LABORATORY NO. 5
ARITHMETICAL, LOGICAL, ROTATE AND SHIFT INSTRUCTIONS FOR 80X86 MICROPROCESSOR

1. Object of laboratory

The purpose of this lab is to present the arithmetical, logical, rotate and shift instructions for x86 microprocessors.

2. Theoretical considerations

The 16 bits microprocessors of the x86 family have computing instructions to allow operations on 8 or 16 bits and to implement routines for multiple bytes or multiword operations. By computing we mean:

- arithmetical operations: add, subtract, multiply, divide, increment, decrement, complement of 2 and compare;
- logical operations: and, or, xor, complement of 1 and test;
- rotate and shift operations.

2.1. Arithmetical instructions

Arithmetical operations are using numbers in byte or word size, in unsigned or C2 representation. Add and subtract operations can also use operands of type BCD unpacked (one decimal digit per byte) or packed BCD (two decimal digits per byte). Multiply and divide operations can be used also for unpacked BCD. In the following table <s> and <d> represent the „source” operand and „destination” operand. Arithmetical instructions generally affect the following flags: AF, CF, OF, DF, PF, ZF. These flags are generally set according to the result of the instruction.

General form	The effect	Affected flags
ADD <d>, <s>	<d> ← {<d>}+{<s>}	AF,CF,OF,PF,SF,ZF
ADC <d>, <s>	<d> ← {<d>}+{<s>}+{CF}	AF,CF,OF,PF,SF,ZF
INC <d>	<d> ← {<d>}+1	AF,OF,PF,SF,ZF

AAA	Decimal correction after addition in unpacked BCD (implicit AL)	AF,CF unmodified OF,PF,SF,ZF undefined
DAA	Decimal correction after addition in packed BCD (implicit AL)	AF,CF,PF,SF,ZF modified OF undefined
SUB <d>, <s>	<d> ← {<d>}-{<s>}	AF,CF,OF,PF,SF,ZF
SBB <d>, <s>	<d> ← {<d>}-{<s>}-{CF}	AF,CF,OF,PF,SF,ZF
CMP <d>, <s>	Only flags are set according to d-s, result is not stored	AF,CF,OF,PF,SF,ZF
DEC <d>	<d> ← {<d>}–1	AF,OF,PF,SF,ZF
NEG <d>	<d> ← [0-{<d>}]	AF,CF,OF,PF,SF,ZF
AAS	Decimal correction after subtraction in unpacked BCD (implicit AL)	AF,CF modified OF,PF,SF,ZF undefined
DAS	Decimal correction after subtraction in packed BCD (implicit AL)	AF,CF,PF,ZF,SF modified OF undefined
CBW	Conversion from byte stored in AL to word stored in AX (sign extension)	---
CWD	Conversion from byte stored in AX to double word stored in DX, AX (sign extension)	---

MUL <s>	if <s> is a byte:
	AX ← (AL)*{<s>}
	if <s> is a word:
	DX, AX ← (AX)*{<s>}
	Operands are handled as unsigned integer
		CF,OF modified
		AF,PF,SF,ZF undefined
		If CF and OF are 1 then AH (resp. DX) store values different from 0

IMUL <s>	if <s> is a byte:
	AX ← (AL)*{<s>}
	if <s> is a word:
	DX:AX ← (AX)*{<s>}
	Operands are handled as signed integer.
		CF,OF modified
		AF,PF,SF,ZF undefined
		If CF and OF are 1 then AH (resp. DX) store values different from 0

AAM	Decimal correction after	SF,PF,ZF modified

	multiplication in BCD unpacked. MUL is used to multiply and then correction is set. AX stores the result.	OF,AF,CF undefined

DIV <s>	if <s> is a byte:
	AL ← [(AX)/{<s>}]
	AH ← (AX) mod {<s>}
	if <s> is a word:
	AX ← [(DX,AX)/{<s>}]
	DX ← (DX,AX) mod {<s>}
	Operands are handled as unsigned integer.
	If the quotient exceeds destination's capacity a level 0 interrupt will be generated.
		AF,CF,OF,PF,SF,ZF undefined

