S.JAGADEESAN

UNIT – II

Arithmetic & Logic Instructions.

→ The arithmetic instructions include addition, subtraction, multiplication, division, comparison, negation, increment and decrement.

Addition.

→ Addition (ADD) appears in many forms in the Microprocessor.

ADD R (Add register to Accumulator)

A ← A+R machine cycle : 1 States : 4

Flags : All, Register addressing, one-byte inst.

→ The content of the operand (register) are added to the content of the accumulator and result is stored in the accumulator.

ADD Destination, SOURCE

Destination ← (Source + Destination).

Flags affected : O S Z A P C

→ Add the contents of source operand (R/M) specified in the instruction or an immediate data to the content of destination operand (R/M)

→ result is in the destination operand.

S. JAGADEESAN

→ both source & destination operands not be memory

→ The condition code flags are affected.

→ The object code is

Register/memory with Register | 0000 00dw | Mod reg R/M |

Immediate to R/M | 1000 00sw | Mod 000 R/M | data | Data. |

Immediate to Accumulator | 0000 010w | data | data |

For ex:- ADD Ax, 0100H Object code = 05, 00, 01

ADD AL, 22H ; Object code = 04, 22

ADD Ax, Bx ; " = 01, D8

ADD AL, [BX] ; " = 02, 07

ADD [BX], CL ; " = 00, 0F

ADD [BX], CX ; " = 01, 0F

ADC Destination, Source

Destination ← (Source + Destination + CF)

→ This instruction performs the same operation as ADD instruction, although the carry flag bit is added with the result.

→ The conditional flags are affected O, S, Z, A, P, C

S. JAGADEESAN

→ The object code is

R/M with Register | 0001 00dw | Mod reg R/M |

Immediate to R/M | 1000 00sw | Mod 010 R/M | data | Data |

Immediate to ACC | 0000 010w | data | data |

For ex:-

ADC Ax, 1234H ; Code = 15, 34, 12

ADC Ax, Cx ; Code = 11, C8

ADC Ax, [SI] ; Code = 13, 04

ADC Ax, [4000] ; Code = 13, 06, 00, 40

ADC SI, Ax ; Code = 11, 04

ADC [4000], Bx ; Code = 11, 1E, 00, 40

8/2/17

Subtraction.

→ Perform subtraction operation & many forms appear in the instruction set.

SUB Destination, Source.

→ This instruction subtracts the source operand (R/M/I) from the destination operand (R/M) and the result is stored in the destination operand.

→ both source and destination operand not a memory.

→ all condition code flags are affected O, S, Z, A, P & C

→ The object code is

R|M with Register | 0010 10dw | Mod reg R|M |

Imm to R|M | 1000 00sw | Mod 101 R|M | data | Data |

Imm to ACC | 0010 110w | data | data |

For Ex:-

SUB AX, 0100; Load 0100H to the AX register immediately.

Object code = 0010 1101, 16-bit data = 2D, 00, 01 as w=1

SUB AL, 44H ; code = 2C, 44

SUB AX, BX ; code = 29, D8

SUB AL, [BX] ; code = 2A, 07

SUB [BX], CL ; code = 28, 0F

SUB [BX], CX ; code = 29, 0F

SBB Destination, Source

Destination ← ((Destination − Source) − CF)

→ Subtracts the Source operand and the borrow flag which is the result of previous operation, from the Destination operand

→ The result is Stored in the destination operand

→ All the flags are affected O, S, Z, A, P & C

→ The object code is

R|M with Register | 0001 10dw | Mod reg R|M |

Imm to reg|M | 1000 00sw | Mod 011 R|M | data | Data |

Imm to ACC | 0000 111w | data | data |

For Ex:- SBB AX, 0010 ; Subtract 0010H & CF from AX register immediately

The object code is | 0000 111w | data | data |

object code = 0000 1111, 16-bit data = ~~CF, 00, 10~~ 0F, 10, 00

SBB AX, BX ; code = 19, D8

SBB AL, [BX] ; code = 1A, 07

SBB [BX], CL ; code = 18, 0F

SBB [BX], CX ; code = 19, 0F

SBB AX, [1000]; code = 1B, 06, 00, 10

Multiplication.

→ multiplication is performed on bytes, words, or doublewords and can be signed (IMUL) or unsigned (MUL) integer.

MUL Source

AX ← (AL * Source8)

DX:AX ← (AX * Source16)

→ This instruction is an unsigned byte or word multiplication by the content of AL or AX.

→ An 8-bit source is multiplied by the content of AL to generate a 16-bit result in AX

S.JAGADEESAN

→ A 16-bit source is multiplied by the content of AX to generate a 32-bit result.

