Microprocessor & Assembly Language

- MACRO

Macros - Overview

- A macro lets the assembler substitute a block of code for an identifier that you have created
- Macros are expanded during the assembler's first pass through the file, called the *preprocessor step*
- Each time a macro is *invoked*, a copy of the macro is inserted in the program.
- Macros tend to generate more executable code than procedure calls
- Macros can receive parameters, making them more flexible

Syntax

MACRO Definition:

```
Macro_name MACRO d1, d2, … dn
statements
ENDM
```

To invoke a macro, the syntax is:

```
Macro_name d1, d2, … dn
```

- A MACRO definition must come before its invocation. So, place MACRO definitions at the beginning of a program.
- Arguments are optional but can be only register or memory variable.
- On each invocation, each dummy argument is replaced by actual arguments.

A Macro to move a word into a word.

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

To invoke the macro MOVW to move B to A, where A and B are two word variables:

```
MOVW A,B
```

mPutchar Macro

This macro displays a "*" character on the console:

```
mPutchar MACRO
    push  ax
    push  dx
    mov   ah,2
    mov   dl,'*'
    int   21h
    pop   dx
    pop   ax
ENDM
```

mPutchar Macros with Parameters

Adding a parameter to the mPutchar macro makes it much more flexible (and therefore useful):

```
mPutchar MACRO char
    push  ax
    push  dx
    mov   ah,2
    mov   dl,char
    int   21h
    pop   dx
    pop   ax
ENDM
```

mDisplayStr Macro

This macro writes a string to standard output:

```
mDisplayStr MACRO string
    push ax
    push dx
    mov  ah,9
    mov  dx,offset string
    int  21h
    pop  dx
    pop  ax
ENDM
```

Restoring Registers

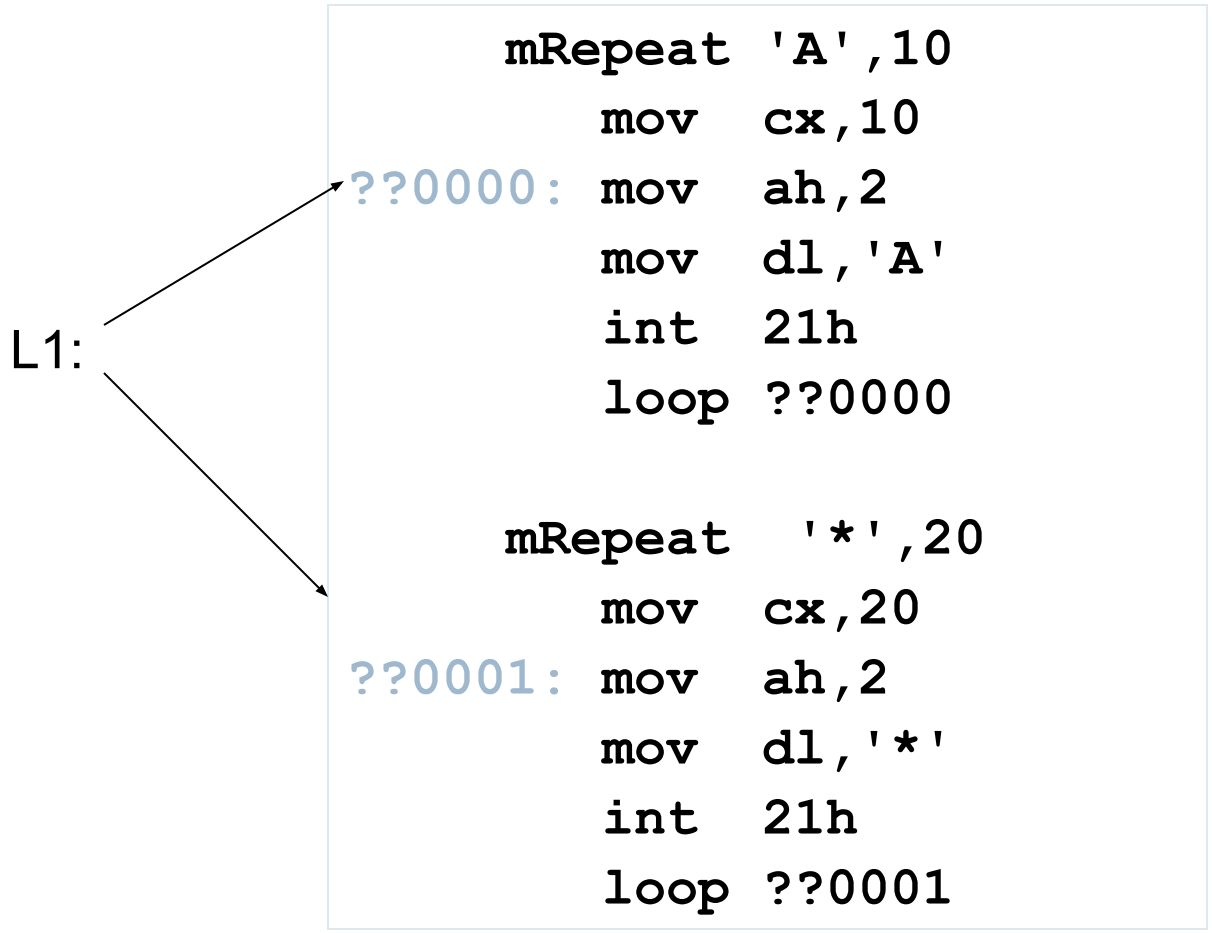
```
EXCH MACRO WORD1, WORD2
     PUSH AX
     MOV AX, WORD1
     XCHG AX, WORD2
     MOV WORD1, AX
     POP  AX
ENDM
```

