Lecture 9

Program Logic and Control

Text:
(5th Edition: Chapter 7)
(4th Edition Chapter 8)

No programming language is complete without the ability to make decisions and react by

- skipping over instructions which should not be executed
- repeating instructions under some control

In high-level languages there are statements which do the skipping and repetition of instructions:

if (*condition*) then *statement* [else *statement*];

```
if (count == 0) return 0
else  return sum / count;
```

while (*condition*) *statement*;

```
while (X<10) {
   sum = sum * 2;
   x ++
}
```

for (*condition; condition; condition*) *statement*;

```
for (i=0; i<10; i++)
   j = j+i;
```

do *statement* while (*condition*);

```
do
   x = x+y
while (x < 100);
```

There are no built-in machine instructions which implement these complex structures.

These statement "structures" must be written explicitly in machine language (and therefore, assembly language)

The instruction which can change the "flow" of the program is the jump instruction:

```
JMP     GoHere
```

The identifier "GoHere" must identify the assembly language statement at which execution should continue. "GoHere" is called a LABEL.

LABELS

A label is used to indicate the destination of a JUMP

```
          JMP  OVERTIME
          …..
OVERTIME:
```

```
AGAIN:   ……
         ……
         JMP   AGAIN
```

The label

- begins in column 1
- is a legal identifier
- is terminated with a colon (“:”) when defined
- has no colon when used
- may be defined on the same or preceding line

Conditional Jump Instructions

The conditional jump instructions examine the 16-bit flags register. Each bit represents the presence or absence of some condition, such as

SF Sign Flag This bit is changed after an arithmetic instruction. If the result of arithmetic is positive, the bit will be 0, if the result is negative it will be 1.

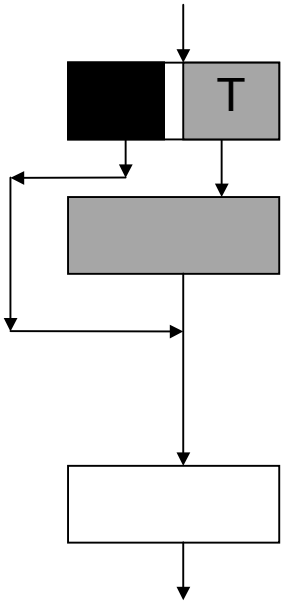
ZF Zero Flag This bit is changed after an arithmetic or comparison instruction. If the result is not zero, the bit is cleared to zero, if the result is zero the bit is set to one.

Some JUMP Instructions (signed data)

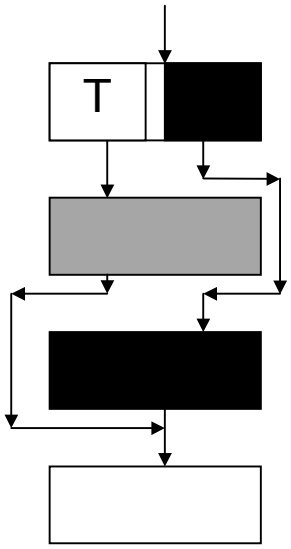
	Description		Description	FlagsTested
JE	Jump if equal	JZ	Jump if zero	ZF
JNE	Jump Not Equal	JNZ	Jump Not Zero	ZF
JG	Jump Greater Than	JNLE	Jump Not Less or Equal	ZF, SF, OF
JGE	Jump Greater Than or Equal	JNL	Jump Not Less	SF, OF
JL	Jump Less Than	JNGE	Jump Not Greater Than or Equal	SF, OF
JLE	Jump Less Than or Equal	JNG	Jump Not Greater Than	ZF, SF, OF

Decision Making

`if - then`



`if - then - else`



Example:

Write assembly code for

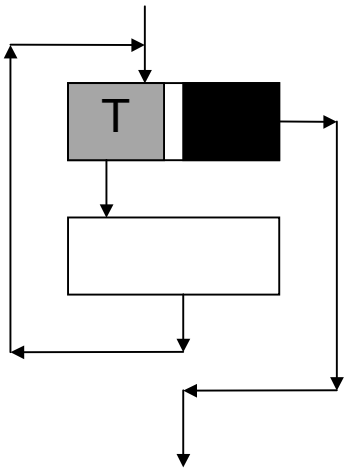
```
if (X = Y) P++ else Q--;
```

Determine the OPPOSITE condition of the comparison (the JUMP is done when it is **FALSE**). (Here, not equal)