IDIV <s>	if <s> is a byte:
	AL ← [(AX)/{<s>}]
	AH ← (AX) mod {<s>}
	if <s> is a word:
	AX ← [(DX,AX)/{<s>}]
	DX ← (DX,AX) mod {<s>}
	Operands are handled as signed integer.
	If the quotient exceeds destination's capacity a level 0 interrupt will be generated.
		AF,CF,OF,PF,SF,ZF undefined

AAD	Decimal correction before a division in BCD unpacked. The correction is made and then DIV is used for division.	PF,SF,ZF modified AF,CF,OF undefined

The operands involved in addition or subtraction are unsigned integers or signed integers represented in C2. The developer of the program must choose how to represent the operands, how to evaluate the result properly and take efficient actions in case of overflow.

An incorrect result, for unsigned operands, can be checked by testing the value of the CF set by the operation. For signed operands the error can be checked by examining the value in the OF.

Overflow can be tested through conditional jump instructions JC, JNC, JO, JNO for handling errors.

Example:

```
DATA        SEGMENT
MEM8        DB    39
DATA        ENDS


CODE SEGMENT
;... ...                                          unsigned   signed
         MOV AL, 26  ;load al                        26        26
         INC AL      ;increment al                    1         1
         ADD AL, 76  ;add immediate date             76        76
                     ;                              ----      ----
                     ;                              103       103
         ADD AL, MEM8;add memory                     39        39
                     ;                              ----      ----
                     ;                              142       -114+OF
         MOV AH, AL  ;copy to ah                    142
         ADD AL, AH  ;add  register                 142
                     ;                              ----
                     ;                              28+CF
;... ...
CODE ENDS
```

For this example the add operation was on 8 bits. When the sum is over 127 the OF is written, when over 255 the CF is written.

Example:

```
DATA        SEGMENT
MEM8        DB      122
DATA        ENDS

CODE SEGMENT
;                                                   UNSIGNED      SIGNED
;… …
        MOV AL, 95  ;load  al                         95            95
        DEC AL      ;decrement                        -1            -1
        SUB AL, 23  ;subtract immediate value         -23           -23
                                                      ----          ----
                                                      71            71
        SUB AL, MEM8        ;subtract memory          -122          -122
                                                      ----          ----
                                                      205+CF        -51
        MOV AH, 119;load ah
        SUB AL,AH   ;subtract register                              -119
                                                                    ----
                                                                    86+OF
;… …
CODE ENDS
```

The instructions ADC and SBB allow implementation for multi-byte or multi-word operations. They perform the action of ADD and SUB and also add or subtract the value of CF indicator.

Example:

```
DATA        SEGMENT
MEM32       DD      316423
DATA        ENDS

CODE SEGMENT
… …
        MOV AX, 43981
        SUB DX, DX                         ;load dx, ax          43981
        ADD  AX, WORD PTR MEM32[0]  ;add inf. word
        ADC  DX, WORD PTR MEM32[2]  ;add sup. word         316423
                                           ;                     --------
                                           ;result in dx:ax      360404
… …
```

```
CODE ENDS
        Example:

DATA         SEGMENT
MEM32A       DD     316423
MEM32B       DD     156739
DATA         ENDS

CODE SEGMENT
… …
        MOV AX, WORD PTR MEM32A[0]          ;load inf. word
        MOV DX, WORD PTR MEM32A[2]          ; load sup. word
        SUB AX, WORD PTR MEM32B[0]          ;subtract inf. word
        SBB DX, WORD PTR MEM32B[2]          ; subtract sup. word
… …
CODE ENDS
```

MUL is used for multiplying unsigned numbers. IMUL is used for multiplying signed numbers. The syntaxes are :

```
MUL   {register | memory}
IMUL  {register | memory}
```

