NAME

CIS240/CNET220 EXERCISE 8
(using CALL statement version)

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 hexcnv32.lib. Instructions on how to add the library to your project are noted below. Note that the template below forms the bases of a calculator project, and uses a "+" sign prompt indicating that an add operation could be programmed in later.

```
TITLE MASM main                                                    (main.asm)
;.686P              ; Pentium Pro or later
;.MODEL flat, stdcall
;.STACK 4096

INCLUDE Irvine32.inc

; These prototype routines are part of hexcnv32.lib.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_BUFFER PROTO
CONV_BUFFER  PROTO
CLRBUF proto NEAR C ,address:dword, lengthbuf:word

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

;Define data variables for PACK_HEX
;DataBuffer    BYTE 200 DUP(?)
Num1H          WORD 0              ;32bit high word value
Num1L          WORD 0              ;32bit low word value
Num2H          WORD 0              ;32bit high word value
Num2L          WORD 0              ;32bit low word value
SizeNum1       WORD 0              ;number of hex digits of 32bit number 1
SizeNum2       WORD 0              ;number of hex digits of 32bit number 2
```

```
;Define data variables for ADD32/SUB32
Result_L        WORD 0
Result_H        WORD 0

asciibuffer     BYTE   200 DUP(0)       ;input buffer

prompt1 BYTE "Type a hex number>",0
prompt2 BYTE "Type + to calculate or enter to accept another number>",0;
; Note add more variables if needed
;
;CONSTANTS

cstCR           = 0Dh           ;ascii carriage return char
cstLF           = 0Ah           ;ascii line feed char
cstSPACE        = 20h           ;ascii space bar char
cstEOL          = 24h           ;end of string of chars

        .CODE

MAIN    PROC

        mov     esi,OFFSET asciibuffer

NEWNUMBER:

        mov edx, OFFSET  prompt1        ;load location of buffer
        call writestring

WAITFORDIGIT:
        call    readchar        ;read char from keyboard
        call    writechar               ;echo it to monitor
        cmp     al,cstCR        ;look for carriage return
        JE              NUMBERENTERED   ;if CR then number entered

        call    ASCII_TO_HEX ;AH returns 1 if error
;       cmp     ah,1
;       call    ERROR_HANDLER   ;this routine needs to be written
        mov     [esi],al        ;save converted character
        inc     esi                     ;point to next char to save
        jmp     WAITFORDIGIT    ;jmp to get next char

NUMBERENTERED:
        call    crlf                    ;go to next line

        mov     al,cstSPACE             ;add a space between numbers
        mov     [esi],al
        inc     esi

        mov     edx, OFFSET  prompt2    ;load location of prompt string
        call    writestring             ;display prompt

        call    readchar        ;read char for next decision
        cmp     al,'+'
        je              ADD_DATA        ;if char= + calculate numbers
        call    crlf
```

```
        jmp     NEWNUMBER
ADD_DATA:
        mov     al,cstEOL               ;end msg string with a null
        mov     [esi],al
        mov     esi,OFFSET asciibuffer  ;Point to beginning of outbuffer

;pack hex converts alls the hex digits to packed hex so that the
;numbers can be used in calculations.  The asciibuffer actually
;contains hex data which was input. The routine pack_hex determines
;the length of the numbers and separates them into two 16 bit halves
;that are stored as Num1L and Num1H for the first 32 bit number
;and Num2L and Num2H for the second 32 bit number.
;sizenum1 is the number of digits in the combined Num1L + Num1H
;sizenum2 is the number of digits in the combined Num2L + Num2H

        mov             eax,OFFSET SizeNum1   ;eBP+32;return value
        push    eax
        mov             eax,OFFSET SizeNum2   ;eBP+28;return value
        push    eax
        mov             eax,OFFSET Num1L      ;eBP+24;return value
        push    eax
        mov             eax,OFFSET Num1H      ;eBP+20;return value
        push    eax
        mov             eax,OFFSET Num2L      ;eBP+16;return value
        push    eax
        mov             eax,OFFSET Num2H      ;eBP+12;return value
        push    eax
        mov             eax,OFFSET asciibuffer ;eBP+8  ;input value
        push    eax

        call    PACK_HEX

;after pack hex is completed separting the string of numbers
;into separate variables (Num1L,Num1H, Num2L,Num2H)  they can
;be used in calculations.  The calculation for ADD32 for example
;will take Num1L & Num1H and add them to Num2L & Num2H in 16 bit
;halves (see example code from notes) then save the result back
;into 16 bit halves (you can use the same variables as the input
;data to also store the results - but it overwrites the inputs -
;so there are more variables defined at the top for the
;results of calculations)

;       *****WRITE YOUR CALCULATION CODE HERE****

;when your calculations are complete your results will be stored
;into variables, but for this example below (since no calculation
