

Name(s): _____

Last week you developed a method of writing a register contents to the screen. This was more challenging than it first appeared. Your code for converting the contents of the AL register to ASCII could have looked something like this...

```
.DATA HEXTABLE DB '0123456789ABCDEF'
      HINIBBLE  DB ?
      LONIBBLE  DB ?
.CODE
...
AL2ASCII PROC NEAR
      PUSHALL
      PUSH AX
      MOV CL,4h
      SHR AL,CL
      AND AL,0Fh
      LEA BX,ASCIITABLE
      XLAT
      MOV HINIBBLE,AL
      POP AX
      AND AL,0Fh
      XLAT
      MOV LONIBBLE,AL
      POPALL
      RET
AL2ASCII ENDP
```

And in your main program, you may have used it like...

```
.DATA COL DB 35h
      ROW DB 12h
      HINIBBLE DB ?
      LONIBBLE DB ?
      CHAR DB ?
...
.CODE
...
CALL AL2ASCII ;...Get data into AL...
CALL CLS ;Convert AL to ASCII
CALL POSCURS ;Clear the screen
;Position cursor based on
;...contents or COL and ROW

MOV AL,HINIBBLE
CALL WRITECHAR ;Write hi nibble of AL
INC COL ;Advance the cursor
CALL POSCURS
MOV AL,LONIBBLE ;Write lo nibble of AL
CALL WRITECHAR
...
```

Use this code approach, or your own, to complete the labs in the handout for lab #7

Do lab 7.1, but modify it to write to the screen, in eight consecutive locations, the contents of AL. Employ STOSB as directed in the text, but use a variable array - write AL to the array and then, in a separate routine, write the contents of the array to screen. To do this second part, you can employ LODSB and XLAT, along with your version of AL2ASCII and a *writchar* routine (along with screen maintenance stuff, like clearing screen and positioning the cursor, etc.) with an ASCII array defined as...

```
.DATA  
HEXTABLE DB '0123456789ABCDEF'
```

We'll employ the full ASCII code set soon.

DEMONSTRATE THIS PROGRAM _____ *sign-off*

Do lab 7.2, which is basically the same as 7.1, but place the length of the array to be written (and shown on the screen) in another variable (named LENGTH, for example).

DEMONSTRATE THIS PROGRAM _____ *sign-off*

Do lab 7.3, using REP and MOVSB as instructed. Here, you will start off with a source string (call it SOURCE, for example), move its contents to a destination string (DESTIN, for example) that is initially defined using...

```
.DATA  
DESTIN DB DUP(?) ;Open ended array of bytes
```

Place the length of the source array into another variable (LENGTH, for example). Write the DESTIN string to the screen in consecutive locations.

DEMONSTRATE THIS PROGRAM _____ *sign-off*

Do lab 7.4, storing your 8-byte array of doubled numbers into a destination array. Write this array to screen, but place a comma (ASCII code 2Ch) and a space bar (ASCII code 20h) between each element on the screen.

Example screen write:

```
01, 02, 04, 08, 10, 20, 40, 80
```

DEMONSTRATE THIS PROGRAM _____ *sign-off*

Now, modify your previous program to accept a keystroke, ranging from 1 to 9, which will define the length of the array. For example, if the '3' key is pressed, the following will be printed to screen

```
01, 02, 04
```

DEMONSTRATE THIS PROGRAM _____ *sign-off*