LC3 Assembly Programming
Week 8 Lab
Exercise:
Subroutines and Input

Show demo to one of the instruction team

Exercise

- Write LC3 assembly program that uses subroutines, loops through array MyArray of N numbers, multiplies each by integer *y*, writes into new array OutArray stored starting at x5000.
- MyArray stored starting at x4000 and N is variable initialized to 8
  - myArray stores values 10, 20,30,…,70,80
- Read value *y* from input where *y* is between 0 and 9
  - Convert the ASCII character to binary
    - How ? Subtract x30 (add -48) ASCII character read into R0
- Multiplication is a subroutine
- Loop through MyArray
  - Call Mult to multiply each element by *y*
  - Store into output array OutArray which is stored starting at address x5000
- Print message “Completed Multiplication:
- Halt program

C code:

```
n=8;
printf(“Enter value of y \n”);
scanf(%d, &y);
while i>0 {
      outArray[i]= myArray[i] * y;}
printf(“Completed Multiplication\n”);
```

More specifications

- MyArray starts at x4000
  - Use same technique from last lab to define MyData and load into program
- OutArray starts at x5000
- Program prints “Enter number Y” then prompts for input from keyboard: y is a between 0 and 9.
- Program calls subroutine Mult to multiply elements by y
  - Input to Mult is passed through registers R1, R2
  - Output from Mult is in register R3
- After looping through array of N values (N=8 in this case), print message “Completed Multiplication”
- Halt

Creating and Loading a "data" file

- LC3Tools permits loading multiple object files
  - Loaded at the address specified in that object file (i.e., .ORIG command)
- Can use this to create and load a file containing the data to be processed by your code.
- Ex: MyData.asm is a list of numbers starting at address x4000
- Assemble the code – creates object code MyData.obj
- Load this object file into simulator
  - Important: make sure you reset program counter is set to start of your main program. (Or – load data first and then load program)

MyData.asm

```
.ORIG x4000
.FILL #10
.FILL #20
.FILL #30
.FILL #40
.END
```

Put values 10,20,30,40
at addresses x4000, 4001, 4002, 4003
Respectively.
Loading MyData.obj will result in these
Values in those memory addresses of
the simulator

Tips….

- Remember to place a breakpoint at the Halt instruction
- Rewrite your multiplication code so that it is a subroutine MULT
  - Inputs are passed through registers R1, R2
  - Output computed in register R3
- Remember to save and then restore registers R1,R2, R3 in your "main" before/after calling subroutine MULT
- Define .STRINGZ to hold the messages to print to display
  - msg1 .STRINGZ "Enter number Y"
  - msg2 .STRINGZ "Completed Multiplication"
- How do you load start of this string to R0 (before calling PUTS?)
  - LEA R0, msg1; copy address of msg1 into R0 works if msg1 is close enough
  - Else ??? Here is one trick:

```
        link    .FILL   msg     ; variable link contains address of msg
In code         LD R0, link     ; loads address of msg into R0
```

Arranging code with subroutines: Observations

```
; start main
        LD R5, temp
        ; other code
        JSR Mult ; call subroutine
        …
        HALT
temp    .FILL 8
MyArray   .FILL x4000
OutArray   .FILL x5000
msg1    .STRINGZ “Enter number Y”
msg2
Mult    …       ; first instruction of Mult
        ..
        RET     ; return from Mult
Who1    .BLKW #1 ; random var for Mult
;
XOR     …       ; first instruction of XOR
        ..      ; body of XOR
        RET     ; return from XOR
```

Code written so that PC never gets to these addresses

In Mult, when you hit RET instruction Program returns to main – does not continue past RET but can load from label Who1

High level programs Analogy

```
int foo (int x) { /* function definition for foo */
…
      return(j);}      /* return from foo */

int bar (int x,y){ /* function def for bar */
…
      return (z);} /* return from bar */
/* start of main */
int main() {     /* start of main */
      int a,b,c;
      b= foo(a);       /* call foo from main */
      a= bar(b,c);     /* call bar from main */
      return 0; /* end of main */
}
```

Reference: assembly program for Multiplication

```
; code to multiply two integers, num1 and num2 stored in
; memory and initialized to x8 and x8
.ORIG x3000
       AND R6, R6, #0       ;clear R6, it will hold the result
       LD R3, num1   ; load first number into R1
       BRz done      ; if number is zero then done
       LD R4, num2   ; load second number into R2
       BRz done      ; if number2 is zero then done
       ; else loop through R3 times adding R4 to
       ; itself – i.e., add R4 to value in R6 (product)
loop   ADD R6, R6, R4       ; add R4 to current product
       ADD R3, R3, #-1      ; decrement counter
       BRp Loop      ; if counter >0 then repeat loop
done   HALT   ; else halt, value of product is in R6

num1   .FILL x8
num2   .FILL  x5
       .END
```