EXPERIMENT TEN: 
DATA CONVERSIONS

INTRODUCTION

So far, only standard ASCII data has been output to the display or read from the keyboard. In many cases this is adequate, but at times we also need to read numeric data in the form of decimal or hexadecimal data. This experiment details the use of both decimal and hexadecimal data in a program and develops procedures or macros that can be used for conversion and display of this type of data. It also details how data in virtually any number base can be read or displayed.

OBJECTIVES

1. Read hexadecimal data from the keyboard and assemble it as a byte or word.
2. Read decimal data from the keyboard and assemble it as a word.
3. Display hexadecimal data on the video display.
4. Display decimal data on the video display.

PROCEDURE

There are many ways to convert between decimal and another number base. This experiment concentrates on one method to accomplish conversions between any number base and decimal. Because the software developed converts between decimal and any number base, the software developed here is as general as possible and as flexible as possible.

Displaying data in any number base.

The basic premise behind displaying data in any number base is division. If a binary number is divided by 10, and the remainder of the division is saved as a significant digit in the result, the remainder must be a number between 0 and 9. On the other hand, if a number is divided by 8, the remainder must be a number between 0 and 7. Because of this, the resultant remainder will be of a different number base than the input which is base 2. To convert from binary to any other base, use the following algorithm:

1. Divide the number to be converted by the desired radix (number base).
2. Save the remainder as a significant digit of the result.
3. Repeat steps 1 and 2 until the resulting quotient is zero.
4. Display the remainders as digits of the result, note the first remainder is the least significant digit, while the last is the most significant.

Example 10-1 applies this algorithm in the form of a macro sequence (could be a procedure if needed), which converts the binary contents of AX into any number base that is displayed on the video display. This macro assumes that any number base beyond base 10 uses the letters A through Z for the digits beyond 9. This allows the conversion and display from binary to any number between base 2 through 37. Yes, this will display a binary number if the selected base is 2.

**Example 10-1**

```
[NUM] MACRO RADIX  ;display in AX in radix
LOCAL N1,N2,N3
PUSH BX  ;;save registers
PUSH DX
MOV BX,RADIX
PUSH BX  ;;indicate end of number
N1:
MOV DX,0  ;;clear DX
DIV BX    ;;divide DX:AX by radix
ADD DL,30H  ;;convert to ASCII
.IF DL > 39H
.ADD DL,7
.ENDIF
PUSH DX  ;;save remainder
CMP AX,0
JNE N1  ;;if quotient is not zero
MOV AH,2
N2:
POP DX  ;;get digit
CMP DX,BX
JE N3  ;;if done
INT 21H
JMP N2  ;;repeat until done
N3:
POP DX  ;;reload registers
POP BX
ENDM
```

Place this macro into your macro file MACS.INC or use the PROC and ENDP statements to form it into a procedure. Once accomplished, this macro is accessible by any program. Notice how the .IF and .ENDIF statements test DL to add an additional 7 to DL if the value in DL is greater than 39H (ASCII-coded 9). For example, if DL = 3AH, a 7 is added to form 41H (ASCII-coded A). This is required so number systems greater than base 10 can be displayed by this macro. To display the contents of AX in base 10 use the macro _NUM 10. To display the contents of AX in base 16 use the macro _NUM 16, and so forth.
**STEP 1:** Using the macro sequence developed in Example 10-1, write a program that loads AX with a 1A34H and displays it in hexadecimal, decimal, octal, and binary forms. The values displayed are:

- Hexadecimal: __________
- Decimal: __________
- Octal: __________
- Binary: __________

It may look better if you display each of these values on separate lines or at least separated by a space or two. Note that this macro (_NUM) does not save the contents of AX as it displays it.

**STEP 2:** Now that this macro functions correctly, write a program that displays the hexadecimal contents of memory beginning at location F800:0000 and ending at location F800:000F. Use the macro developed in Example 10-1. Make sure that one space is placed between each number as they are displayed. Could this macro be used to develop a memory dump that appears exactly as the one generated by the D (dump) command in DEBUG? (Explain your answer.)

