ARITHMETIC AND BOOLEAN OPERATORS

Lecturers: ARD, SWJ Lab Tutors: VDE, WUL

OBJECTIVES

1. Able to use Arithmetic and Boolean operators
2. Able to define flags in assembly language
3. Able to create a simple program using arithmetic and Boolean operators
4. Able to use assembly instructions, such as: INC, DEC, SUB, ADD, MUL, DIV, NEG, NOT, OR, XOR

THEORITICAL REVIEW

INC and DEC Operation

The INC (increment) and DEC (decrement) instructions, add 1 and subtract 1 from a register or memory operand, respectively. The syntax is

```
INC reg/mem
DEC reg/mem
```

Some examples are as follows:

```
.data
myWord WORD 1000h
.text
inc myWord       ; myWord = 1001h
mov bx, myWord
dec bx     ; BX = 1000h
```

The Overflow, Sign, Zero, Auxiliary Carry, and Parity flags are changed according to the value of the destination operand. The INC and DEC instructions do not affect the Carry flag.

ADD Instruction

The ADD instruction adds a source operand to a destination operand of the same size. The syntax is

```
ADD dest, source
```

Source is unchanged by the operation, and the sum is stored in the destination operand. Here is a short code example that adds two 32-bit integers:

```
.data
var1 DWORD 10000h
var2 DWORD 20000h
.text
mov eax,[var1] ; EAX = 10000h
add eax,[var2] ; EAX = 30000h106
```

The Carry, Zero, Sign, Overflow, Auxiliary Carry, and Parity flags are changed according to the value that is placed in the destination operand.

SUB Instruction

The SUB instruction subtracts a source operand from a destination operand. The set of possible operands is the same as for the ADD and MOV instructions. The syntax is

```
SUB dest,source
```

Here is a short code example that subtracts two 32-bit integers:

```
.data
var1 DWORD 30000h
var2 DWORD 10000h
.text
mov eax,[var1] ; EAX = 30000h
sub eax,[var2] ; EAX = 20000h
```

Flags The Carry, Zero, Sign, Overflow, Auxiliary Carry, and Parity flags are changed according to the value that is placed in the destination operand.

NEG Instruction

The NEG (negate) instruction reverses the sign of a number by converting the number to its two's complement. The following operands are permitted:

```
NEG reg
NEG mem
```

MUL/IMUL Instruction

There are two instructions for multiplying binary data. The MUL (Multiply) instruction handles unsigned data and the IMUL (Integer Multiply) handles signed data. Both instructions affect the Carry and Overflow flag.

Multiplicand	**Multiplier**	**Product**
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

The following statements multiply AL by BL, storing the product in AX. The Carry flag is clear (CF = 0) because AH (the upper half of the product) equals zero:

```
mov al,5h
mov bl,10h
mul bl ; AX = 0050h, CF = 0
```

DIV/IDIV Instruction

The division operation generates two elements - a quotient and a remainder. In case of multiplication, overflow does not occur because double-length registers are used to keep the product. However, in case of division, overflow may occur. The processor generates an interrupt if overflow occurs. The DIV (Divide) instruction is used for unsigned data and the IDIV (Integer Divide) is used or signed data.

Dividend	**Divisor**	**Quotient**	**Remainder**
AX	reg/mem8	AL	AH
DX:AX	reg/mem16	AX	DX
EDX:EAX	reg/mem32	EAX	EDX

The following instructions perform 16-bit unsigned division (8003h/100h), producing a quotient of 80h and a remainder of 3. DX contains the high part of the dividend, so it must be cleared before the DIV instruction executes:

```
mov dx,0 ; clear dividend, high
mov ax,8003h ; dividend, low
mov cx,100h ; divisor
div cx ; AX = 0080h, DX = 0003h
```

AND Instruction

The AND instruction performs a boolean (bitwise) AND operation between each pair of matching bits in two operands and places the result in the destination operand: AND destination,source The following operand combinations are permitted, although immediate opperands can be no larger than 32 bits:

```
AND reg,reg
AND reg,mem
AND reg,imm
AND mem,reg
AND mem,imm
```

For example, suppose AL is initially set to 10101110 binary. After ANDing it with 11110110, AL equals 10100110:

```
mov al,10101110b
and al,11110110b  ; result in AL = 10100110
```

OR Instruction

The OR instruction performs a boolean OR operation between each pair of matching bits in two operands and places the result in the destination operand:

```
OR destination,source
```

The OR instruction uses the same operand combinations as the AND instruction:

```
OR reg,reg
OR reg,mem
OR reg,imm
OR mem,reg
OR mem,imm
```

For example, if AL is initially equal to 11100011 binary and then we OR it with 00000100, the result equals 11100111:

```
mov al,11100011b
or al,00000100b   ; result in AL = 11100111
```

The OR instruction always clears the Carry and Overflow flags. It modifies the Sign, Zero, and Parity flags in a way that is consistent with the value assigned to the destination operand.

