Lecture 7

UNSIGNED ADDITION AND SUBTRACTION

Addition of Unsigned Numbers

Write a program to calculate the total sum of 5 bytes of data. Each byte represents the daily wages of a worker. This person does not make more than $255 (FFH) a day. The decimal data is as follows: 125, 235, 197, 91 and 48.

```
        .MODEL SMALL
        .STACK 64
        .DATA
COUNT   EQU 05
DATA    DB 125, 235, 197,91,48
        ORG 8H
SUM     DW ?
        .CODE
MAIN    PROC FAR
        MOV AX,@DATA
        MOV DS, AX
        MOV CX, COUNT
        MOV SI, OFFSET DATA
        MOV AX, 00
BACK:   ADD AL, [SI]
        JNC OVER
        INC AH
OVER:   INC SI
        DEC CX
        JNZ BACK
        MOV SUM, AX
        MOV AH, 4CH
        INT 21H
MAIN    ENDP
        END MAIN
```

```
BACK:   ADD AL, [SI]
        JNC OVER
        INC AH
OVER:   INC SI
```

Can be replaced with these lines.

```
BACK:   ADD AL, [SI]
        ADC AH, 00
        INC SI
```

- ADC AH, 00 means add 00+AH+CF and place the result in AH.

Write a program to calculate the total sum of 5 words of data. Each word represents the yearly wages of a worker. This person does not make more than $65,555 (FFFFH) a year. The decimal data is as follows: 27345, 28521, 29533, 30105 and 32375.

```
                .MODEL SMALL
                .STACK 64
                .DATA
COUNT           EQU 05
DATA            DW 27345, 28521, 29533, 30105, 32375
                ORG 10H
SUM             DW 2 DUP(?)
                .CODE
MAIN            PROC FAR
                MOV AX,@DATA
                MOV DS, AX
                MOV CX, COUNT
                MOV SI, OFFSET DATA
                MOV AX, 00
                MOV BX, AX
BACK:           ADD AX, [SI]
                ADC BX,0
                INC SI
                INC SI
                DEC CX
                JNZ BACK
                MOV SUM, AX
                MOV SUM+2, BX
                MOV AH, 4CH
                INT 21H
MAIN            ENDP
                END MAIN
```

Multiword Addition:

- If the numbers being added could be upto 8 bytes wide or more, since the registers are only 16 bits wide (2 bytes), it is the job of the programmer to write the code to break down these large numbers into smaller chunks to be processed by the CPU.
- If the 16 bits register is used and the operand is 8 bytes wide, that would take a total of four iterations.
- If an 8 bits register is used , the same operands would require eight iterations.

Multiword Addition:

```
                .MODEL SMALL
                .STACK 64
                .DATA
COUNT           EQU 04
DATA 1          DQ 548FB9963CE7H
                ORG 10H
DATA2           DQ 3FCD4FA23B8DH
                ORG 20H
DATA3           DQ ?
                .CODE
MAIN            PROC FAR
                MOV AX,@DATA
                MOV DS, AX
                CLC             ; Clear carry before first addition
                MOV CX, COUNT
                MOV SI, OFFSET DATA1
                MOV DI, OFFSET DATA2
                MOV BX, OFFSET DATA3
BACK:           MOV AX, [SI]
                ADC AX, [DI]
                MOV  [BX], AX
                ADD SI, 2
                ADD DI, 2
                ADD BX, 2
                LOOP BACK
                MOV AH, 4CH
                INT 21H
MAIN            ENDP
                END MAIN
```

- Here is new instruction in the program that is LOOP BACK, which replaces the following instructions:

  DEC CX

  JNZ BACK

- When LOOP BACK is executed, CX is decremented automatically, and if CX is not equal to zero, the microprocessor will jump to target address BACK.

UNSIGNED ADDITION AND SUBTRACTION

Subtraction of Unsigned Numbers

Subtraction of Unsigned Numbers:

- In subtraction, the x86 microprocessor (indeed, almost all modern CPUs) use the 2's complement method.
- CPU executes the SUB instruction for unsigned numbers as follows:
  - Take the 2's complement of the source operand.
  - Add it to the destination operand.
  - Invert the carry.