LOCAL Directive

The LOCAL directive tells the assembler to create a unique label name each time the macro is expanded.

```
mRepeat MACRO char, count
     LOCAL L1
     mov  cx,count
L1:  mov  ah,2
     mov  dl,char
     int  21h
     loop L1
ENDM
```

Expanding the mRepeat Macro



Place Largest of two words in AX

```
GET_BIG MACRO WORD1, WORD2
    LOCAL EXIT
    MOV AX, WORD1
    CMP AX, WORD2
    JG EXIT
    MOV AX, WORD2
EXIT:
   ENDM
```

A macro can invoke another macro

```
SAVE_REGS MACRO R1, R2, R3
       PUSH R1
    PUSH R2
    PUSH R3
ENDM
```

```
RESTRE_REGS MACRO S1, S2, S3
       POP S1
    POP S2
    POP S3
ENDM
```

A macro to copy a string

```
COPY MACRO SOURCE, DESTINATION, LENGTH
    SAVE_REGS CX, SI, DI
    LEA SI, SOURCE
    LEA DI, DESTINATION
    CLD
    MOV CX, LENGTH
    REP MOVSB
    RESTORE_REGS DI, SI, CX
ENDM
```

Examples

- Write a macro to return to DOS.
- Write a macro to execute a carriage return and line feed
- Write a macro to print hex digit.

MACRO vs. Procedures

Procedure	**Macro**
1. Called at execution time	1. Invoked at assembly time
2. Whenever a procedure is called in a program, the control transfers to the procedure and returns after executing the procedure.	2. Assembler copies macro statements into the program at the position of invocation. No transfer of controls occurs during execution.
3. Suitable whenever there is a need to perform a big task.	3. Suitable for small and frequently occurring statements.

MACRO vs. Procedures Contd..

- A program containing MACROS usually takes longer to assemble than a similar program containing procedures, because it takes time to expand a macro.
- The code generated by MACRO expansion generally executes faster than a procedure call, because latter involves saving return addresses, transferring control, returning from procedure.

Ch#13 – Exercise Questions

- Solve Q#1, 2 and 3.