For multiplication one of the operands must be loaded in the accumulator register (AL for 8 bits operands and AX for 16 bits operands). This is an implicit register, it is not specified in the instruction. The information stored in this register will be destroyed by the result. The second operand must be specified as an operand in register or memory. This operand will not be destroyed by the operation unless it is DX, AH or AL. Multiplying two 8 bits numbers leads to a 16 bits result stored in AX. Multiplying two 16 bits numbers leads to a 32 bits result stored in DX, AX. For both cases if the high part of the result is 0 (for unsigned MUL) or it coincides with the sign extension (for IMUL in sign representation), the indicators CF and OF are set on 0; otherwise are set on 1. The other indicators have undefined values.

```
        Example:
DATA         SEGMENT
MEM16        DW     –30000
DATA         ENDS
```

```
CODE SEGMENT
                        ;unsigned multiply on 8 bits
      MOV AL, 23        ;load al                    23
      MOV BL, 24        ;load bl                    24
      MUL BL            ;multiply with bl
                        ;                           ----
                        ;result in ax               552
                        ; CF and OF are set
                        ;multiply with sign on 16 bits
      MOV AX, 50        ;load ax                    50
      IMUL MEM16        ;multiply with mem.         -30000
                        ;                           --------
                        ;result in dx,ax            -1500000
                        ; CF and OF are set
… …
CODE ENDS
```

DIV instruction is used for dividing unsigned numbers; IDIV is used for signed values. The syntaxes are:

```
DIV     {register | memory}
IDIV    {register | memory}
```

In order to divide a 16 bits number by an 8 bits number the first operand is loaded in AX. The result overwrites the content of AX. If the divider is on 8 bits, register or memory location, after the division AL holds the quotient and AH the rest.

In order to divide a 32 bit number by a 16 bit number the first operand is loaded in the pair DX: AX. The information stored in DX and AX will be lost after the operation. After the division AX stores the quotient and DX the rest.

For dividing 2 numbers of equal length (8 or 16 bits) the first action is to convert to a double length (16 or 32 bits) the first operand. For unsigned numbers the conversion consists in deleting the upper byte of the first operand, register AH, and respectively the most significant word, register DX. For sign numbers conversion consists in sign extension and is obtained through CWB and CWD instructions.

If the divider is 0 or the quotient exceeds it’s assigned register (AL or AX) then the processor generates a level 0 interruption. If this interruption is not handled by the developer the operating system will abandon the program. There are two methods for dealing with the situation: testing the divider before the operation takes place and calling, when needed, a routine for handling errors; writing your own routine for handling errors to replace the routine for level 0 interruption.

Example:

```
DATA        SEGMENT
MEM16       DW    –2000
MEM32       DD    500000
DATA        ENDS

CODE SEGMENT
                        ; unsigned  division of a 16 bits operand
                        ;by an 8 bits operand
      MOV AX, 700       ;load  operand                  700
      MOV BL, 36        ;load divider                   36
      DIV BL            ;unsigned division
                        ;quotient is in al              19
                        ;rest is in ah                  16
                        ;
                        ;signed division of  a 32 bits operand
                        ;by a 16 bits operand
      MOV AX, WORD PTR MEM32[0]  ;load ax
      MOV DX, WORD PTR MEM32[2]  ; load dx              500000
      IDIV MEM16        ; signed division
                        ; quotient is in ax             -250
                        ; rest is in dx                 0
                        ; signed division of  a 16 bit operand
                        ; by a 16 bit operand

      MOV AX, WORD PTR MEM16      ; load operand        -2000
      CWD                         ;convert to double word
      MOV BX, -421                ; load divider        -421
      IDIV BX                     ; signed division
                                  ; quotient is in ax   4
                                  ; rest is in dx       -316
CODE ENDS
```

2.2. Operations in unpacked BCD:

The instruction set has 4 instructions for unpacked BCD or ASCII correction: AAA (ASCII Adjust after Addition), AAS (ASCII Adjust after Subtraction), AAM (Ascii Adjust after Multiplication) and AAD (ASCII Adjust **before** Division), these instruction will correct the result to unpacked BCD format.. Arithmetical operations are computed on byte size operands only. The result must be in AL register implicitly used by the adjust instructions. If an operation implies 2 one digit operands with a result of two digits, the adjust instruction for correction will place the least significant digit in AL, and the most significant in AH. If the result stored in AL generates carry to AH or needs to borrow from AH the flags CF and AF are set.