;actually occured) we are just going to display back the original
;data.  In order to display it back it must be converted back to
;ascii.  This is a two step process, 1st the numbers must be
;converted to back to unpacked hex.  Then the Print buffer routine
;converts the unpacked hex for each number to ascii and writes it
;to the monitor

;the unpack hex routine takes the packed hex from the result
;in this case Num1L & Num2L unpackes them and stores them into
;asciibuffer (however the asciibuffer won't have ascii yet just
```

```
;unpacked hex)
        mov             eax,OFFSET Num1L        ;eBP+16;input value
        push    eax
        mov             eax,OFFSET Num1H        ;eBP+12;input value
        push    eax
        mov             eax,OFFSET asciibuffer  ;eBP+8  ;return value
        push    eax

;clrbuf clears extra characters that cause writestring to fail
        invoke  clrbuf, offset asciibuffer, sizeof asciibuffer

        call    UNPACK_HEX

;conv_buffer calls the hex to ascii convertion for each digit
;then the writestring routine prints the buffer
        mov             eax,OFFSET asciibuffer  ;eBP+8  ;input value
        push    eax
        call    CONV_BUFFER
        mov             edx, OFFSET asciibuffer ;load location of buffer
        call    writestring

;this second conversion won't be needed if you actually had
;performed a calculation such as ADD32 since there would only
;have been one result, so this next operation just displays back
;the second number that was originally entered
        mov             eax,OFFSET Num2L        ;eBP+16;input value
        push    eax
        mov             eax,OFFSET Num2H        ;eBP+12;input value
        push    eax
        mov             eax,OFFSET asciibuffer  ;eBP+8  ;return value
        push    eax
        INVOKE          CLRBUF, offset asciibuffer, sizeof asciibuffer

        call    UNPACK_HEX


        mov             eax,OFFSET asciibuffer  ;BP+8
        push    eax
        call    CONV_BUFFER
        mov             edx, OFFSET asciibuffer ;load location of buffer
        call    writestring
        call    crlf
        exit

MAIN    ENDP
```

```
;******************SUBROUTINES*********************
; NOTE In the make32.bat file invokes Link.exe which links the hexcnv32.lib
; to the main routine for any of the subroutines that are used
;   (make sure that this library is in the
; masm611\lib directory) this library has the all the subroutines
; invoked in the main routine.

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

UNPACK_HEX_1        PROC            NEAR
        PUSH    eBP             ;Save eBP, NOW eIP=eBP+4
        MOV     eBP,eSP         ;Get current stack address
        PUSH    AX
        PUSH    eBX
        PUSH    CX
        PUSH    DX
        PUSH    eDI

;Packed hex Inputs
        MOV     eBX,[eBP+12]    ;OFFSET to  Number High word (NumxH)
        MOV     AX,[eBX]        ; Packed Number High word value
        MOV     eBX,[eBP+16]    ; OFFSET to Number Low Word (NumxL)
        MOV     AX,[eBX]        ; Packed Number Low word value

;Unpacked hex return values need to be stored in this buffer
        MOV     eDI,[eBP+8]     ; OFFSET to unpacked DataBuffer


UNPACK_LOOP:

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

LOOP_END:


MOV     AL,cstSPACE             ;add a space after 8 byte number
MOV     [eDI],AL
INC     eDI
MOV     AL,cstEOL               ;add end of line
MOV     [eDI],AL

DONE_UNPACKING:

        POP     eDI
        POP     DX
        POP     CX
        POP     eBX
        POP     AX
        POP     eBP             ;restore BP from stack for calling routine
        RET     12              ;clean up stack
UNPACK_HEX_1        ENDP
```

```
;******* Example code called in main routine*******

CONV_BUFFER          PROC   NEAR   PUBLIC
        PUSH   eBP              ;Save eBP, NOW eIP=eBP+4
        MOV    eBP,eSP          ;Get current stack address
        PUSH   AX
        PUSH   eDX
        PUSH   eSI

        MOV    eSI,[eBP+8]      ;Retrive Pointer to beginning of data buffer
CONV_LOOP:
        MOV    AL,[eSI]
        CMP    AL,cstEOL
        JE     CONV_DONE
        call   HEX_TO_ASCII
        MOV    [eSI],AL
  INC    eSI
        JMP    CONV_LOOP
CONV_DONE:
        mov    al,0             ;this step overwrites the cstEOL (24h) with 0
        mov    [esi], al        ;the writestring routine needs a 0 at the end

        POP    eSI
        POP    eDX
        POP    AX
        POP eBP                 ;restore BP from stack for calling routine
        RET    4
CONV_BUFFER          ENDP

;this routine clears a memory buffer where “address” is the buffer address and “lengthbuf” is the size of the
buffer
CLRBUF   PROC NEAR C USES eax ebx edx,address:dword, lengthbuf:word
        xor ecx,ecx
        MOV    CX,lengthbuf
        mov ebx,address
        mov al,0
CLRLOOP:
        MOV [ebx],AL
        INC ebx
        loopne CLRLOOP
        ret
CLRBUF          ENDP

        END  main               ;End of program
```