Experiment N^o 6

Shift Rotate and Jump Instructions

Introduction:

In this experiment you will be introduced to the shift and rotate instructions. You will also practice how to control the flow of an assembly language program using the compare instruction, the different jump instructions and the loop instructions.

Objectives:

1- Shift Instructions.
2- Rotate Instructions.
3- Compare Instruction.
4- Jump Instructions.
5- Loop Instructions.

References:

- Textbook: Sections 3.4, 3.5 and 4.1,
- Lecture notes.

The Shift Operations:

The shift operations are used to multiply or divide a number by another number that is a power of 2 (i.e. 2^n or 2^{-n}). Multiplication by 2 is achieved by a one-bit left shift, while division by 2 is achieved by a one-bit right shift. The Shift Arithmetic Right (SAR) instruction, is used to manipulate signed numbers. The regular Right Shift (SHR) of a signed number affects the sign bit, which could cause numbers to change their sign. The SAR preserves the sign bit by filling the vacated bits with the sign of the number. Shift Arithmetic Left (SAL) is identical in operation to SAR.

The rotate operations are very similar to the shift operations, but the bits are shifted out from one end of a number and fed back into the other end to fill the vacated bits. They are provided to facilitate the shift of long numbers (i.e. numbers of more than 16 bits). They are also used to reposition a certain bit of a number into a desired bit-location. The rotate right or left instructions through the carry flag (RCL and RCR) are similar to the regular rotate instructions (ROL and ROR), but the carry flag is considered as a part of the number. Hence, before the rotate operation, the carry flag bit is appended to the number as the least significant bit in the case of RCL, or as the most significant bit in the case of RCR.

Type	Instruction	Example	Meaning	F. Flags					
				OF	SF	ZF	AF	PF	CF
Shift	SHL	SHL AX,1	Shift AX left by 1 bit. Fill vacated bit with 0.	*	*	*	?	*	*
	SAL	SAL AX,1	Shift AX left by 1 bit. Fill vacated bit with 0.	*	*	*	?	*	*
	SHR	SHR NUM2,CL	Shift NUM2 right by the number of bits indicated in CL. Fill vacated bits with 0.	*	0	*	?	*	*
	SAR	SAR NUM2,CL	As SHR but fill vacated bits with the sign bit.	*	*	*	?	*	*
Rotate	ROL	ROL BH,CL	Same as SHL, but shifted bits are fed back to fill vacated bits.	*	-	-	-	-	*
	RCL	RCL BH,CL	Same as ROL, but carry flag is appended as MSB, and its content is shifted with the number.	*	-	-	-	-	*
	ROR	ROR NUM1,1	Same as SHR, but shifted bits are fed back to fill vacated bits.	*	-	-	-	-	*
	RCR	RCR NUM1,1	Same as ROR, but carry flag is appended as LSB, and its content is shifted with the number.	*	-	-	-	-	

Table 6. 1: Summary of the Shift and Rotate Instructions of the 8086 Microprocessor

Compare instruction:

The compare instruction is used to compare two numbers. At most one of these numbers may reside in memory. The compare instruction subtracts its source operand from its destination operand and sets the value of the status flags according to the subtraction result. The result of the subtraction is not stored anywhere. The flags are set as indicated in Table 6. 2.

Instruction	Example	Meaning
CMP	CMP AX, BX	If (AX = BX) then ZF ← 1 and CF ← 0
		If (AX < BX) then ZF ← 0 and CF ← 1
		If (AX > BX) then ZF ← 0 and CF ← 0

Table 6. 2: The Compare Instruction of the 8086 Microprocessor

Jump Instructions:

The jump instructions are used to transfer the flow of the process to the indicated operator. When the jump address is within the same segment, the jump is called *intra-segment jump*. When this address is outside the current segment, the jump is called *inter-segment jump.* An overview of all the jump instructions is given in Table 6. 3. Table 6. 4 lists the possible addressing modes used with the jump instructions. Whereas,
Table 6. 5 gives examples on the use of such instructions.