Example:

```
        ; unpacked BCD addition
MOV AX, 9           ;load ax                    0009h
MOV BX,3            ; load bx                   0003h
ADD AL, BL          ;addition                   000ch
AAA                 ;adjust after addition      ax=0102h
                    ; AF and CF are set
        ;unpacked BCD subtraction
MOV AX, 0103H       ;load ax                    0103h
MOV BX, 4           ; load bx                   0004h
SUB AL, BL          ;subtract                   01feh
AAS                 ; adjust after
                    ;subtraction                ax=0009h
                    ;AF and CF are positioned

        ;unpacked  BCD multiplication
MOV AX, 0903H               ; load ax           0903h
MUL AH            ;unsigned multiplication      001bh
AAM               ; adjust after MUL            ax=0207h

        ;unpacked BCD division
MOV AX, 0205H       ; load ax with dividend     25 unpBCD
MOV BL, 02          ; load bl with divisor      2 unpBCD
AAD                 ; adjust before
                    ;division                   AX=0019H
DIV BL              ;unsigned division          result is 010CH
```

```
                        ;quotient in al                  0CH
                        ;rest in ah                      01H
        AAM             ;adjust after
                        ;division the quotient  ax=0102H  12unpBCD
                        ;the rest is lost
```

The rest will be lost. If needed, it must be saved in a different register before adjusting the quotient. The rest can also be corrected. For this it should be moved in AL.

2.3. Operations in packed BCD

The instruction set has two instructions for decimal correction DAA (decimal adjust after addition) and DAS (decimal adjust after subtraction) which allow adding and subtracting in packed BCD. ADD and SUB instructions are used to add and subtract followed by appropriate instructions to adjust the result.

Arithmetical operations must be on byte size in order to store the result in AL.

Instructions for decimal corrections in packed BCD never affect AH register. AF indicator is positioned in case of carry or borrow from the least significant digit to the most significant one. CF indicator is positioned in case of carry or borrow to exterior.

Example:

```
                ;Adding in packed BCD
        MOV AX, 8833H           ;load ax                   8833H
        ADD AL, AH              ;add to al               al=0BBH
        DAA             ;decimal adjust
                        ;after adding                    al=021H
                        ; CF is set
                        ;the result is     121H = 121 pBCD
                ;Subtracting in packed BCD
        MOV AX, 3883H           ;load ax                  3883H
        SUB AL, AH      ;subtract                         al=04BH
        DAS             ;decimal adjust
                        ;after subtraction                al=045H
                        ; CF is 0
```

2.4. Logical instructions

General form	Effect	Affected conditioning indicators
AND <d>, <s>	<d> ← {<d>} and {<s>}	CF,OF,PF,SF,ZF set AF undefined
TEST <d>, <s>	The indicators are set as for AND but {<d>} does not change	CF,OF,PF,SF,ZF set AF unmodified
OR <d>, <s>	<d> ← {<d>} or {<s>}	CF,OF,PF,SF,ZF set AF undefined
XOR <d>, <s>	<d> ← {<d>} xor {<s>}	CF,OF,PF,SF,ZF set AF undefined
NOT <s>	<s> ← not <s> (complemental to 1)	----

Logical instructions operate on bits, over bits of same rank of two operands. There are 5 logic instructions: AND, TEST, OR, XOR and NOT.

