UNIT – 2: *Assembly Language Instructions - I*

Arithmetic Instructions

ADD / ADC
ADC – Add with Carry
ADD – Add

ADC Destination, Source
ADD Destination, Source

These instructions add a number form some source to a number from some destination and put the result in the specified destination. The Add with Carry instruction, ADC, also adds the status of the carry flag into the result. The source may be an immediate number, a register, or a memory location specified by any one of the 24 addressing modes. The destination may be a register or a memory location specified by any one of the 24 addressing modes. The source and the destination in an instruction cannot both be memory location. The source and the destination must be of the same type. In other words, they must both be byte locations, or they must both be word locations. If you want to add a byte or a word, you must copy the byte to a word location and fill the upper byte of the word with 0's before adding. Flags affected: AF, CF, OF, PF, SF, ZF.

Examples:

ADD AL, 47H	: Add immediate number 74H to contents of AL. Result in AL
ADC CL, BL	: Add contents of BL plus carry status to contents of CL.
ADD DX, BX	: Add contents of BX to contents of DX.
ADD DX, [SI]	: Add word from memory at offset [SI] in DS to contents of DX

INC – Increment – INC Destination

The INC instruction adds 1 to a specified register or to a memory location specified in any one fo the 24 ways. AF, OF, PF, SF and ZF are affected by this instruction. Note that Carry Flag CF is not affected. This means that if an 8-bit destination containing FFH or a 16-bit destination containing FFFFH is incremented, the result will be all 0's with no carry.

Examples:

INC BL	: Add 1 to contents of BL register
INC CX	: Add 1 to contents of CX register

SUB – Subtract

SUB Destination, Source

This instruction subtracts the number in the indicated source from the number in the indicated destination and put the result in the indicated destination. The subtract instruction, SUB, subtracts just the contents of the specified source form the contents of the specified destination.

The source and the destination must both be of type byte or both be of type word. If you want to subtract a byte from a word, you must first move the byte to a word location such as a 16-bit register and fill the upper byte of the word with 0's. AF, CF, OF, PF, SF, and ZF are updated by the SUB instruction.

Examples:

SUB CX, BX	: CX – BX, results in CX
SUB AX, 3427H	: Subtract immediate number 3427H from AX

DEC – Decrement Destination Register or Memory – DEC Destination

This instruction subtracts 1 from the destination word or byte. The destination can be a register or a memory location specified by any one of the 24 addressing modes. AF, OF, PF, SF and ZF are update, but CF is not affected. This means that if an 8-bit destination containing 00H or a 16-bit destination containing 0000H is decremented, the result will be FFH or FFFFH with no carry.

Examples:

DEC CL	: Subtract 1 from contents of CL register
DEC BP	: Subtract 1 from contents of BP register

CMP- Compare Byte or Word

CMP Destination, Source

This instruction compares a byte from the specified source with a byte form the specified destination, or a word from the specified source with a word from the specified destination. The source can be an immediate number, a register, or a memory location specified by one of the 24 addressing modes. The destination can be a register or a memory location. However, the source and the destination cannot both be memory locations in the same instruction. The comparison is actually done by subtracting the source byte or word from the destination byte or word. The source and the destination are not changed, but the flags are set to indicate the results of the comparison. AF, OF, SF, ZF, PF and CF are updated by the CMP instruction. For the instruction CMP CX, BX, CF, ZF and SF will be left as follows:

	CF	ZF	SF	
CX = BX	0	1	0	: Result of subtraction is 0
CX > BX	0	0	0	: No borrow required, so CF = 0
CX < BX	1	0	1	: Subtraction required borrow, so CF = 1

Examples:

CMP AL, 01H	: Compare immediate number 01H with byte in AL.

CMP BH, CL	: Compare byte in CL with byte in BH
CMP CX, TEMP_MIN	: Compare word in DX at displacement TEMP_MIN with word in CX.

MUL – Multiply Unsigned Bytes or Words

MUL Source

This instruction multiplies an unsigned byte form some source times an unsigned byte in the AL register or an unsigned word from some source times an unsigned word in the AX register. The source can be a register or a memory location specified by any one of the 24 addressing modes. When a byte is multiplied by the contents of AL, the result is put in AX. A 16-bit destination is required because the result of multiplying an 8-bit number by an 8-bit number can be as large as 16 bits. The most significant byte of the result is put in AH, and the least significant byte of the result is put in AL. When a word is multiplied by the contents of AX, the product can be as large as 32 bits. The most significant word of the result is put in the DX register, and the least significant word of the result is put in the AX register. If the most significant byte of a 16-bit result or the most significant word of a 32 bit result is 0, CF and OF will both be 0's.