Syntax:

```
SUB destination, source          ; destination = destination-source
```

Subtraction of Unsigned Numbers:

Example: Show the steps involved in the following:

```
MOV AL, 3FH
MOV BH, 23H
SUB AL, BH
```

Solution:

AL	3F	0011 1111	0011 1111	
-BH	-23	-0010 0011	+ 1101 1101	(2's complement)
	1C		1 0001 1100	(CF=0) ;step 3

Note: The programmer must look at the carry flag (not the sign flag) to determine if the result is positive or negative.

After the execution of SUB, if CF = 0, the result is positive; if CF = 1, the result is negative and the destination has the 2's complement of the result.

Subtraction of Unsigned Numbers:

Example: Analyze the following program:

```
;from the data segment
DATA1   DB      4CH
DATA2   DB      6EH
DATA3   DB      ?
;from the code segment
                MOV DH, DATA1
                SUB DH, DATA2
                JNC NEXT
                NOT DH
                INC DH
NEXT:           MOV DATA3, DH
```

Solution:

```
 4C     0100 1100      0100 1100
-6E     0110 1110     +1001 0010  (2's complement)
-22                   0 1101 1110  (CF=1) ;step3 (result is negative)
```

Subtraction of Unsigned Numbers:

SBB (Subtract with Borrow):

- This instruction is used for multibyte (multiword) numbers and will take care of the borrow of the lower operand.
- If the carry flag is zero, SBB works like SUB. If the carry flag is 1, SBB subtracts 1 from result.

Subtraction of Unsigned Numbers:

Example: Analyze the following program:

```
DATA_A  DD      62562FAH
DATA_B  DD      412963BH
RESULT  DD      ?
. . . .                 . . . .         . . . .
MOV     AX, WORD PTR DATA_A
SUB     AX, WORD PTR DATA_B
MOV     WORD PTR RESULT, AX
MOV     AX, WORD PTR DATA_A+2
SBB     AX, WORD PTR DATA_B+2
MOV     WORD PTR RESULT+2, AX
```

Solution:

After the SUB, AX=62FA – 963B = CCBF and the carry flag is set. Since CF=1, when SBB is executed, AX=0625 – 0412 – 1 = 212. Therefore, the value stored in result is 0212CCBF.

- Notice the PTR operand in given example.
- The PTR (pointer) data specifier directive is widely used to specify the size of the operand when it differs from the defined size.
- In given example, WORD PTR tells the assembler to use a word operand, even though the data is defined as a double word.

UNSIGNED MULTIPLICATION AND DIVISION

Multiplication of Unsigned Numbers

Multiplication of Unsigned Numbers:

- In discussing multiplication, the following cases will be examined:
  - Byte times byte.
  - Word times word.
  - Byte times word.

Table 1: Unsigned Multiplication Summary

Multiplication	Operand 1	Operand 2	Result
byte x byte	AL	Register or memory	AX
word x word	AX	Register or memory	DX AX
word x byte	AL=byte, AH=0	Register or memory	DX AX

Multiplication of Unsigned Numbers:

Byte x Byte:

In byte by byte multiplication, one of the operands must be in the AL register and the second operand can be either in a register or a memory. After multiplication, the result is saved in AX.

Example:

```
DATA1       DB      25H
DATA2       DB      65H
RESULT1     DW      ?               ; result is defined in DS
            . . . .
            MOV AL, DATA1
            MOV BL, DATA2
            MUL BL                  ; AX = 25H x 65H
            MOV RESULT1, AX         ; the result is saved.
```

Multiplication of Unsigned Numbers:

Word x Word:

In word by word multiplication, one operand must be in AX, and the second operand can be in a register or memory. After the multiplication, registers AX and DX will contain the result. Since it can produce 32 bits result, AX will hold the lower word and DX the higher word.