The syntax:

```
AND   {register | memory}, { register | memory | immediate date}
TEST  { register | memory }, { register | memory | immediate date }
OR    { register | memory }, { register | memory | immediate date }
XOR   { register | memory }, { register | memory | immediate date }
NOT   { register | memory }
```

Example:

```
        ;example for  AND
MOV AL, 35H         ;load al                      00110101
AND AL, 0FBH        ;and with immediate value     11111011
                    ;                             ------------
                    ;                             00110001
AND AL, 0F8H        ;                             11111000
                    ;                             ------------
                    ;                             00110000
        ;example for OR
MOV AL, 35H          ;load al                   00110101
OR AL, 08H           ;or with immediate value   00001000
                     ;                          ------------
        OR AL, 07H   ; or with immediate value  00000111
                     ;                          ------------
                     ;                          00111111
        ;example for XOR
MOV AL, 35H          ;load al                   00110101
XOR AL, 08H          ;xor with immediate value  00001000
                     ;                          ------------
                     ;                          00111101

XOR AL, 07H          ; xor with immediate value 00000111
                     ;                          ------------
                     ;                          00111010
```

Logical instructions can be used to compare an operand with 0 (OR BX, BX instead of CMP BX, 00) or to initialize with 0 (XOR CX, CX; SUB CX, CX instead of MOV CX, 00) having a more compact form.

2.5. Shift and rotation instructions:

General form	Effect	Affected conditioning indicators
SHL <s>, 1 SAL <s>, 1	Logic shift to left CF will store the most significant bit that was shifted. If <CF> <> the initial sign OF becomes 1.	CF,OF,SF,ZF,PF AF undefined
SHL <s>, CL SAL <s>, CL	Logic shift to left with a number of positions indicated by CL. CF will store the last shifted bit.	CF,OF,SF,ZF,PF AF undefined
SHR <s>, 1	Logic shift to right. Zeroes are inserted. CF will store the most significant bit. If the most significant bits of the result are different OF becomes 1.	CF,OF,SF,ZF,PF AF undefined
SHR <s>, CL	Logic shift to right with a number of positions indicated by CL. CF will store the last shifted bit.	CF,OF,SF,ZF,PF AF undefined

SAR <s>, 1	Arithmetic shift to right. Sign extension. The least significant bit will be stored by CF. If the most significant bits of the result are different OF becomes 1.	CF,OF,SF,ZF,PF AF undefined
SAR <s>, CL	Arithmetic shift to right with a number of positions indicated by CL. CF will store the last shifted bit.	CF,OF,SF,ZF,PF AF undefined
ROL <s>, 1	Rotate left by carry. If CF <> sign then OF becomes 1	CF, OF
ROL <s>, CL	Rotate left by carry with a number of positions indicated by CL.	CF, OF
ROR <s>,1	Rotate right by carry. If (CF) <> sign OF becomes 1.	CF, OF
ROR <s>, CL	Rotate right by carry with a number of positions indicated by CL.	CF, OF
RCL <s>, 1	Rotate left with carry. If (CF) <> sign OF becomes 1.	CF, OF
RCL <s>, CL	Rotate left with carry with a number of positions indicated by CL.	CF, OF
RCR <s>, 1	Rotate right with carry. If (CF) <> sign OF becomes 1.	CF, OF
RCR <s>, CL	Rotate right with carry with a number of positions indicated by CL.	CF, OF

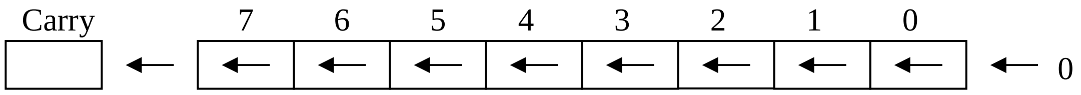
The format for all shift and rotate instructions is identical:

OPCODE {register | memory}, {CL | 1|nr}

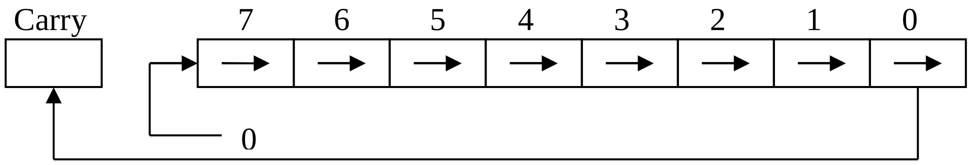
The result overwrites the source operand. The number of shift/rotate positions can be, number stored previously in register CL or nr for later processors.

