LABORATORY WORK NO. 7
FLOW CONTROL INSTRUCTIONS

1. Object of laboratory

The x86 microprocessor family has a large variety of instructions that allow instruction flow control. We have 4 categories: jump, loop, calling and return instructions.

2. Theoretical considerations

2.1. Jump instructions.

Jumping is the most direct method of modifying the instruction flow. The jump instructions change the value of the IP register and sometimes of the CS register (for intersegment jump), so the IP and CS registers will be loaded with the address of the target.

2.1.1. The unconditional jump.

The JMP instruction is used for making an unconditional jump to a specified address. The jump in the same segment can be short/relative or near, the destination address that can be between -126..129 bytes relative to the jump instruction for short jump or in the same segment for near jump. A far jump is a jump to a different segment.

From the destination address specification point of view there are direct and indirect jumps. In the direct jumps, the destination address is specified through a label. The syntax is:

```
JMP    label
```

For the **short** jumps in the same segment, the addressing is IP **relative**.

After the instruction code there is a displacement on a byte that represent the distance from the current address to the destination.

Example:

```
ALFA:
        …
        JMP     ALFA
```

If the distance to the label is under 126 bytes and has been defined **before** the jump instruction, than a short jump type is encoded.

If the label is defined **after** the jump instruction than a near jump type is encoded, indifferent if the distance between the jump instruction and label is lesser or not than 129 bytes. We can force a short jump by using the SHORT operator.

Example:

```
        JMP     SHORT BETA
        …
BETA:
```

Observation: Using the SHORT operator in an improper situation will generate an assembly error.

For a near jump the target address is encoded in the instruction on 2 bytes.

In case of using the direct addressing for jumping between different segments the instruction code is followed by a displacement of four bytes that representing the destination address segment : offset.

If the destination label has been defined before, the encoding is correct. If the label is defined afterwards, it is necessary to specify the FAR type for this label.

Example:

```
        JMP     FAR PTR GAMA
        …
GAMA:
```

In case of indirect jumping, the destination address is specified through an operand, the syntax is:

```
        JMP {register| memory}
```

Example:

```
        JMP     AX
        JMP     [BX]
        JMP     ALFA            ; ALFA is a var. word or double word
```

If the variable is defined after the jump instruction in the case of far jumps we must use DWORD PTR operator.

Example:
```
        JMP     DWORD PTR ALFA
```

Example:
```
CODE SEGMENT
        JMP             PROCES

CTL_TBL     LABEL       WORD
        DW              EXTENDED            ; the key with extended
                                            ; code (2 car.)
        DW              CTRLA               ; the key CTRL/A
        DW              CTRLB               ; the key CTRL/B

PROCES:
        MOV             AH, 8H              ; reading the key in AL
        INT             21H
        CBW
        MOV             BX, AX
        SHL             BX, 1               ; the address calculation
in the table
        JMP             CTL_TBL [BX]
        …

EXTENDED:
        MOV             AH, 8H              ; takes the second cod
        INT             21H
        …

CTRLA:                                      ; routine for CTRL/A
        …
        JMP             NEXT
CTRLB:                                      ; routine for CTRL/B
        …
        JMP             NEXT
NEXT:                                       ; continue
        …
CODE ENDS
```

2.1.2. Conditional jumps

The conditional jump is the most frequent method of modifying the instruction flow. It consists of a process in two steps. In the first step the condition is tested and in the second the jump is done if the condition is true or the next instruction in executed if the condition is false

The jump instruction syntax is:

```
Jcc     label
```

The conditional jumps are short type, so that the distance at the destination address must be in the -126..129 range. Otherwise an error is signaled. The destination address is specified through a displacement on a signed byte relative to the current address. The conditional jumps use as a condition the flags or logical combination of the flags.

The flags can be set by any of the instructions that affect the flags. The most frequent WAY IS TO use the CMP or TEST instructions.

The conditional jump is made using one of the 13 conditional jump instructions.

If the target of the conditional jump is out of range, it must be replaced through a conditional jump of reverse condition followed by an unconditional jump.

Example:

```
        CMP   AX, 7
        JE    NEAR
        CMP   AX, 6            ; if AX is 6 and the jump is greater
than 129 bytes
                               ; the instruction of conditional jump is
replaced
        JNE   NEAR
        JMP   FAR

NEAR:                          ; less than 128 bytes
                               ; from the jump instruction
        …
FAR:                           ; more than 128 bytes
                               ; from the jump instruction
```

2.1.3. Compare and jump

The CMP instruction compares 2 operands by subtracting the source operand from the destination operand without affecting the destination and setting the flags. The syntax is:

CMP {register | memory}, {register | memory | immediate value}

The conditional jump instruction used after the compare instruction has the flow chart accordance with the tested relation, generated form the next letters:

LETTER	MEANING	
J	Jump	
G	Greater than	(for signed value)
L	Less than	(for signed value)
A	Above	(for unsigned values)
B	Below	(for unsigned values)
E	Equal	
N	Not	

In the next table there are represented the conditional jump instruction according to each relation:

Jump condition	Compare with sign	Jump condition	Compare without sign	Jump condition
Equal =	JE	ZF=1	JE	ZF=1
Not equal <>	JNE	ZF=0	JNE	ZF=0
Greater than >	JG or JNLE	ZF=0 and SF=OF	JA or JNBE	ZF=0 and CF=0
Less than <	JL or JNGE	SF<>OF	JB or JNAE	CF=1
Greater than or equal >=	JGE or JNL	SF=OF	JAE or JNB	CF=0
Less than or equal <=	JLE or JNG	ZF=1 or SF=OF	JBE or JNA	CF=1 or ZF=1

Example

```
; IF (CX< -20) THEN DX=30 ELSE DX=20
        CMP   CX, -20
        JL    LESS
        MOV   DX, 20
        JMP   CONT
LESS:
        MOV   DX, 30
CONT:
```

Example:

```
; IF (CX>= -20) THEN DX=30 ELSE DX=20
        CMP   CX, -20
        JNL   NOTLESS
        MOV   DX, 20
        JMP   CONT
NOTLESS:
        MOV   DX, 30
CONT:
```

Jumps based on flag value are:

INSTRUCTIONS	JUMP CONDITIONS
JO	OF=1
JNO	OF=0
JC	CF=1
JNC	CF=0
JZ	ZF=0
JNZ	ZF=1
JS	SF=1
JNS	SF=0
JP	PF=1
JNP	PF=0
JPE	PF=1
JPO	PF=0
JCXZ	CX=0

As it can be observed JCXZ is the only conditional jump instruction that does not test the flags but the content of the CX register.

Example:

```
        ADD   AX, BX
        JO    OVERFLOW
        …
OVERFLOW:
```

2.2. Loop instructions

The cycling instructions allow an easy programming of the control structures of the final test cycle type.

The syntax of these instructions is:

```
LOOP       label      ; CX is decremented and if CX is not
                      ; zero the loop is done.
LOOPE      label      ; CX is decremented and if CX is not
                      ; zero and ZF=1 the loop is done.
LOOPZ      label      ; identical with LOOPE
LOOPNE     label      ; CX is decremented and if CX is not
                      ; zero and ZF=0 the loop is done.
LOOPNZ     label      ; identical with LOOPNE
```

The loop instructions decrement the content of the CX register and if the jump condition is fulfilled the loop is done.

The distance between the looping instruction and the destination address must be between the range -126..129 bytes

Example:

```
        MOV   CX, 200            ; initialize counter
NEXT:
        …
        LOOP  NEXT               ; repeat if CX in not null
                                 ; continue after the cycle
```

This loop has the same effect as the one in the next example:

Example:

```
        MOV   CX, 200
NEXT:
        …
        DEC   CX
        CMP   CX, 0
        JNE   NEXT
```

The first version is more efficient.

Using the JCXZ instruction allows us to avoid executing a loop for CX=0.

Example:

```
NEXT:
        JCXZ  CONT
        …
        LOOP NEXT
CONT:
```

2.3. Using procedures

The procedures are code units that fulfill specific functions. They represent a way of dividing the code in functional parts or blocks so that a specific function can be executed from any other point in the program without having to insert the same code again and again.

The procedures from the assembler language are comparable with the C functions..

For defining and using procedures there are two pseudo-instructions and two instructions. The PROC and ENDP directives mark the beginning and the end of the procedure. The CALL instruction is used to call the defined procedures, and the RET instruction is used for returning to the calling point.

The CALL and RET instructions use the stack to store and restore the return address. The CALL instruction pushes on stack the return address (the address after the CALL instruction) and then a jump to the address at the beginning of the procedure is done.

The RET instruction extracts from the stack the address introduced by the CALL instruction and returns to the instruction after the call.

The procedures can be found or not in the same segment with the calling instructions.

From this point of view there are NEAR and FAR type procedures. When declaring the procedures their type is declared too. The NEAR type is implicit.

The procedure definition syntax is:

```
Label       PROC  [NEAR | FAR]
            …
            RET   [constant]
Label       ENDP
```

The RET instruction allows one constant operand that specify a number of bytes that will be added to the content of the SP register after returning from the procedure. This operand can be used for deleting from the stack the arguments that were transmitted to the procedure through the stack.

The call procedure syntax is:

```
CALL   {register | memory}
```

3. Lab tasks

1. Study the instructions and the examples presented before.
2. Write a program sequence that transforms the ASCII code of a small letter in the ASCII code of the capital letter. The code will be taken from a memory location and saved in the same memory location.
3. Write a program that calculates the average of the numbers from an array of unsigned values. Write the average obtained on the display and the message "The average is: ". The average will be calculated as a integer number. Use DOS system function calls to print messages.
4. Compute the average only for the number between [5..10]
5. Write a program that displays the content of AX register in decimal. HINT: divide AX several times with 10, print the results in reverse order
6. Write a program that reads an integer without sign from the keyboard until the enter key is pressed. HINT: every digit you read will be converted to its numeric value. Compute like this: 145=(((1*10)+4)*10)+5
7. Write a procedure that converts a hex digit (0 to F) in an ASCII character. Send the hex digit to the procedure in the AL register, and the procedure returns the ASCII character in the same register.