→ The most significant word of the result is stored in DX & the least significant word of the result is stored in AX.

→ All flags are modified depending upon the result

→ The object code is

1111 011w	Mod 100 R/M

The second operand is mod & R/M.
The first " " always AL or AX.

for Ex:- MUL BL ; Assume AL = 22 & BL = 11 H.

Object code = 1111 011w, mod 100 R/M = 1111 0110, 11 100 011.

= F6, E3 as w=0

MUL CL ; code = F6, E1
MUL BX ; code = F7, E3
MUL CX ; code = F7, E1
MUL DX ; code = F7, E2
MUL [BX+10] ; code = F7, 67, 10

IMUL Source

AX ← (AL * Source8)
DX : AX ← (AX * Source16)

S.JAGADEESAN

→ Signed multiplication of two signed numbers

→ A signed byte in source operand is multiplied by the content of AL to generate a 16-bit result in AX.

→ The source can be a general-purpose register, memory operand, index register or base register, but not an immediate data.

→ A 16-bit source operand is multiplied by the contents of AX to generate a 32-bit result.

higher-order word is stored in DX
lower-order word is stored in AX.

→ The AF, PF, SF & ZF are undefined after IMUL

→ If AH & DX contain parts of 16-bit & 32-bit result, CF & OF both will set.

→ AL & AX are implicit operands in case of 8-bit & 16-bit multiplication. The unused higher bits of the result are filled by the sign bit & CF, AF are cleared.

→ The object code is

1111 011w	mod 101 R/M

Mod & R/M → second operand either R/M.
AL or AX → always first operand.

S=0 → +ve number S=1. → −ve number

16-bit Multiplication → 15 bits represent a number

16th bits represent sign.

Ex:- IMUL BL ; object code = 1111 011w, mod.101 R/M =

1111 0110, 11 101 011 = F6, EB

IMUL CL ; code = F6, E9

IMUL BH ; code = F6, EF

IMUL BX ; code = F7, EB

IMUL CX ; code = F7, E9

IMUL DX ; code = F7, EA

IMUL [BX+10] ; code = F7, 2F, 10.

DIV Source (unsigned)

AL ← (AX ÷ Source8) AH ← Remainder.

AX ← (DX:AX ÷ Source16) DX ← Remainder.

→ This instruction is used to divide a 16-bit unsigned number by an 8-bit unsigned number.

→ When a 16-bit number in AX is divided by an 8-bit source operand, the quotient is stored in AL & the remainder is stored in AH

→ If the result is too big to fit in AL, a divide by zero (type 0) interrupt is generated.

→ This instruction to divide a 16-bit unsigned number by a 16-bit or 8-bit operand. The dividend must be in AX for 16-bit operation & divisor may be specified any one of the addressing modes except immediate.

→ A 32-bit number in DX:AX is divided by a 16-bit source with the quotient remaining in AX & the remainder in DX.

→ When a quotient is greater than FFFFH a divide by zero (type 0) interrupt is generated.

→ The object code is

1111 011w	mod 110 R/M.

For Ex:- DIV BL ; BL = 8-bit divisor & AX = 16-bit dividend

object code = 1111 011w, mod 110 R/M = 1111 0110, 11 110 011 = F6, F3 &w=0

DIV CL ; code = F6, F1

DIV BX ; code = F7, F3

DIV CX ; code = F7, F1

DIV DX ; code = F7, F2

DIV [BX+10] ; code = F7, 37, 10

→ This instruction does not affect any flag.

IDIV Source (Signed)

AL ← (AX ÷ SourceB) AH ← Remainder

AX ← (DX:AX ÷ Source16) DX ← remainder.

→ This instruction performs the same as DIV operation

→ A 16-bit value in AX is divided by an 8-bit source with the quotient in AL & the remainder in AH.

→ if the result is too big fit in AL or AX divide by 0 interrupt flag is generated

→ A 32-bit number in DX:AX is divided by 16-bit source with the quotient in AX & the remainder in DX.

→ All flags are undefined after IDIV instruction.

The object code is

1111 011w	mod 111 R/M

for IDIV BL ; In this instruction AX ÷ CL

object code = 1111 011w, mod 111 R/M = 1111 0110, 11 111 011 = F6, FB as w = 0

```
IDIV BH    ; Code = F6, FF
IDIV CL    ; code = F6, F9
IDIV BX    ; code = F7, FB
IDIV CX    ; code = F7, F9
IDIV DX    ; code = F7, FA
IDIV [BX+10] ; code = F7, 3F, 10
```


Comparison.

→ Performs a nondestructive subtraction of source from destination but the result is not stored.

CMP Destination, Source.