The following figures show the result of these instructions on a byte operand for one position shifting/rotation.

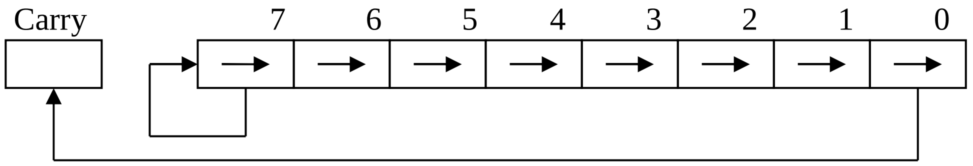
SHL, SAL



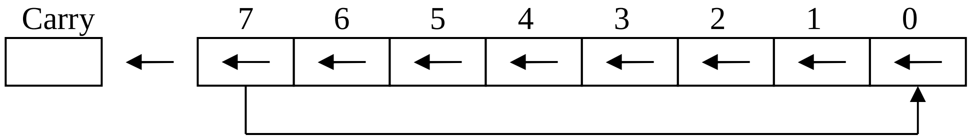
SHR



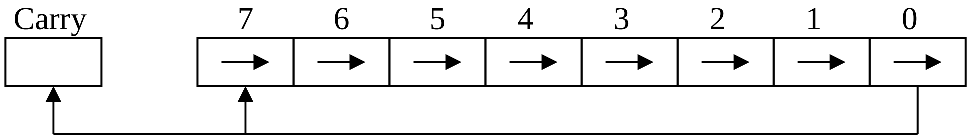
SAR



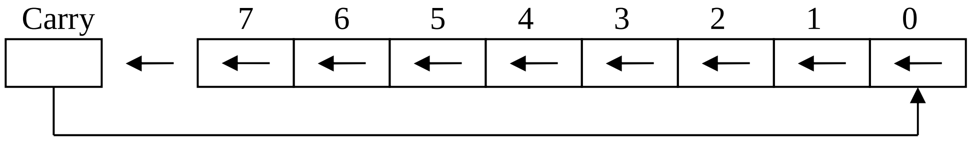
s ROL



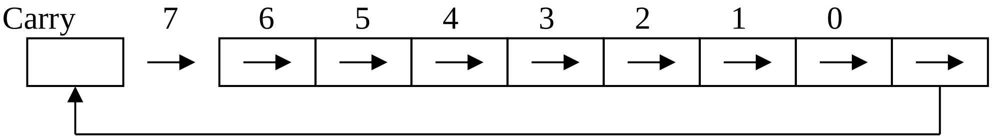
ROR



RCL



RCR



Example:

```
            ;a number stored in ax
            ;is multiplied by 10
SHL AX, 1               ;*2
MOV BX, AX              ;
SHL AX, 1               ;*4
SHL AX, 1               ;*8
ADD AX, BX              ;*10

            ;an unsigned number stored in ax
            ;is divided by 512
SHR AX, 1               ;/2
XCHG AH, AL             ;
XOR AH,AH                           ;/512

            ;a number stored in ax represented in C2 with sign
            ;is divided by 2
MOV AX, -16             ;
SAR AX, 1               ;/2

            ;a 32 bits unsigned number
            ;is divided by 2
```

```
DATA        SEGMENT
            MEM32 DD 500000
DATA        ENDS
CODE SEGMENT
… …
        SHR WORD PTR MEM32[2], 1        ;shifting in CF
        RCR WORD PTR MEM32[0], 1        ;rotation with CF
… …
CODE ENDS
```

3. Lab tasks

1. Study the examples.
2. Trace the examples with Turbo Debugger.
3. Write a program that generates an integer in byte representation and stores it to a REZ location after the formula:

   REZ = AL*NUM1+(NUM2*AL+BL)

   All parameters are byte size.
4. Implement the following operations using arithmetic and shift instructions:

   AX = 7*AX–2*BX–BX/8

   Parameters are byte size.
5. (complementary) Design an algorithm to multiply two 4 bytes numbers in C2 representation.