XOR Instruction

The XOR instruction performs a boolean exclusive-OR operation between each pair of matching bits in two operands and stores the result in the destination operand:

```
XOR destination,source
```

The XOR instruction always clears the Overflow and Carry flags. XOR modifies the Sign, Zero, and Parity flags in a way that is consistent with the value assigned to the destination operand.

NOT Instruction

The NOT instruction toggles (inverts) all bits in an operand. The result is called the one’s complement. The following operand types are permitted: NOT reg NOT mem For example, the one’s complement of F0h is 0Fh:

```
mov al,11110000b
not al     ; AL = 00001111b
```

No flags are affected by the NOT instruction.

FLAGS

- The Carry flag indicates unsigned integer overflow. For example, if an instruction has an 8-bit destination operand but the instruction generates a result larger than 11111111 binary, the Carry flag is set.
- The Overflow flag indicates signed integer overflow. For example, if an instruction has a 16- bit destination operand but it generates a negative result smaller than 32,768 decimal, the Overflow flag is set.
- The Zero flag indicates that an operation produced zero. For example, if an operand is subtracted from another of equal value, the Zero flag is set.
- The Sign flag indicates that an operation produced a negative result. If the most significant bit (MSB) of the destination operand is set, the Sign flag is set.
- The Parity flag indicates whether or not an even number of 1 bits occurs in the least signifi- cant byte of the destination operand, immediately after an arithmetic or boolean instruction has executed.
- The Auxiliary Carry flag is set when a 1 bit carries out of position 3 in the least significant byte of the destination operand.

Here is sample code for adding and subtracting:

```
section       .text
    global main   ;must be declared for using gcc
main:    ;tell linker entry point
       mov    eax, [a] ;'3'
       sub     eax, [b]  ;'0'
       mov    ebx, [c] ;'4'
       sub     ebx, [b] ;'0'
       add    eax, ebx
       add    eax, [b] ;'0'
```

```
        ;add eax, 0x30
        mov    [sum], ax
        mov    ecx, msg
        mov    edx, len
        mov    ebx,1  ;file descriptor (stdout)
        mov    eax,4  ;system call number (sys_write)
        int    0x80   ;call kernel
        mov    ecx, sum
        mov    edx, 1
        mov    ebx,1  ;file descriptor (stdout)
        mov    eax,4  ;system call number (sys_write)
        int    0x80   ;call kernel
        mov  eax,1       ; 1 is the exit syscall number
        mov  ebx,0       ; the status value to return
        int  0x80        ; execute a system call

section .bss
        sum resb 1

section .data
        a dq 0x3
        b dq 0x0
        c dq 0x4
        msg db "The sum is:", 0xA,0xD
        len equ $ - msg
```

TASKS

1. What will be the value of BX and flags affected after the following instructions execute?
   ```
   mov bx,0xFFFF
   and bx,6Bh
   ```
2. Create a program for calculating this formula:
   C = (A+B)/D
   with A= 3, B= 7, D= 4
3. Create a program for calculating this formula:
   C = (A-B)/(D+A)
   with A= 3, B= 7, D= 4
4. Write code using byte operands that adds two negative integers and causes the Overflow flag to be set.
5. Write an assembly language program to compute the average of 4 grades. Use memory locations for the 4 grades. Make the grades all different numbers from 0 to 100. Store the average of the 4 grades in memory and also store the remainder from the division in memory
6. Write an assembly language program to compute the squared distance between 2 points in the 2D plane identified as 2 integer coordinates each, stored in memory. Remember the Pythagorean Theorem!

REFERENCES

[1] Seyfarth R. 2012. Introduction to 64 Bit Intel Assembly Language Programming for Linux. CreateSpace
[2] Irvine K. 2011. Assembly Language for X86 Processors. Pearson Education, Limited [Online]. Available: http://books.google.co.id/books?id=0k20RAAACAAJ
[3] Cavanagh J. 2013, X86 Assembly Language and C Fundamentals. CRC Press, 2013.
[4] http://www.tutorialspoint.com/assembly_programming