Type	Instruction		Meaning (jump if)	Condition
Unconditional	JMP		unconditional	None
Comparisons	JA	jnbe	above (not below or equal)	CF = 0 and ZF = 0
	JAE	jnb	above or equal (not below)	CF = 0
	JB	jnae	below (not above or equal)	CF = 1
	JBE	jna	below or equal (not above)	CF = 1 or ZF = 1
	JE	jz	equal (zero)	ZF = 1
	JNE	jnz	not equal (not zero)	ZF = 0
	JG	jnle	greater (not lower or equal)	ZF = 0 and SF = OF
	JGE	jnl	greater or equal (not lower)	SF = OF
	JL	jnge	lower (not greater or equal)	(SF xor OF) = 1 i.e. SF ≠ OF
	JLE	jng	lower or equal (not greater)	(SF xor OF or ZF) = 1
	JCXZ	loop	CX register is zero	(CF or ZF) = 0
Carry	JC		Carry	CF = 1
	JNC		no carry	CF = 0
Overflow	JNO		no overflow	OF = 0
	JO		overflow	OF =1
Parity Test	JNP	jpo	no parity (parity odd)	PF = 0
	JP	jpe	parity (parity even)	PF = 1
Sign Bit	JNS		no sign	SF = 0
	JS		sign	SF = 1
Zero Flag	JZ		zero	ZF = 1
	JNZ		non-zero	ZF = 0

Table 6. 3: Jump Instructions of the 8086 Microprocessor

Label Pointer	Range	Addressing Mode	Specified By	Encoded As	Directive
Short	+127/-128 bytes IP ← IP + Offset	Immediate	Word	Differentially*	SHORT
Near	Intra-segment IP ← Address	Immediate	Word	Differentially	NEAR PTR
		Register	Word	Absolute address	
		Memory	Word	Absolute address	
Far	Inter-segment IP ← Address CS ← Segment	Immediate	Double Word	Absolute address	FAR PTR
		Memory	Double Word	Absolute address	

*Differentially = Difference between current and next address.

Table 6. 4: Jump Instructions and Addressing Modes

Instruction	Example	Meaning
JMP	JMP FAR PTR [BX]	IP ← [BX], CS ←[BX+2]
JNZ	JNZ END	If (ZF=0) Then IP ← Offset of END
JE	JE FIRST	If (ZF=1) Then IP ← Offset of FIRST
JC	JC SECOND	If (CF=1) Then IP ← Offset of SECOND

Table 6. 5: Examples of Jump Instructions of the 8086 Microprocessor

The LOOP Instructions:

The LOOP instruction is a combination of a DEC and JNZ instructions. It causes execution to branch to the address associated with the LOOP instruction. The branching occurs a number of times equal to the number stored in the CX register. All LOOP instructions are summarized in Table 6. 6.

Instruction	Example	Meaning
LOOP	LOOP Label1	If (CX≠0) then IP ← Offset Label1
LOOPE **LOOPZ**	LOOPE Label1	If (CX≠0 and ZF = 0) then IP ← Offset Label1
LOOPNE **LOOPNZ**	LOOPNZ Label1	If (CX≠0 and ZF = 0) then IP ← Offset Label1

Table 6. 6: Summary of the LOOP Instructions.

The Loop Program Structure, Repeat-Until and While-Do:

Like the condionnal and unconditionnal jump instructions which can be used to simulate the IF-Then-Else structure of any programming language, the Loop instructions can be used to simulate the Repeat-Until and While-Do loops. These are used as shown in the following (Table 6. 7).

Structure	Repeat-Until	While-Do
Code	; Repeat until CX = 0 - MOV CX, COUNT Again: - - - - - - LOOP Again - -	; While (CX ≠ 0) Do - MOV CX, COUNT Again: JZ Next - - - - - LOOP Again Next: - -

Table 6. 7: The Loop Program Structure.

Pre Lab Work:

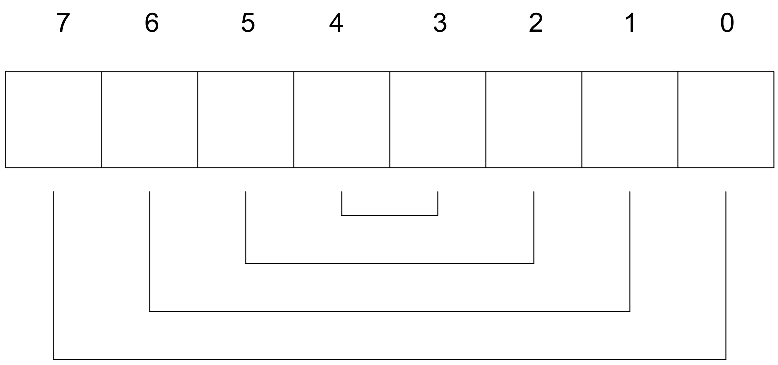
1. Complete program 6.1, according to the given guidelines.
2. Check on some values and see if it is working properly.
3. Comment on the program, trying to understand how conversion is done.
4. Write program 6.2 and make sure it contains no errors.
5. Do the modifications given in the guidelines. This will be program 6.3.
6. Bring your work to the lab.