If you want to multiply a byte by a word, you must first move the byte to a word location such as an extended register and fill the upper byte of the word with all 0's.

Examples:

MUL BH	: AL times BH, result in AX
MUL CX	: AX times CX, result high word in DX, low word in AX
MUL BYTE PTR[BX]	: AL times byte in DX pointed to by [BX]

IMUL – Multiply Signed Numbers

IMIL Source

This instruction multiplies a signed byte from some source times a signed byte in AL or a signed word from some source times a signed word in AX. The source can be another register or a memory location specified by any one of the 24 addressing modes. When a byte from some source is multiplies by AL, the signed result will be put in AX. A 16-bit destination is required because the result of multiplying two 8-bit numbers can be as large as 16 bits. When a word from some source is multiplied by AX, the result can be as large as 32 bits.

Examples:

ILMUL BH	: Signed byte in AL times signed byte in BH, result in AX
IMUL AX	: AX times AX, result in DX and AX.

Multiplying a signed byte by a signed word

DIV – Unsigned Divide

DIV Source

This instruction is used to divide an unsigned word by a byte or to divide an unsigned doubleword (32 bits) by a word.

When a word is divided by a byte, the word must be in the AX register. The divisor can be in a register or a memory location. After the division, AL will contain an 8-bit result (quotient), and AH will contain an 8-bit remainder. If an attempt is made to divide by 0 or if the quotient is too large to fit in AL, the 8086 will automatically do a type 0 interrupt.

When a doubleword is divided by a word, the most significant word of the doubleword must be in DX, and the least significant word of the dobleword must be in AX. After then division, Ax will contain the 16-bit result (quotient), and DX will contain a 16-bit remainder. Again, if an attempt is made to divide by 0 or if the quotient is too large to fit in AX, the 8086 will do a type 0 interrupt.

For a DIV, the dividend (numerator) must always be in AX or DX and AX, bu the source of the divisor (denominator) can be a register or a memory location specified by any one of the 24 addressing modes. If the divisor does not divide an integral number of times into the dividend, the quotient is truncated, not rounded.

Examples:

DIV BL	: Divide word in AX by byte in BL. Quotient in AL, Remainder in AH.
DIV CX	: Divide doubleword in DX and AX by word in DX. Quotient in AX, remainder in DX.

IDIV – Divide by Signed Byte or Word

IDIV Source

This instruction is used to divide a signed word by a signed byte, or to divide a signed doubleword (32 bits) by a signed word.

When dividing a signed word by a signed byte, the word must be in AX register. The divisor can be in an 8-bit register or a memory location. After the division, AL will contain the signed result (quotient), and AH will contain the signed remainder. The sign of the remainder will be the same as the sign of the dividend.

When dividing a signed doubleword by a signed word, the most significant word of the dividend (numerator) must be in the DX register, and the least significant word or the dividend must be in the AX register. The divisor can be in any other 16-bit register or memory location. After the division, AX will contain a signed 16-bit quotient, and DX will contain a signed 16-bit remainder. If the divisor does not divide evenly into the dividend, the quotient will be truncated, not rounded.

Examples:

IDIV BL	: Signed word in AX/signed byte in BL
IDIV BP	: Signed doubleword in DX and AX/Signed word in BP

NEG – Form 2's Complement

NEG Destination

This instruction replaces the number in a destination with the 2's complement of that number. The destination can be a register or a memory location specified by any one of the 2 addressing modes. This instruction forms the 2' complement by subtracting the original word or byte in the indicated destination from zero.

Examples:

NEG AL	: Replace number in AL with its 2's complement
NEG BX	: Replace word in BX with its 2's complement
NEG BYTE PTR[BX]	: Replace byte at offset [BX] in DS with its 2's complement

Data Transfer Instructions

MOV – Copy a Word or Byte

MOV Destination, Source

The MOV instruction copies a word or byte of data form a specified source to a specified destination The destination can be a register or a memory location. The source can be a register, a memory location, or an immediate number. The source and destination in an instruction cannot both be memory location. The source and destination in a MOV instruction must both be of type byte, or they must both be of type word. MOV instructions do not affect any flags.

