CIS40 EXERCISE 8

NAME

In this exercise you will unpack previously packed hex information so it can be converted to Ascii and printed. The packed numbers reside in variables named Num1_L, Num1_H, & Num2_L and Num2_H. These form two 32 bit numbers that will eventually be added. The two raw 32 bit binary numbers must be separated into eight 4 bit hex digits each that are stored in the original input Databuffer. The process of extracting the 4 bit hex numbers from the 16bit variable number halves is called Unpacking. The UNPACK_HEX_1 template accepts two number halves (one 32 bit number) as pass by reference stack parameters. Finish implementing UNPACK_HEX_1 routine as defined in the Subroutine section below, Refer to exercise 4 for using a rotate command and extracting register information using a mask operation (hint here you will be extracting 4 bits at a time). If you want to see how it should work change the name of UNPACK_HEX_1 to UNPACK_HEX in the MAIN routine, then you can temporarily borrow the UNPACK_HEX from the hexcnvt.lib. Instructions on how to add the library to your project are noted below.

```
        .MODEL  small

; These prototype routines are part of hexcnvt.lib
; NOTE under Options/Link Options then in Additional Global Libraries add hexcnvt.lib
; (make sure that this library is copied to the masm611\lib directory)
; this library has the all the subroutines invoked in the main routine.
HEX_TO_ASCII PROTO
ASCII_TO_HEX PROTO
PACK_HEX     PROTO
UNPACK_HEX   PROTO
PRINT_BUFFERPROTO

;this prototype is for your own version of the UNPACK_HEX routine
UNPACK_HEX_1        PROTO

        .STACK

        .DATA
;Define data variables for PACK_HEX
DataBuffer    BYTE 200 DUP(?)
Num1H         WORD 0
Num1L         WORD 0
Num2H         WORD 0
Num2L         WORD 0
SizeNum1      WORD 0
SizeNum2      WORD 0

;Define data variables for ADD32/SUB32
Result_L      WORD 0
Result_H      WORD 0
;;
        .CONST
cstCR         EQU 0Dh
cstLF         EQU 0Ah
cstSPACE      EQU 20h
cstEOL        EQU 24h
```

```
        .CODE

        .STARTUP
MAIN    PROC
        MOV     SI,OFFSET DataBuffer

WAITFORLF:
        MOV     AH,1            ;read ascii with echo command
        INT     21H             ;execute interrupt command
        CMP     AL,cstCR        ;check if a carriage return
        JE      DATAENTERED             ;if is, then enter was pressed
        INVOKE          ASCII_TO_HEX ;else convert ascii char to hex
;       CMP     AH,1            ;check if entered char outside hex
;       JE      ERROR_MESSAGE       ;TBD implement error message
        MOV     [SI],AL         ;save converted hex value
        INC     SI              ;point to next DataBuffer position
        JMP     WAITFORLF       ;get next input character

DATAENTERED:
        MOV     AH,2            ;write to video w/echo command
    MOV         DL, cstLF       ;add line feed to move to next line
    INT         21H

        MOV     AL,cstSPACE     ;place a space after entered data
        MOV     [SI],AL
        INC     SI

        MOV     AH,0            ;read ascii without echo command
        INT     16H
        CMP     AL,'+'
        JE      ADD_DATA
        JMP     WAITFORLF

ADD_DATA:
        MOV     AL,cstEOL
        MOV     [SI],AL

        MOV     AX,OFFSET SizeNum1          ;BP+16
        PUSH    AX
        MOV     AX,OFFSET SizeNum2          ;BP+14
        PUSH    AX
        MOV     AX,OFFSET Num1L             ;BP+12
        PUSH    AX
        MOV     AX,OFFSET Num1H             ;BP+10
        PUSH    AX
        MOV     AX,OFFSET Num2L             ;BP+8
        PUSH    AX
        MOV     AX,OFFSET Num2H             ;BP+6
        PUSH    AX
        MOV     AX,OFFSET DataBuffer        ;BP+4
        PUSH    AX

        INVOKE          PACK_HEX
```

```
        MOV     AX,OFFSET Num1L             ;BP+8
        PUSH    AX
        MOV     AX,OFFSET Num1H             ;BP+6
        PUSH    AX
        MOV     AX,OFFSET DataBuffer        ;BP+4
        PUSH    AX

        INVOKE          UNPACK_HEX_1

        MOV     AX,OFFSET DataBuffer        ;BP+4
        PUSH    AX
        INVOKE          PRINT_BUFFER

        MOV     AX,OFFSET Num2L             ;BP+8
        PUSH    AX
        MOV     AX,OFFSET Num2H             ;BP+6
        PUSH    AX
        MOV     AX,OFFSET DataBuffer        ;BP+4
        PUSH    AX
        INVOKE          UNPACK_HEX_1

        MOV     AX,OFFSET DataBuffer        ;BP+4
        PUSH    AX
        INVOKE          PRINT_BUFFER

MAIN    ENDP
        .EXIT
```

```
;*************** SUBROUTINES ********************
;packed hex to unpacked hex conversion routine
;inputs NumxL,NumxH
;outputs DataBuffer pointer to unpacked hex numbers

; A select case structure has been set up for retrieving the passed parameters off of the stack
; you must create a loop that breaks down the number half information that is stored in the AX
;register (the case structure pulls the information from the stack into AX one half number at a time)
;the upper half with leading zeros is generated then the lower half and saved to the Databuffer [DI]

UNPACK_HEX_1        PROC            NEAR
        PUSH    BP              ;Save BP, NOW IP=BP+2
        MOV     BP,SP           ;Get current stack address
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    DI

        XOR     CX,CX
        MOV     DI,[BP+4]       ; OFFSET DataBuffer
        MOV     DH,2            ;outer loop

SELECT_CASE_UNPACK_NumX:
        CMP     DH,2
        JNE     CASE_ELSE_UNPACK_NumxL
CASE_UNPACK_NumxH:
        MOV     BX,[BP+6]               ;OFFSET NumxH
        MOV     AX,[BX]
        JMP     UNPACK_LOOP
CASE_ELSE_UNPACK_NumxL:
;assume DH = 1
        MOV     BX,[BP+8]               ; OFFSET NumxL
        MOV     AX,[BX]

UNPACK_LOOP:

;*****************ADD UNPACK LOOP CODE HERE******************

        MOV     AL,cstSPACE             ;add space after  number
        MOV     [DI],AL
        INC     DI
        MOV     AL,cstEOL               ;add end of line
        MOV     [DI],AL
DONE_UNPACKING:

        MOV     BP,[BP]                 ;restore original stack position to BP
        POP     DI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        POP     BP                      ;restore BP from stack for calling routine
        RET
UNPACK_HEX_1        ENDP
        END             ;End of program
```