Another method of displaying hexadecimal data appears in the macro listed in Example 10-2. This macro may be needed to display hexadecimal data that includes leading zeros. Step 2 just showed that the _NUM macro suppresses leading zeros. For example, if _NUM 16 displays a 0AH, it appears as a single A. The leading zero is suppressed. The _HEXAL macro, listed in Example 10-2, displays the 2-digit contents of AL in hexadecimal format without suppressing leading zeros.

### Example 10-2

```
.HEXAL MACRO
    PUSH AX
    SHR AL,1          ;position left digit
    SHR AL,1
    SHR AL,1
    SHR AL,1
    ADD AL,30H        ;convert to ASCII
    .IF AL > 39H
    ADD AL,7
    .ENDIF
    DISP AL          ;use DISP macro
    POP AX
```

STEP 3: Rewrite the program in step 2 so it uses the macro listed in Example 10-2 in place of the _NUM 16 macro of Example 10-1. Execute this program and compare the displayed output with that obtained in step 2.

**Reading data in any number base.**

To read keyboard data in any number base and convert it into binary, multiply by the number base to generate the binary result. As data is read from the keyboard, a 30H is subtracted from numbers and a 37H from letters to generate a binary equivalent from the key code. To assemble the binary key codes, the following algorithm is used:

1. Start with a result of zero.
2. Read a key from the keyboard and make sure it is uppercase, if a letter, and in the correct range. Subtract 30H from a number (0-9) and 37H from a letter (A-Z). If the key is a space, comma, tab, or carriage return (enter) end the macro or procedure.
3. Multiply the result, cleared in step 1, by the number base desired.
4. Add the binary value of the key to this result and repeat steps 2 and 3.

Because of the amount of work that this macro must accomplish to read a number in any number base, it may appear slightly longer than other macros covered in this manual. As with all other macros, a procedure can be used in place of the macro. Example 10-3 lists the macro called _RNUM, which uses one parameter to transfer the desired number base to the macro. Notice how this macro sequence deletes any unwanted or invalid keystroke. Also notice how, the .IF and .ENDIF statements are used many times to make a lowercase letter uppercase; check for the space, comma, tab, or enter keys; check for a number or letter; and test to see if the number or letter typed is within the correct range. Notice how the AND (&&), OR (||), and equal (==) operators are used in this macro as well as others. The macro also uses DOS INT 21H function 07H to read a key without echo so invalid characters can be easily deleted.

**Example 10-3**

```
_RNUM MACRO RADIX ; read a number in any base to AX
LOCAL R1,R2
PUSH DX
PUSH BX
PUSH BP
MOV BX,RADIX
MOV BP,0 ; clear result
R1:
```

```assembly
AND AL, OFH
ADD AL, 30H
.IF AL > 39H
ADD AL, 7
.ENDIF
DISP AL
ENDM
```
STEP 4: Using the macro of Example 10-3, develop a program that reads a number from the keyboard in decimal format and displays it as decimal, hexadecimal, octal, and binary. Use the macro developed in Example 10-1 (_NUM) to display the number as read by _RNUM 10 in these various number bases.

STEP 5: Write a program that accepts two hexadecimal numbers (_RNUM 16) and displays the decimal sum (_NUM 10). To make this program accept the plus sign and equal signs as delimiters, change the .IF statement in _RNUM by modifying the .IF AL == ',' || ... statement with an .IF AL == '+' || AL == '=' || AL == ',' ... statement. This allows the plus sign and equal sign to be used to end a number in addition to the space, tab, comma, or enter keys. The format of this program's display, appears in Example 10-4, which must do repeated additions until the comma key is typed.
Example 10-4.

1A+20=58
3C1+3A=1019

QUESTIONS

10-1. Explain how the .IF AL == 10H instruction functions.

10-2. What is the operator || used for in an .IF statement?

10-3. What is the operator && used for in an .IF statement?

10-4. Which relational operator is used for greater than or equal?

10-5. Develop a program that uses the macro presented in this experiment that reads binary data and displays them in base 11 format.

10-6. Write a program that reads a decimal number of 123 and displays it in base 20 format. What is the result in base 20?

10-7. What is the purpose of the .IF AL >= 'a' && AL <= 'z' statement in example 10-3?

10-8. What is the purpose of the .IF AL > '9' && AL < 'A' statement in Example 10-3?

10-9. What is the purpose of the .IF AL >= BL statement in Example 10-3?

10-10. What is the purpose of the == operator used with the .IF statement?