Examples:

MOV CX, 037AH	: Put the immediate number 037AH in CX
MOV AX, BX	: Copy the content of register BX to AX
MOV DL, [BX]	: Copy byte from memory at [BX] to DL

XCHG

XCH Destination, Source

The XCHG instruction exchanges the contents of a register with the contents of another register or the contents of a register with the contents of memory locations. The XCHG cannot directly exchange the contents of two memory locations. A memory location can be specified as the source or as the destination by any of the 24 addressing modes. The source and destination must both be words, or they must both be bytes. The segment register cannot be used in this instruction. No flags are affected by this instruction.

Examples:

XCHG AX, BX	: Exchange word in AX with word in DX
XCHG	: Exchange bye in BL with byte in CH

LEA – Load Effective Address

LEA Register, Source

This instruction determines the offset of the variable or memory location named as the source and puts this offset in the indicated 16-bit register. LEA changes no flags.

Examples:

LEA BX, PRICES	: Load BX with offset of PRICES in DS
LEA BP, SS: STACK_TOP	: Load BP with offset of STACK_TOP in SS

Rotate and Shift Instructions

ROL – Rotate All Bits of Operand Left, MSB to LSB

ROL Destination, Count

This instruction rotates all the bits in a specified word or byte to the left some number of bit positions. The operation can be thought of as circular, because the data bit rotated out of the MSB is circled back into the LSB. The data bit rotated out of the MSB is also copied to CF during ROL. In the case of multiple bit rotates, CF will contain a copy of the bit most recently move d out of the MSB.

The destination operand can be in a register or in a memory location specified by any one of the 24 addressing modes. If you want to rotate the operand one bit position, you can specify this by putting a 1 in the count position of the instruction. To rotate more than one bit position, load the desired number in the CL register and put “CL” in the count position of the instruction.

ROL affects only CF and OF. After ROL, CF will contain the bit most recently rotated out of the MSB. OF will be a 1 after a single bit ROL if the MSB was changed by the rotate.

THE ROL instruction can be used to swap the nibbles in a byte or to swap the bytes in a word. It can also be used to rotate a bit into CF, where it can be checked and acted upon by the Conditional Jump instructions JC (Jump if Carry) and JNC (Jump if No Carry).

Examples:

```
ROL AX, 1          : Word in AX 1 bit position left, MSB to LSB and CF
MOV CL, 04H        : Load number of bits to rotate in CL
ROL BL, CL         : Rotate BL 4 bit positions
```

ROR – Rotate All Bits of Operand Right, LSB to MSB

ROR Destination, Count

This instruction rotates all the bits of the specified word or byte some number of bit positions to the right. The operation is described as a rotate rather than a shift because the bit moved out of the LSB is rotated around into the MSB. To help visualize the operation, think of the operand as a loop with the LSB connected around to the MSB. The data bit moved out of the LSB is also copied to CF during ROR. In the case of multiple-bit rotates, CF will contain a copy of the bit most recently move out of the LSB.

The destination operand can be in a register or in a memory location specified by any one of the 24 addressing modes. If you want to rotate the operand one bit position, you can specify this by putting a1 in the count position of the instruction. To rotate more than one bit position, load the desired number in the CL register and put “CL” in the count position of the instruction.

ROR affects only CF and OF. After ROR, CF will contain the bit most recently rotated out of the LSB. For a single-bit rotate, OF will be a 1 after ROR if the MSB is changed by the rotate.

THE ROR instruction can be used to swap the nibbles in a byte or to swap the bytes in a word. It can also be used to rotate a bit into CF, where it can be checked and acted upon by the Conditional Jump instructions JC (Jump if Carry) and JNC (Jump if No Carry).

Examples:

```
ROR BL, 1              : Rotate all bits in BL, right 1 bit position LSB to MSB and to CF
MOV CL, 08H                        : Load CL with number of bit positions to be rotated.
ROR WORD PTR [BX], CL          : Rotate word in DS at offset [BX] 8 bit positions right
```

SAL / SHL – Shift Operand Bits Left, Put Zero in LSB(s)

SAL / SHL Destination, Count

SAL and SHL are tow mnemonics for the same instruction. This instruction shifts each bit in the specified destination some number of bit positions to the left. As a bit is shifted out of the LSB position, a 0 is put in the LSB position. The MSB will be shifted into CF. In the case of multiple bit shifts, CF will contain the bit most recently shifted in form the MSB. Bits shifted into CF previously will be lost.

The destination operand can be a byte or a word. It can be in a register or in a memory location specified by any one of the 24 addressing modes.

If the desired number of shifts is one, this can be specified by putting a 1 in the count position of the instruction. For shifts of more than 1 bit position, the desired number of shifts is loaded into the CL register, and CL is put in the count position of the instruction. The advantage of using the CL register is that the number of shifts can be dynamically calculated as the program executes.