Example:

```
DATA3       DW      2378H
DATA4       DW      2F79H
RESULT2     DW 2 DUP(?)
            . . . .
            MOV AX, DATA3
            MUL DATA4
            MOV RESULT2, AX       ; store the lower word result
            MOV RESULT2+2, DX     ; store the higher word result
```

Multiplication of Unsigned Numbers:

Word x Byte:

This is similar to word by word multiplication except that the AL contains the byte operand and AH must be zero.

Example:

```
DATA5       DB      6BH
DATA6       DW      12C3H
RESULT3     DW 2 DUP(?)
            . . . .
            MOV AL, DATA5 ; AL holds the byte operand
            SUB AH, AH      ; AH must be cleared
            MUL DATA6       ; byte in AL multiply by word operand
            MOV BX, OFFSET RESULT3      ; BX points to product
            MOV [BX], AX                ; AX holds the lower word
            MOV [BX]+2, DX              ; DX holds the higher word
```

UNSIGNED MULTIPLICATION AND DIVISION

Division of Unsigned Numbers

Division of Unsigned Numbers:

- In the division of unsigned numbers, the following cases are discussed:
  - Byte over byte
  - Word over word
  - Word over byte
  - Double word over word
- In divide, there could be cases where the CPU cannot perform the division. In these cases interrupt is activated. This is referred to as an exception. The cases are defined below:
  - If the denominator is zero.
  - If the quotient is too large for the assigned register.
- In the IBM PC and compatibles, if either of these cases happens, the PC will display the ‘divide error’ message.

Division of Unsigned Numbers:

Table 2: Unsigned Division Summary

Division	Numerator	Denominator	Quotient	Remainder
byte/byte	AL = byte, AH = 0	Register or memory	AL*	AH
word/word	AX = word, DX = 0	Register or memory	AX**	DX
word/byte	AX = word	Register or memory	AL*	AH
doubleword/ word	DXAX = doubleword	Register or memory	AX**	DX

* Divide error interrupt if AL > FFH.

** Divide error interrupt if AX > FFFFH

Division of Unsigned Numbers:

Byte/byte:

In dividing a byte by a byte, the numerator must be in the AL register and AH must be set to zero. The denominator cannot be immediate but can be in a register or memory. After the DIV instruction is performed, the quotient is in AL and the remainder is in AH.

Example:

```
DATA7       DB      95
DATA8       DB      10
QOUT1       DB      ?
REMAIN1     DB      ?
            . . . .
            MOV AL, DATA7   ; AL holds numerator
            SUB AH, AH      ; AH must be cleared
            DIV DATA8       ; divide AX by DATA8
            MOV QOUT1, AL   ; quotient = AL = 09
            MOV REMAIN1, AH       , remainder = AH = 05
```

Division of Unsigned Numbers:

Word/word:

In this case, the numerator is in AX and DX must be cleared. The denominator must be in a register or memory. After the DIV, AX will have the quotient and the remainder will be in DX.

Example:

```
MOV AX, 10050        ; AX holds numerator
SUB DX, DX           ; DX must be cleared
MOV BX, 100          ; BX used for denominator
DIV BX
MOV QOUT2, AX        ; QOUT2 = AX = 64H = 100
MOV REMAIN2, DX      ; REMAIN2 = DX = 32H = 50
```

Division of Unsigned Numbers:

Word/byte:

In this case, the numerator is in AX and the denominator can be in a register or memory. After the DIV instruction, AL will contain the quotient and AH will contain the remainder. The maximum quotient is FFH.

Example:

```
MOV AX, 2055        ; AX holds numerator
MOV CL, 100         ; CL holds denominator
DIV CL
MOV QUO, AL         ; AL holds quotient
MOV REMI, AH        ; AH holds remainder
```

Division of Unsigned Numbers:

Doubleword/word:

The numerator is in AX and DX, with the most significant word in DX and the least significant word in AX. The denominator can be in a register or memory. After the DIV instruction, the quotient will be in AX, and the remainder in DX. The maximum quotient is FFFFH.

