CpE 390 Microprocessor Systems

Lab 3: Data Structures and Subroutines

1. Searching for a numeric key in an array

The following code is similar to that given in Lecture 7 as an example of how one might search for a particular key (in this case, the numeric value $190) in an array of 16-bit integers. The array in this example (*vecx*) has been preloaded with fifteen 16-bit hex values. The program searches for the key in the array. If it finds the key, it stores the array index of the word that matches the key in the location named *result*. If it does not find the key, it stores the "*not found*" code, which is defined as $FF (or -1)

```
N:          EQU     15              ; length of array
NFC:        EQU     $FF             ; not-found code
key:        EQU     $190
CR:         EQU     $0D             ; ASCII return
LF:         EQU     $0A             ; ASCII line feed

            ORG     $5000
result:     DS.B    1               ;reserve a byte for result
vecx:       DC.W    $D1A, $B5, $39F, $980, $E4F, $186, $E3, $319, $430
            DC.W    $4, $190, $22C, $189, $A55, $30D
str1:       DC.B    "Key found", CR, LF, 0
str2:       DC.B    "Key not found", CR, LF, 0

            ORG $4000
            clrb                    ; initialize index = 0
            movb    #NFC, result    ; initialize search result
            ldy     #key            ; set key we're searching for
            ldx     #vecx           ; set up X as pointer to array
loop:       tfr     B, A            ; copy index to A
            lsla                    ; and multiply by 2 to give byte offset
            cpy     A, X            ; compare key to array element
            beq     found           ; branch if element = key
            incb                    ; if not – increment index
            cmpb    #N              ; are we at the end of the array yet?
            bne     loop            ; no – go check next value
            ldx     #str2           ; yes - we're done without finding key
            jsr     putstr
            bra     done
found:      stab    result          ; write index into result
            ldx     #str1
            jsr     putstr
done:       swi                     ; return to monitor
```

There are two differences to the code we used in the lecture. The first is that we are using hex numbers instead of decimal, because it's easier to do input and output using hex characters. The second difference is that we are calling a subroutine *putstr* to output a string message on the terminal. This message will tell us whether or not the program found the specified key. Don't worry too much about how this routine works – we haven't done serial I/O in class yet. All you need to know is that you pass a pointer to the string you want to print out in register X.

(a) Enter the program into MiniIDE.

(b) Append the following output routines to the end of your program

```
SCIOSR1:    EQU     $00CC
SCIODRL:    EQU     $00CF

putstr:     psha                        ; output null terminated string to terminal
ploop:      ldaa    1, X+               ; X contains pointer to string
            beq     pdone
            jsr     putc
            bra     ploop
pdone:      pula
            rts

putc:       brclr   SCIOSR1, $80, *     ; output single character to terminal
            staa    SCIODRL
            rts
```

(c) Assemble and download your program

(d) Run the program and determine whether the key was found in the array. Did the correct string print out on the terminal? Check the data in labeled memory location *result* to see if it correctly identified the location of the key in the array.

(e) Change the value of the key (could be a value that *is* or *is not* in the array), reassemble, load and rerun and see if you still get the expected result.

(f) Append the code on the next page to the end of your program. This adds another output routine *puthx8* which will convert the 8-bit value in accumulator A to two hex digits and output them on the terminal.

(g) Add code to your program to print out the value of the index result in the form:

Index = $??

Where "??" are the two hex digits that are the value of the result. You can do this in three steps.

1. Print out the string "Index=$"
2. Print out the hex value of the result
3. Print out CR, LF to end the line of text

```
puthx8:     psha                    ; output 8bit value in acc A as two hex digits
            lsra
            lsra
            lsra
            lsra
            jsr     puthx4
            pula
            anda    #$0F
            jsr     puthx4
            rts

puthx4:     cmpa    #$A             ; output 4-bit value in acc A as a hex character
            blo     hxnum
            adda    #$7
hxnum:      adda    #$30
            jsr     putc
            rts
```

2. Counting characters and words in a string

The following program counts the characters and words in a string – a standard function in any word processing program. The program assumes that the string is terminated by a NULL character and that words are terminated by one or more space characters. The program does not check for illegal text characters or other spaces such as tab, newline etc.

```
    space:      EQU         $20

                ORG         $5000
ch_cnt:         ds.b        1               ;character count
wd_cnt:         ds.b        1               ;word count
str_x:          dc.b        "Peter Piper picked a peck of pickled peppers", 0

                ORG         $4000
                ldx         #str_x              ;X is character pointer
                clr         ch_cnt              ;clear character count
                clr         wd_cnt              ;clear word count
str_lp:         ldab        1,x+                ;read a character
                beq         done                ;end of string?
                inc         ch_cnt              ;increment character count
                cmpb        #space              ;check for space character
                beq         str_lp
                inc         wd_cnt              ;found non-space: must be at start of word
wd_lp:          ldab        1,x+                ;read a character
```

```
                beq         done            ;end of string?
                inc         ch_cnt          ;increment character count
                cmpb        #space          ;check for space character
                beq         str_lp
                bra         wd_lp           ;non-space character is part of word
done:           swi
```

(a) Enter, assemble and download your program

(b) Run the program and determine whether the correct number of words and characters are found. Do spaces count as characters?

(c) Change the sentence in str_x to something of your choosing and re-run the program. Verify that it still runs correctly.

3. Re-write character and word counting function as a subroutine

Now take the code that you just used to count words and characters in a string and convert it into a subroutine. The main program should pass the address of the string in the X register. The word count and character count results should be stored in memory locations *ch_cnt* and w*d_cnt* as before. You will need to complete the following steps:

(a) Take the counting portion of your program and change it to a subroutine by putting a label “count:” at the first executable instruction. Note that you will not need to load the X register as part of the subroutine – the subroutine assumes that the string address has already been loaded into the X register. The first executable instruction in the subroutine will be the one that clears the character count. Right before this first executable instruction, use an ORG directive to place the subroutine at a suitable place in memory.

(b) Place a return from subroutine **rts** instruction at the end of the subroutine.

(c) To your code, add a main program that will load the X register with the correct string address and then call the subroutine *count* that you created in (a) and (b) above. You do not need to set up the stack pointer – the stack has already been set up in memory by the monitor program. Use an ORG directive to place you main program at a suitable place in memory.

(d) Assemble and download your new program.

(e) Run the program. Does it still give the correct result?

(f) Append the output routines we used in Part (1) to your code and use them to print out the values of the word count and character count.