The flags are affected as follows: CF contains the bit most recently shifted in from MSB. For a count of one, OF will be 1 if CF and the current MSB are not the same. For multiple-bit shifts, OF is undefined. SF and ZF will be updated to reflect the condition of the destination. PF will have meaning only for an operand in AL. AF is undefined.

The SAL or SHL instruction can also be used to multiply an unsigned binary number by a power of 2.

Examples:

```
          SAL BX, 1          : Shift word in BX 1 bit position left, 0 in LSB
          MOV CL, 02H        : Load desired number of shifts in CL
          SAL BP, CL         : Shift word in BP left (CL) bit
```

SHR – Shift Operand Bits Right, Put Zero in MSB(s)

SHR Destination, Count

This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted right out of the MSB position, a 0 is put in its place. The bit shifted out of the LSB position goes to CF. In the case of a multiple-bit shift, CF will contain the bit most recently shifted in from the LSB. Bits shifted into CF previously will be lost.

The destination operand can be a byte or a word in a register or in a memory location specified by any one of the 24 addressing modes.

If the desired number of shifts is one, this can be specified by putting a 1 in the count position of the instruction. For shifts of more than one bit position, the desired number of shifts is loaded into the CL register, and CL is put in the count position of the instructions.

The flags are affected by SHR as follows: CF contains the bit most recently shifted in from the LSB. For a count of one, OF will be a 1 if the two MSBs are not both 0's. For multiple-bit shifts, OF is meaningless. SF and ZF will be updated to show the condition of the destination. PF will have meaning only for the lower 8 bits of the destination. AF is undefined.

The SHR instruction can be used to divide an unsigned binary number by a power of 2.

Examples:

```
SHR BP, 1               : Shift word in BP one bit position right, 0 in MSB
MOV CL, 03H             : Load desired number of shifts in CL
SHR BYTE PTR[BX]        : Shift byte in DS at offset [BX] 3 bits right 0's in 3
                          MSBs
```

Logical Instructions

AND - AND Corresponding Bits of Two Operands

AND Destination, Source

This instruction ANDs each bit in a source byte or word with the same number bit in a destination byte or word. The result is put in the specified destination. The contents of the specified source will not be changed. The result for each bit position will follow the truth table for a two – input AND gate. In other words, a bit in the specified destination will be a 1 only if that bit is a 1 in both the source and the destination operands. Therefore, a bit can be masked (reset) by ANDing it with 0.

The source operand can be an immediate number, the contents of a register, or the contents of a memory location specified by one of the 24 addressing modes. The destination can be a register or a memory location. The source and the destination cannot both be memory locations in the same instruction. CF and OF are both 0 after AND. PF, SF and ZF are updated by AND. AF is undefined.

Examples:

```
AND CX, [SI]        : AND word in DS at offset [SI] with word in CX register. Result in
```

	CX register.
AND BH, CL	: AND byte in CL with byte in BH. Result in BH

OR – Logically OR Corresponding Bits of Two Operands

OR Destination, Source

This instruction ORs each bit in a source byte or word with the corresponding bit in a destination byte or word. The result is put in the specified destination. The contents of the specified source will not be changed. The result for each bit will follow the truth table for a two-input OR gate. In other words, a bit in the destination will become a 1 if that bit is a 1 in the source operand or that bit is a 1 in the original destination operand. Therefore, a bit in the destination operand can be set to a 1 by simply ORing that bit with a 1 in the same bit of the source operand. A bit ORed with 0 is not changed.

The source operand can be an immediate number, the contents of a register, or the contents of a memory location specified by one of the 24 addressing modes. The destination can be a register or a memory location. The source and the destination cannot both be memory locations in the same instruction.

Examples:

OR AH, CL	: CL ORed with AH, result in AH, CL not changed.
OR BP, SI	: SI ORed with BP, result in BP, SI not changed.
OR SI, BP	: BP ORed with SI, result in SI, BP not changed.
OR BL, 80H	: BL ORed withy immediate 80H, Set MSB of BL to a 1.

NOT – Invert Each Bit of Operand

NOT Destination

The NOT instruction inverts each bit (forms the 1’s complement) of the byte or word at the specified destination. The destination can be a register or a memory location specified by any one of the 24 addressing modes. No flags are affected by the NOT instruction.

NOT BYTE PTR [BX]	: Complement memory byte at offset [BX] in data segment