CpE 390 Microprocessor Systems

Lab2: Program Loops and Branching

1. Create a program using the following steps:

a. Set up a couple of symbols to define where your data and code will be located with the following memory map equates:

```
PROG:   EQU     $7000
DATA:   EQU     $4000
```

b. Set up 4 variable data buffers of 15 bytes each as shown in the assembler directives below. Use the labels BUF1, BUF2, BUF3 and BUF4 to refer to them in your program.

```
        ; variable data storage
        ORG     DATA        ; locate BUF1 at address defined by DATA
BUF1:   DS.B    15          ; allocate 15 bytes of storage starting at DATA
        ORG     DATA+$10    ; locate BUF2 on next 16-byte boundary
BUF2:   DS.B    15          ; 15 bytes of storage starting at DATA+$10
        ORG     DATA+$20
BUF3:   DS.B    15
        ORG     DATA+$30
BUF4:   DS.B    15
```

c. Write a program to copy data from a constant buffer labeled DATA1 to the first of your variable buffers BUF1. The program should set up a loop to transfer the data one byte at a time. It should use pointers and post-increment addressing to scan the buffers. You could, for example, use the following code:

```
        ldaa    #15         ; accumulator A holds count value
        ldx     #DATA1      ; X points to source buffer
        ldy     #BUF1       ; Y points to destination buffer
loop:   movb    1, x+, 1, y+ ; copy one byte and increment pointers
        dbne    a, loop     ; are we done yet?
```

d. Finally, add the following fixed data buffers to the end of your program:

```
            ;constant data used by program
DATA1:      DC.B   $48,$41,$56,$45,$20,$41,$20,$47,$4F,$4F,$44,$20,$44,$41,$59
DATA2:      DC.B   $0F,$04,$F6,$FE,$2F,$0C,$25,$D9,$05,$00,$DC,$2C,$1D,$21,$1A
```

e. Your overall program structure should look something like:

```
PROG        EQU         $7000
DATA        EQU         $4000

ORG         DATA
…
(variable data buffers here)
…
ORG         PROG
…
(write your program here)
…
swi
(put your constant data buffers here)
```

f. Assemble, load and run your program to copy the 15 bytes of data from DATA1 to BUF1. Use the **md** command to examine the contents of BUF1 to check that your program has executed correctly. *(Note: The data values in DATA1 are a string of ASCII characters which spell out "HAVE A GOOD DAY")*

g. Add code to the end of your program (before the **swi** instruction) so that once it has completed copying DATA1 to BUF1, it then copies DATA2 to BUF2. Check the result. Do the characters copied into BUF2 spell out anything useful?

h. Again, add code to the end of your program so that once it has copied DATA2 to BUF2, it then adds each byte of BUF1 to the corresponding byte of BUF2 and stores the result in BUF3, i.e. BUF3[i] = BUF1[i] + BUF2[i] for each i. *(Hint: Since we only have two index registers, it's hard to do this with a single loop. It's easier to do this with two loops. The first copies BUF1 into BUF3. The second adds BUF2 to BUF3)* Examine the result. Describe what has happened?

i. Now add to your program so that once it has generated the character string in BUF3, it then exclusive-OR's each byte in BUF3 with the number $20 and stores the result in the corresponding byte of BUF4. You can use the **eora** or **eorb** instruction to do the exclusive-OR operation. What result do you get in BUF4? What has happened?

j. When you have all sections of the code running correctly, record accurately the contents of BUF1, BUF2, BUF3 and BUF4 in your report in the following format:

```
BUF1:       $48,$41,$56,$45,$20,$41,$20,$47,$4F,$4F,$44,$20,$44,$41,$59
BUF2:       $0F,$04,$F6,$FE,$2F,$0C,$25,$D9,$05,$00,$DC,$2C,$1D,$21,$1A
BUF3:          (content of BUF3)
BUF4:          (content of BUF4)
```

2. Now it's time to write your own program:

(a) Set up a data buffer labeled **DATA** containing 20 8-bit numbers at memory address $5000.
(b) Declare two 8-bit variables **MAXS** and **MAXU** at memory locations $5020 and $5021 respectively.

You can use the following code to set up these labeled memory locations:

```
      ORG $5000

DATA: DC.B $23, $74, $55, $C2, $00, $17, $F3, $BD, $A8, $38
      DC.B $67, $97, $66, $08, $7A, $EF, $4B, $43, $1A, $28

      ORG $5020

MAXS: DS.B 1
MAXU: DS.B 1
```

(You don't have to use the data values I have provided for DATA – you can use your own if you wish)

(c) Now write a program starting at address $6000 to find the maximum *signed* value in **DATA** and store that result in **MAXS**. Run the program and see if you get the correct result.
(d) Now modify your program to <u>also</u> find the maximum *unsigned* value in **DATA** and store this result in **MAXU**. Check your result.
(e) Show a copy of your code and the results you obtained in your report.