→ The source & destination operand are compared

→ The source operand may be R/M/I & the destination operand may be R/M but not an immediate.

→ The result will not be stored. but the flags are affected depending upon the result of subtraction.

→ both source & destination operand are equal ZF = 1

→ Source operand > destination operand ; Cf = 1 Otherwise CF = 0

→ Flags are affected O, S, Z, A, P & C

→ The object code is

R/M with Register

0011 10dw	mod reg R/M

Imm to R/M

1000 00sw	mod reg R/M	data	data

Imm to ACC

0011 110w	data	data

for ex:- CMP AX, 0100 ; compare 0100 with the content of AX

The object code is

0011 1w0w	data	data

object = 0011 110w 16-bit data = 0011 1101, 0100 = 3D, 00, 01H

```
CMP BX,1234   ; code = 81,FB,34,12
CMP AL,22     ; code = 3C,22
CMP BX,[SI]   ; code = 3B,1C
CMP [0100],BX ; code = 39,1E,00,01
CMP [BX],CX   ; code = 39,0F
```

NEGate

→ Performs a complement operation.

NEG Destination (Changes the sign of the operand)

→ This instruction performs a 2's complement of destination.

→ To obtain 2's complement, it subtracts the content of the destination from zero & the result is stored in the destination operand. (R/M)

→ All the condition flags O,S,Z,A,P & C are affected.

→ OF is set, The operation not completed successfully.

→ The object code is

1111 011w	mod 011 R/M.

```
ex:- NEG AX ; code = F7,D8
     NEG BX ; code = F7,DB
     NEG CX ; code = F7,D9
     NEG DX ; code = F7,DA
     NEG CL ; code = F6,D9
     NEG DL ; code = F6,DA.
```

S.JAGADEESAN

Increment & Decrement.

→ This instruction either increment or decrement by 1 either 8-bit or 16-bit value.

INC Destination

Destination ← Destination + 1

→ This instruction is executed, the content of the specified R/M increased by 1

→ The condition flags (O,S,Z,A&P) are affected but the CF is not affected.

→ immediate data cannot be operand.

R/M

1111 111w	mod 000 R/M

Reg.

01 ~~Reg~~ 000	reg

For ex:- INC AX ; code = 01 000 reg = 01 000 000 = 40

```
INC BX   ; code = 43
INC CX   ; code = 41
INC DX   ; code = 42
INC [BX] ; code = FF,07.
```

S. JAGADEESAN

DEC Destination.

Destination ← Destination − 1

→ The content of the specified R|M by 1

→ After execution, all the condition flags are affected (O, S, Z, A & P) depending upon the result.

→ CF is not affected

→ The object code is

R|M | 1111 111w | mod 001 R/M |

Register | 01 001 reg |

Ex:- DEC AX ; The object code = 01001000 = 48

DEC BX ; 4B

DEC CX ; 49

DEC DX ; 4A

DEC [BX] ; FF, 0F.

BCD & ASCII Arithmetic.

→ The microprocessor allows arithmetic manipulation of both BCD (Binary coded Decimal) & ASCII (American Standard Code for Information Interchange)

→ BCD, two arithmetic techniques operate with BCD data: Addition & subtraction. ie DAA & DAS.

S. JAGADEESAN

→ The ASCII arithmetic instructions function with ASCII coded numbers. These numbers range from 30H to 39H for the number 0-9 & 4 instructions are used.

DAA (Decimal Adjust After Addition)

AL ← (AL adjusted for BCD addition)

→ This instruction is used to transfer the result of the addition of two packed BCD numbers to a valid BCD number.

→ The result is stored in AL register only.

→ if after addition, the lower nibble is greater than 9, AF is set, then 06 will be added to the lower nibble in AL.

→ if the upper nibble is greater than 9, CF is set 60H is add to AL.

→ AF, CF, PF & ZF flags are affected.

→ OF is undefined.

The object code is 00100111 = 27H.

Ex:- add two numbers 54 & 26 & use DDA.

1) MOV AL, 54

2) MOV BL, 26

3) ADD AL, BL ; AL = 7A

4) DAA ; 7A + 06 = 80

S.JAGADEESAN

DAS (Decimal Adjust for Subtraction)

AL ← (AL adjusted for BCD subtraction

→ This instruction is used to convert the result of subtraction of two packed BCD numbers to a valid BCD numbers.

→ The subtraction will be stored in AL only.

→ The lower nibble of AL is greater than 9, 06 will be subtracted, CF is set.

→ The upper nibble is greater than 9, 60H will be subtracted from AL

→ AF, CF, SF, PF & ZF are affected.

→ OF is undefined.

→ The object code of DAS is 0010 1111 = 2FH.

S.JAGADEESAN