit position	
SHR VAR,1	Shift memory location VAR right by 1 bit position	
SHR DX,CL	Shift DX right by CL bit positions	

SHRD	Shift double operand right logical (80386 or later)	O D I T S Z A P C * - - - * * ? * *
Description:		
General Form	*Function*	
Example		

SIDT	Store interrupt descriptor table register (80286 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
General Form	*Function*	
Example		

SLDT	Store local descriptor table register (80286 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
General Form	*Function*	
Example		

SMSW	Store machine status word (80286 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
General Form	*Function*	
Example		

STC	Set carry flag	O D I T S Z A P C - - - - - - - - 1
Description: This instruction will set the carry flag, CF, to 1.		
General Form	*Function*	
STC	Set the carry flag	
Example		
STC	Set the carry flag	

STD	Set direction flag	O D I T S Z A P C - 1 - - - - - - -
Description: This instruction will set the direction flag, DF, to 1. This setting causes string instructions to decrement the pointer registers.		
General Form	*Function*	
STD	Set the direction flag to 1	
Example		
STD	Set the direction flag to 1	

STI	Set interrupt flag	O D I T S Z A P C - - 1 - - - - - -
Description: This instruction will set the interrupt flag. When the interrupt flag is set, the processor will respond to interrupt requests.		
General Form	*Function*	
STI	Set the interrupt flag	
Example		
STI	Set the interrupt flag	

STOS	Store string	O D I T S Z A P C - - - - - - - - -
Description: This instruction will write the contents of the accumulator (AL, AX or EAX) to the memory location specified by ES:DI (ES:EDI for 32 bit operations). It then adjusts DI (EDI) according to the size of the operand the current setting of the direction flag. The operand size is added to DI (EDI) if the direction flag is clear. It is subtracted from DI (EDI) if the direction flag is set. A repeat prefix (REP) can be used with this instruction to cause it to be repeated.		
General Form	*Function*	
STOSB STOSW STOSD	Store string byte Store string word Store string double word (80386 and later)	
Example		
STOSB STOSW	Store contents of AL at memory location ES:DI and adjust DI by 1 Store contents of AX at memory location ES:DI and adjust DI by 2	

STR	Store task register (80286 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
General Form	*Function*	
Example		

SUB	Subtract borrow	O D I T S Z A P C * - - - * * * * *
Description: This instruction subtracts the source operand from the destination operand. This instruction treats the carry flag as a borrow flag and will set the carry flag if a borrow occurs.		
General Form	*Function*	
SUB *reg*,*idata*	Subtract *idata* from register *reg*	
SUB *mem*,*idata*	Subtract *idata* from memory location *mem*	
SUB *regd*,*regs*	Subtract register *regd* from register *regs*	
SUB *reg*,*mem*	Subtract memory location *mem* from register *reg*	
SUB *mem*,*reg*	Subtract register *reg* from memory location *mem*	
Example		
SUB AX,CX	Subtract CX from AX	
SUB VAR,DX	Subtract DX from memory location VAR	
SUB BL,VAR	Subtract memory location VAR from BL	

TEST	Test bits	O D I T S Z A P C 0 - - - * * ? * 0
Description: This instruction is used to perform a logical comparison of the bits in the two operands. The contents of the source and destination registers are bitwise ANDed. The result of the AND operation is discarded and the flags are set according to the logical result of the AND.		
General Form	*Function*	
TEST *reg*,*idata*	Bitwise and register *reg* with *idata* and set the flags	
TEST *mem*,*idata*	Bitwise and memory location *mem* with *idata* and set the flags	
TEST *regd*,*regs*	Bitwise and register *regd* with register *regs* and set the flags	
TEST *reg*,*mem*	Bitwise and register *reg* with memory location *mem* and set the flags	
TEST *mem*,*reg*	Bitwise and memory location *mem* with register *reg* and set the flags	
Example		
TEST AL,3Fh	AND the contents of AL with 3Fh and set the flags	
TEST AX,DX	AND the contents of AX with DX and set the flags	
TEST VAR,01h	AND the contents of memory location VAR with 01h and set flags	

VERR	Verify read access (80286 or later)	O D I T S Z A P C - - - - - * - - -
Description:		
General Form	*Function*	
Example		

VERW	Verify write access (80286 or later)	O D I T S Z A P C - - - - - * - - -
Description:		
General Form	*Function*	
Example		

WAIT	Wait until not busy	O D I T S Z A P C - - - - - - - - -
Description: This instruction causes the processor to go into an idle state until the BUSY pin goes to an inactive state. It is normally used to synchronize the main processor with a coprocessor such as the math coprocessor (8087). This instruction should be used after floating point coprocessor instructions to ensure that the coprocessor instruction has completed prior to accessing the result.		
General Form	*Function*	
WAIT	Wait for coprocessor not busy	
Example		
WAIT		

WBINVD	Write and invalidate cache (80486 or later)	O D I T S Z A P C - - - - - - - - -
Description:		
General Form	*Function*	
Example		

XADD	Exchange and add (80486 or later)	O D I T S Z A P C * - - - * * * * *
Description:		
General Form	*Function*	
Example		

XCHG	Exchange	O D I T S Z A P C - - - - - - - - -
Description: This instruction will exchange the contents of the two operands.		
General Form	*Function*	
XCHG regd,regs XCHG reg,mem XCHG mem,reg	Exchange the contents of register regd with register regs Exchange the contents of register reg with memory location mem Exchange the contents of memory location mem with register reg	
Example		
XCHG AX,DX XCHG VAR,CL	Exchange the contents of AX with DX Exchange the contents of CL with memory location VAR	

XLAT	Translate using table	O D I T S Z A P C - - - - - - - - -
Description: This instruction uses the contents of AL as an index into a table located at the memory location specified by DS:BX (DS:EBX for 32 bit operation). The contents of AL is replaced by the byte at the indexed location in the table.		
General Form	*Function*	
XLAT	Translate using table	
Example		
XLAT	Translate AL using table at DS:BX	

XOR	Logical exclusive OR	O D I T S Z A P C 0 - - - * * ? * 0
Description: This instruction performs a logical exclusive or operation between each bit of the source operand and each bit of the destination operand. The result is stored in the destination.		
General Form	*Function*	
XOR *reg,idata*	Logical XOR *reg* with *idata*	
XOR *mem,idata*	Logical XOR contents of memory location *mem* with *idata*	
XOR *regd,regs*	Logical XOR *regd* with *regs*	
XOR *reg,mem*	Logical XOR *reg* with contents of memory location *mem*	
XOR *mem,reg*	Logical XOR contents of memory location *mem* with *reg*	
Examples		
XOR AL,07FH	Inverts all but the high bit of AL	
XOR DATA3,DX	Logical XOR of word at memory location DS:DATA3 with DX	
XOR CL,ES:[DI+2]	Logical XOR of byte at memory location ES:DI+2 with CL	
XOR BX,CX	Logical XOR of BX with CX	
XOR AX,MASK[SI]	Logical XOR of word at memory location DS:MASK+SI with AX	

Revision History:

09/05/2001 (GeneA): Initial version completed
09/10/2001 (GeneA); Corrected error in example to MOV instruction