Lab Work:

1- Show program 6.1 and your comments to your lab instructor.
2- Run program 6.2 using CodeView.
3- See what is the effect of such program on NUM1.
4- Run program 6.3, and see the effect on NUM1 after displaying NUM2.
5- Modify program 6.2, using program 6.1, such that you enter an 8-bit binary number from the keyboard, and invert swap the high and the low nibbles of the number, and finally display it. Call this program 6.4.
6- Show all your work to the instructor, and submit it at the end of the lab session.

Lab Assignment:

Write a program that prompts the user to enter an 8 bit binary number, between 0 and 255. The program then inverts all bits according to the figure below. This program is useful in Signal Processing for the calculation of the Fast Fourier Transform (FFT). The operation is called Decimation (in time or frequency), and the bit manipulation is called bit shuffling i.e. rearrangement of bits.



Bit (i) ← Bit (7- i) for i = 0 – 7

Figure 6. 1: Bit Shuffling

TITLE "Experiment 6, Program 1"

```
; This program adds 2 binary numbers and prints the result in binary format

.MODEL SMALL
.STACK 200
.DATA
        CRLF DB 0DH, 0AH, '$'
        PROMPT1 DB 'Enter the first 8-bit binary number:  ','$'
        PROMPT2 DB 'Enter the second 8-bit binary number: ','$'
        PROMPT3 DB 'The sum of the two numbers in binary is: ','$'
        NUM1 DW ?
        NUM2 DW ?

.CODE
.STARTUP
                ; DISPLAY PROMPT1
                ; READ AND CONVERT THE FIRST NUMBER

                CALL READ

                MOV NUM1,BX                 ; READ FROM STACK

                MOV DX, OFFSET CRLF
                MOV AH, 09H
                INT 21H
                ; DISPLAY PROMPT2

                ; READ AND CONVERT THE SECOND NUMBER
                CALL READ
                MOV NUM2,BX                 ; READ FROM STACK


                MOV BX, NUM1
                ADD BX, NUM2                ; ADD THE TWO NUMBERS
                ;DISPLAY PROMPT3
                CALL RESULT
.EXIT

;**************************************************
; PROC READ A NUMBER AND CONVERT IT TO BINARY
READ   PROC     NEAR
                MOV BX, 0000
                MOV CX, 0008
                MOV AH, 01H
L1:             INT 21H
                SUB AL, 30H
                SHL BL, 1
                OR BL, AL
                LOOP L1
                XOR BH, BH
                RET
READ ENDP
END
```

```
;*************************************************
; PROC. DISPLAY RESULT
RESULT PROC NEAR
            MOV CX, 0008            ; DISPLAYING
            CLC
NEXT:       RCL BL, 1
            JNC BIT_0
            MOV DL, '1'
            MOV AH, 02H
            INT 21H
            JMP LAST
BIT_0:      MOV DL, '0'
            MOV AH, 02H
            INT 21H
LAST:       LOOP NEXT
            RET
RESULT ENDP
```

```
;*************************************************
;*************************************************
 TITLE "PROGRAM 2 EXPERIMENT 6"
; This program shows how to manipulate a two-digit numbers

.MODEL SMALL
.STACK 200
.DATA
      NUM1 DB ?
      NUM2 DB ?

.CODE
.STARTUP

      ; READ NUMBER NUM1

      MOV AL, NUM1
      AND AL, 0FH
      MOV CL, 04H
      SHL AL,CL
      MOV BL, AL

      MOV AL, NUM1
      AND AL, 0F0H

      MOV CL, 04H
      SHR AL, CL

      OR BL, AL
      MOV NUM2, BL


      ; DISPLAY NUMBER NUM1

      ; CONVERT NUM2 TO BINARY
      ; DISPLAY NUMBER NUM2
.EXIT
END
```