Example:

```
DATA1   DD      105432
DATA2   DW      10000
QUOT            DW      ?
REMAINDW        ?
                . . . .
        MOV AX, WORD PTR DATA1      ; AX holds lower word of numerator
        MOV DX, WORD PTR DATA1+2    ; DX holds higher word of numerator
        DIV DATA2
        MOV QUOT, AX                ; AX holds quotient
        MOV REMAIN, DX              ; DX holds remainder
```

LOGIC INSTRUCTIONS

AND Instruction:

Syntax:

AND destination, source

This instruction will perform a logical AND on the operands and place the result in the destination. The destination operand can be a register or in memory. The source operand can be a register, in memory or immediate.

AND will automatically change the CF and OF to zero, and PF, ZF and SF are set according to the result.

Table 3: Logical AND Function

A	B	A . B
0	0	0
0	1	0
1	0	0
1	1	1

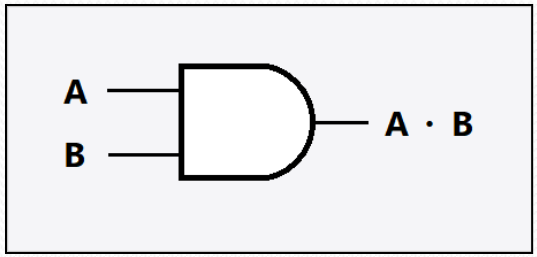

Figure 1: AND Gate Symbol

AND Instruction:

Example: Show the result of the following:

```
MOV BL, 35H
AND BL, 0FH
```

Solution:

35H	0011 0101
0FH	0000 1111
05H	0000 0101

Flag settings will be: SF = 0, ZF = 0, PF = 1, CF = OF = 0

OR Instruction:

Syntax:

OR destination, source

OR can be used to set certain bits of an operand to 1. The destination operand can be a register or in memory. The source operand can be a register, memory or immediate.

The flags will be set the same as for AND instruction.

Table 4: Logical OR Function

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

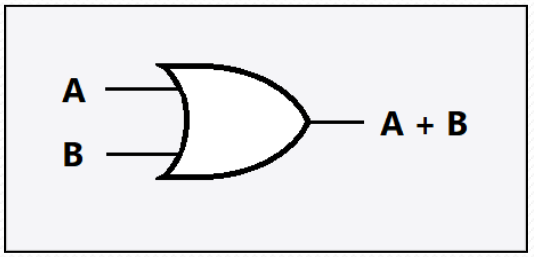

Figure 2: OR Gate Symbol

OR Instruction:

Example: Show the result of the following:

MOV AX, 0504H

OR AX, DA68H

Solution:

0504H	0000 0101 0000 0100
DA68H	1101 1010 0110 1000
DF6CH	1101 1111 0110 1100

Flags will be: SF = 1, ZF = 0, PF = 1, CF = OF = 0.

Notice that the parity is checked for the lower 8 bits only.

XOR Instruction:

Syntax:

XOR destination, source

XOR set the result bits to 1 if they are not equal; otherwise they are reset to zero. The flags are set the same as for AND instruction.

XOR can also be used to see if two registers have the same value.

'XOR BX, CX' will make ZF = 1 if both registers have same value.

Table 5: Logical XOR Function

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

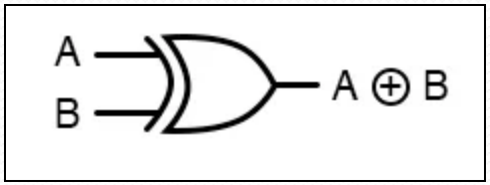

Figure 3: XOR Gate Symbol

XOR Instruction:

Example: Show the result of the following:

MOV DH, 54H

XOR DH, 78H

Solution:

54H	0101 0100
78H	0111 1000
2CH	0010 1100

Flags setting will be: SF = 0, ZF = 0, PF = 1, CF = OF = 0