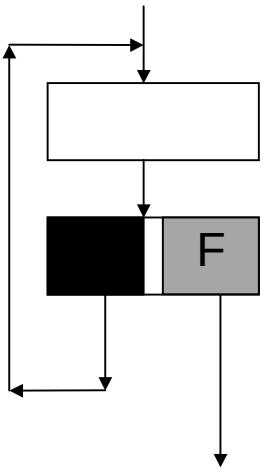
```
if:
      mov     ax,X     ; need one op in reg
      cmp     ax,Y     ; if X is
      jne     else     ;   equal to Y
      inc     P        ;   do this
      jmp     endif    ;   and get out
else:                  ; else
      dec     Q        ;   do this
endif:                 ; and we're done
```

Statement repetition :

while



do - while



Example:

```
while (X > Y) {
    X--;
    Y++
}
```

```
while:
        mov    ax,X     ; copy of X needed
        cmp    ax,Y     ; while X is
        jng    endwhile  ; greater than Y
        dec    X        ;      x--
        inc    Y        ;      y++
        jmp    while    ; check again
endwhile:
```

Try:

```
do
    X++;
while (X<100);
```

Nesting Control Structures

Suppose you are given the following (silly) Java program and you wish to write it in assembly language:

```
X = 0;
while (X < 10) {
    if (X = 4) then A = 1
               else B = 2;
    X = X + 1
}  // while
```

The program contains a while loop and an if-then-else statement.

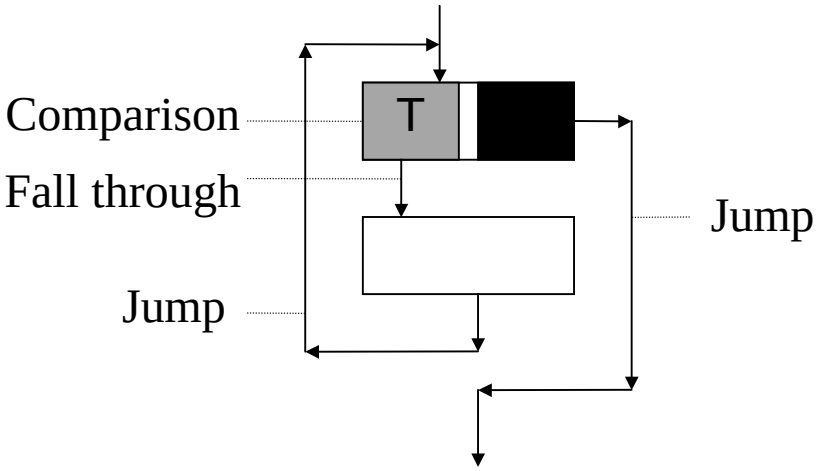
Two values can be compared using the CMP instruction. Suppose that the value of X is in the AX register. The comparison for the while loop could be done with:

```
        CMP    AX,10
```

But what should the next instruction do after the comparison?

Look at the structure of a while loop!

While loop:



The while loop structure evaluates the condition, and if the condition is FALSE, a jump is done to exit the loop. If the condition is true, then fall through.

After the loop body is done, another jump follows which is always done (there are no conditions, so it is called an **unconditional** jump).

The two jumps require two different labels (one to go back, one to exit).

```
WHILE:
    CMP    AX,10      ;compare X to 10
    jump to EXIT if not less than;
    do the body of the loop
    JMP    WHILE
EXIT:                 ;end of the loop
```

```
WHILE:
    CMP    AX,10      ;compare X to 10
    JNL    EXIT
    do the body of the loop
    JMP    WHILE
EXIT:                 ;end of the loop
```

The while loop is now done. Now we need to code the if-then-else statement and the assignment statement.

