CSE 331/L-15/30.03.2024/

From Page- 117 (Yu - Manuth)

For Loop:

LOOP ⇒ Background Execution ⇒ DEC cx
cmp cx, 0
JNE destination_label

↳ Format:

LOOP destination_label

6.8/ Write a count-controlled loop to display a row of 80 stars.

Algorithm:

FOR 80 times DO
    display '*'
END_FOR

Code:

```
MOV CX, 80 ; count register, how many time loop will execute.
MOV AH, 2  ; prepare for output
MOV DL, '*' ; loading character to output register

TOP:
    INT 21h  ; display it
LOOP TOP ;  run the three instruction describe above and
            loop until cx become 0.
```

⇒ Caution!

According to background execution, you can see, content of cx will decrement first then it will compare with 0. And then it will jump if cx is not equal to 0.

careful, not equal
doesn't check above or bellow.

⇒ And also the statement will execute first then loop will execute. So, if by any chance cx became 0 before enter the statement body. loop will decrement the cx first.

cx = 0000 0000 ⇒ 0000 H ⇒ 0

DEC cx = 1111 1111 ⇒ FFFF H ⇒ 65535

And then it will compare with 0, which result in False. So the loop will execute 65536 times.

⇒ To prevent this:

JCXZ ⇒ Jump if cx is zero

```
CMP CX,0
JE  destination_label
```

Code:

```
MOV CX, 80    ; set counter register
MOV AH, 2     ; prepare for output
MOV DL, '*'   ; storing character to output register
JCXZ SKIP     ; checking if cx is zero or not
```

```
TOP:
    INT 21h     ; display it
LOOP TOP        ; loop until cx became 0.

SKIP:
    ; normal execution flow
```

While Loop:

- first check the condition, then run the statement and repeat.

6.9/ Write some code to count the number of characters in an input line.

Algorithm:

```
initialize count to 0
read a character
WHILE character <> carriage return DO
        count = count + 1
        read a character
END. WHILE
```

Code:

```
MOV DX,0    ; used for counting
MOV AH, 1   ; prepare to read
INT 21h     ; first character read

WHILE_:
    CMP AL, 13  ; comparing with carriage return
    JE END_WHILE ; if yes, break the loop
    INC DX      ; if not count the character
    INT 21h     ; read another character to check before
                  next loop run.
    JMP WHILE_  ; loop back

END_WHILE:
        ; normal execution flow
```

⇒ any variable involved in the condition, must be initialize before the loop is entered.

REPEAT LOOP:

- Like Do-WHILE loop
- execute statement first, then check the condition

6.10/ Write some code to read characters until a blank is read.

Algorithm:

REPEAT

read a character

UNTIL character is a blank

Code:

```
MOV AH, 1  : prepare for input

REPEAT:
        INT 21H  ; read char in AL
        CMP AL, ' ' ; comparing with space
        JNE REPEAT  ; no, repeat read
```

normal label, not reserve word

WHILE vs REPEAT (Do-While)

⇒ WHILE loop can be bypassed if the terminating condition is initially false.

But in REPEAT, statement will run at least once before condition check.

⇒ REPEAT loop has only one jump and very short in code.

But in WHILE loop, it has two jump, one for break and another for repeating the loop, so little bit complex and larger code.

6-5 ⇒ Sample Question for Practice ⇒ H.W

Chapter-8

Stack and Introduction to Procedures.

Page ⇒ 151 - 157 ⇒ Example & already covered in previous classes.

8.3 ⇒ Page-158

Call Return Procedure

↳ use PUSH, POP to keep track the content of IP.

Call ⇒ PUSH IP
RET ⇒ POP IP

} Stack segment used

⇒ also the used registers will push to the stack during call and pop on return.

Formate/Structure:

```
Name  PROC  Type
    body
RET
Name  ENDP
```

Type → NEAR ⇒ within the code segment
Type → FAR ⇒ Out of the code segment
↳ more details on chapter-4 Read as your wish.
Type → if not declared, complier use NEAR by default.

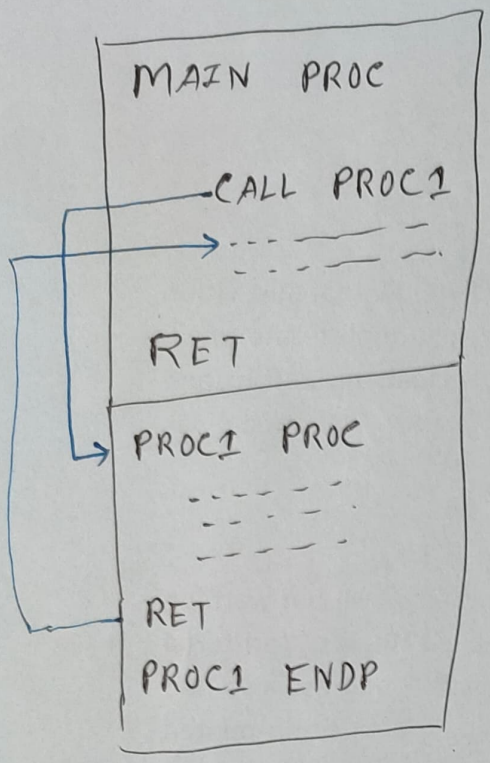

Fig. 8·4A, B
8·5A, B

Page- 162 - 169
- Example

Chapter - 13

Macros

Page- 267

⇒ Macro is quite same a procedure, but it don't do any PUSH, POP or don't have return option.
What it does is, paste its own code on the position where the macro was called.

⇒ Syntax:

```
Name  MACRO  d1, d2, d3, .... dn
      ....
      statement
      ....
ENDM
```

You can also relate it with function used in high level language. Passing parameter, but it manipulate the parameter but don't return any value. It's the difference.

13.1/ Define a macro to move a word into a word.

⇒ name of macro — in higher language we used (int a, int b)

```
MOVW   MACRO   WORD1, WORD2
       PUSH    WORD2
       POP     WORD1
       ENDM
```

⇒

```
MAIN PROC                    MAIN PROC
- - - -                     - - - -
- - -                       - - -
- - -                       - - -
MOVW A, DX   --after compile-->  PUSH DX
- - - -                          POP  A
- - -                       - - - -
- - -                       - - -
RET                          RET
```

Page - 268 - 286
- Example

(X) Difference between MACRO and PROC.

	MACRO	PROC
Assembly Time	✗ more	✓ less
Execution Time	✓ less	✗ more
Program Size	✗ more	✓ less

Describe more in sentence. Don't use table. Also add the structure difference describe in previous page. And add sample code on syntax.

When to use:

⇒ suitable for small and frequently occuring tasks.

Because, PROC alway do some PUSH and POP which take more time if you call PROC too frequently.

⇒ Consider Memory Size:

if you have less memory, go for PROC no other way.

if you have enough memory, think about execution time.

if frequently occurs then use it

otherwise go for PROC

Page- 286 - Done

Next class ⇒ Chapter-15

Midterm - 25.04.2024

Syllabus up to chapter 15 Next class