HW6

T1

The main program below calls a subroutine `F`. The `F` subroutine uses R3 and R4 as input, and produces an output which is placed in R0. The subroutine modifies registers R0, R3, R4, R5, and R6 in order to complete its task. `F` calls two other subroutines, `SaveRegisters` and `RestoreRegisters`, that are intended handle the saving and restoring of the modified registers (although we will see in question (b) that this may not be the best idea!).

```
; Main Program;
.ORIG x3000
...
...
JSR F
...
...
HALT
; R3 and R4 are input.
; Modifies R0, R3, R4, R5, and R6
; R0 is the output
;
F
JSR SaveRegisters
...
...
...
JSR RestoreRegisters
RET
.END
```

(a) Write the two subroutines `SaveRegisters` and `RestoreRegisters`.

(b) When we run the code we notice there is an infinite loop. Why? What small change can we make to our program to correct this error. Please specify both the correction and the subroutine that is being corrected.

T2

Assume that you have the following table in your program:

```
MASKS
.FILL x0001
.FILL x0002
.FILL x0004
.FILL x0008
.FILL x0010
.FILL x0020
.FILL x0040
.FILL x0080
.FILL x0100
.FILL x0200
.FILL x0400
.FILL x0800
.FILL x1000
.FILL x2000
.FILL x4000
.FILL x8000
```

(a) Write a subroutine `CLEAR` in LC-3 assembly language that clears a bit in R0 using the table above. The index of the bit to clear is specified in R1. R0 and R1 are inputs to the subroutine.

(b) Write a similar subroutine `SET` that sets the specified bit instead of clearing it.

Hint: You should remember to save and restore any registers your subroutine uses (the "callee save" convention). Use the `RET` instruction as the last instruction in your subroutine (R7 contains the address of where to return to.)

T3

> Adapted from 8.11

The following program needs to be assembled and stored in LC-3 memory.

```
        .ORIG x4000
        AND  R0,R0,#0
        ADD  R1,R0,#0
        ADD  R0,R0,#4
        LD       R2,B
A       LDR  R3,R2,#0
        ADD  R1,R1,R3
        ADD  R2,R2,#1
        ADD  R0,R0,#-1
        BRnp A
        JSR  SHIFTR
        ADD  R1,R4,#0
        JSR  SHIFTR
        ST       R4,C
        TRAP x25
B       .BLKW 1
C       .BLKW 1
        .END
```

(a) How many memory locations are required to store the assembled program?

(b) What is the address of the location labeled C?

(c) Before the program can execute, the location labeled B must be loaded by some external means. You can assume that happens before this program starts executing. You can also assume that the subroutine starting at location SHIFTR is available for this program to use. SHIFTR takes the value in R1, shifts it right one bit, and stores the result in R4.

After the program executes, what is in location C?

T4

> Adapted from 8.13

Our code to compute n factorial worked for all positive integers n. Augment the iterative solution to FACT to also work for 0!.

```
FACT    ST       R1,SAVE_R1
                ADD  R1,R0,#0
                ADD  R0,R0, #-1
                BRz  DONE
AGAIN   MUL  R1,R1,R0
                ADD  R0,R0,#-1  ; R0 gets next integer for MUL
                BRnp AGAIN
DONE    ADD  R0,R1,#0   ; Move n! to R0
                LD       R1,SAVE_R1
                RET
SAVE_R1 .BLKW 1
```

T5

> Adapted from 9.6 & 9.9

(a) What problem could occur if a program does not check the *Ready* bit of the KBSR before reading the KBDR?

(b) What problem could occur if the keyboard hardware does not check the KBSR before writing to the KBDR?

(c) Which of the above two problems is more likely to occur? Give your reason.

T6

> Adapted from P353 9.13

Some computer engineering students decided to revise the LC-3 for their senior project.

In designing the LC-4, they decided to conserve on device registers by combining the KBSR and the DSR into one status register: the IOSR (the input/output status register). IOSR[15] is the keyboard device ready bit and IOSR[14] is the display device ready bit.

What are the implications for programs wishing to do I/O? Is this a poor design decision?

T7

> Adapted from 9.16

(a) How many `TRAP` service routines can be implemented in the LC-3? Why?

(b) How many accesses to memory are made during the processing of a `TRAP` instruction?

T8

> Adapted from P354 9.19

The following LC-3 program is assembled and then executed. There are no assemble time or run-time errors. What is the output of this program? Assume all registers are initialized to 0 before the program executes.

```
           .ORIG x3000
           LEA R0, LABEL
           STR R1, R0, #3
           TRAP x22
           TRAP x25
LABEL  .STRINGZ "FUNKY"
LABEL2 .STRINGZ "HELLO WORLD"
           .END
```

T9

> Adapted from P355 9.21

Assume that an integer greater than 2 and less than 32,768 is deposited in memory location A by another module before the program below is executed.

```
           .ORIG x3000
           AND R4, R4, #0
           LD R0, A
           NOT R5, R0
           ADD R5, R5, #2
           ADD R1, R4, #2
           ;
REMOD  JSR MOD
           BRz STORE0
           ;
           ADD R7, R1, R5
           BRz STORE1
           ADD R1, R1, #1
           BR REMOD
           ;
STORE1 ADD R4, R4, #1
STORE0 ST R4, RESULT
           TRAP x25
           ;
MOD    ADD R2, R0, #0
           NOT R3, R1
           ADD R3, R3, #1
DEC    ADD R2, R2, R3
           BRp DEC
           RET
           ;
A          .BLKW 1
RESULT .BLKW 1
           .END
```

In 25 words or fewer, what does the above program do?

T10

> Adapted from P360 9.31

The program below, when complete, should print the following to the monitor:
ABCFGH
Insert instructions at (a)–(d) that will complete the program.

```
            .ORIG x3000
                LEA R1, TESTOUT
BACK_1  LDR R0, R1, #0
                BRz NEXT_1
                TRAP x21
                ------------ (a)
                BRnzp BACK_1
                ;
NEXT_1  LEA R1, TESTOUT
BACK_2  LDR R0, R1, #0
                BRz NEXT_2
                JSR SUB_1
                ADD R1, R1, #1
                BRnzp BACK_2
                ;
NEXT_2  ------------ (b)
                ;
SUB_1   ------------ (c)
K               LDI R2, DSR
                ------------ (d)
                STI R0, DDR
                RET
DSR     .FILL xFE04
DDR     .FILL xFE06
TESTOUT .STRINGZ "ABC"
                .END
```

T11

> Adapted from P362 9.33

Interrupt-driven I/O:

(a) What does the following LC-3 program do?

```
         .ORIG x3000
         LD R3, A
         STI R3, KBSR
         AGAIN LD R0, B
         TRAP x21
         BRnzp AGAIN
A        .FILL x4000
B        .FILL x0032
KBSR .FILL xFE00
         .END
```

(b) If someone strikes a key, the program will be interrupted and the keyboard interrupt service routine will be executed as shown below. What does the keyboard interrupt service routine do?

```
         .ORIG x1000
         LDI R0, KBDR
         TRAP x21
         TRAP x21
         RTI
KBDR .FILL xFE02
         .END
```

(c) Finally, suppose the program of part a started executing, and someone sitting at the keyboard struck a key. What would you see on the screen?

(d) In part c, how many times is the digit typed shown on the screen? Why?

T12

> Adapted from P364 9.43

Two students wrote interrupt service routines for an assignment. Both service routines did exactly the same work, but the first student accidentally used RET at the end of his routine, while the second student correctly used RTI.

There are three errors that arose in the first student’s program due to his mistake. Describe any two of them.