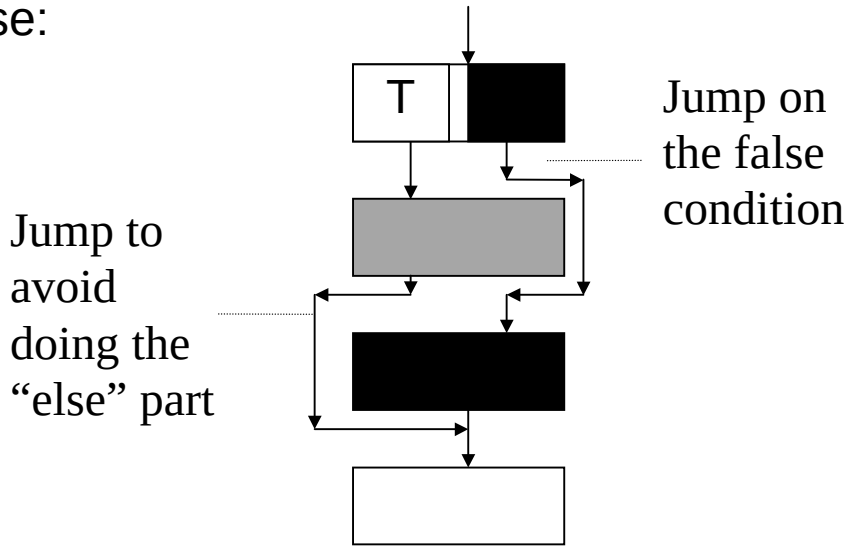
```
X = 0;
while (X < 10) {
    if (X == 4)  A = 1;
       else B = 2;
    X = X + 1
} //while
```

They should be placed, in sequence, inside the while loop (where the body of the while loop goes).

```
WHILE:
    CMP    AX,10      ;compare X to 10
    JNL    EXIT
    do the body of the loop
    JMP    WHILE
EXIT:                 ;end of the loop
```

The technique for coding if-then and if-then-else is similar to that for the while loop – look at the structure and determine where the conditional jumps go, and under what conditions they jump.

If-then-else:




```
X = 0;
while (X < 10) {
    if (X == 4)  A = 1;
      else B = 2;
    X = X + 1
} //while
```

```
WHILE:
    CMP    AX,10       ;while x is
    JNL    EXIT        ;  less than 10
    CMP    AX,4        ;if (X = 4)
    JNE    ELSE        ;   then
    MOV    A,1         ;      A = 1
    JMP    ENDIF
ELSE:                  ;   else
    MOV    B,2         ;      B = 2
ENDIF:
    INC    AX          ;X = X + 1
    JMP    WHILE
EXIT:                  ;end of the loop
```

The LOOP instruction

- works in conjunction with the CX register
- must jump to a short address
- does several operations

1. Subtract 1 (one) from the CX register

2. If the CX register is non-zero, JUMP to the short address, otherwise fall through to the next instruction.

```
        page 60,132
TITLE   P08LOOP (COM)  Illustration of LOOP
        .MODEL SMALL
        .CODE
        ORG    100H
BEGIN   PROC   NEAR
        MOV    AX,01     ;Initialize AX,
        MOV    BX,01     ;  BX, and
        MOV    DX,01     ;  DX to 01
        MOV    CX,10     ;Initialize
A20:                     ;  number of loops
        INC    AX        ;Add 01 to AX
        ADD    BX,AX     ;Add AX to BX
        SHL    DX,1      ;Double DX
        LOOP   A20       ;Decrement CX,
                         ;  loop if nonzero
        MOV    AX,4C00H  ;Exit to DOS
        INT    21H
BEGIN   ENDP
        END    BEGIN
```

Exercises - Lecture 9

Write assembly code for the following statements. Unless told, presume all variables are integer.

1.
```
if (a <= b)
    z = a+b;
```

2.
```
if (p==4) {
    p = 0;
    r = 1  }
else {
    p = 1;
    r = 0;
} //else
```

3.
```
if ((p==1) and (q==2)) {
    q = 1;
    p = 2;
} //if
```

4.
```
if ((a==b) or (b==c) or (c==d)) {
    a=0;
    b=0;
    c= d+1;
} //if
```

5.
```
while ( x != y) x++;
```

6.
```
n=1;
sumodd = 0;
while (n < last) {
    sumodd += n;
    n += 2;
}
```

7.
```
n=1;
sumodd = 0;
while (n < last) {
    if ((n != 9) or (n != 99))
        sumodd += n;
    n += 2;
}
```

8.
```
do {
    p++;
    q++;
while (p < 100)
```

9.
```
do {
    p++;
    q--;
while (p != q)
```

10. Encode the following statements using the LOOP instruction.

```
x=0;
for (i=1; i<=10; i++) x = x+i;
```