Adnar Lozano
CSE 313 Section 03
CSUSB Spring 17
06/06/17

Homework # 3

Chapter 7: 7.4, 7.9, 7.10, 7.13

7.4 Create the symbol table entries generated by the assembler when translating the following routine into machine code:

```
        .ORIG   x301C
        ST      R3, SAVE3
        ST      R2, SAVE2
        AND     R2, R2, #0
TEST    IN
        BRz     TEST
        ADD     R1, R0, #-10
        BRn     FINISH
        ADD     R1, R0, #-15
        NOT     R1, R1
        BRn     FINISH
        HALT
FINISH  ADD     R2, R2, #1
        HALT
SAVE3   .FILL   X0000
SAVE2   .FILL   X0000
        .END
```

SYMBOL	ADDRESS
TEST	x301F
FINISH	x3020
SAVE3	x302F
SAVE2	x3030

7.9 What is the purpose of the .END pseudo-op? How does it differ from the HALT instruction?

The .END pseudo-op tells the assembler where the program ends. Any instructions after this pseudo-op are disregarded and not processed by the assembler. It is very different from HALT instruction in a few ways such as:

1. It is not an instruction, which cannot be executed
2. It does not stop the machine or the program
3. It is a marker that tells the assembler where to stop assembling.

7.10 The following program fragment has an error in it. Identify the error and explain how to fix it.

```
        ADD     R3, R3, #30
        ST      R3, A
        HALT
A       .FILL   #0
```

Will this error be detected when this code is assembled or when this code is run on the LC-3?

R3 was not initialized before the ADD R3, R3, #30 instruction; therefore, the result of this instruction may not be correct . This error will be detected during run time and not when is assembled.

7.13 The following program adds the values stored in memory locations A, B, and C, and stores the result into memory. There are two errors in the code. For each, describe the error and indicate whether it will be detected at assembly time or at run time.

```
Line No.
1               .ORIG  x3000
2       ONE     LD  R0, A
3               ADD R1, R1, R0
4       TWO     LD  R0, B
5               ADD R1, R1, R0
6       THREE   LD  R0, C
7               ADD R1, R1, R0
8               ST  R1, SUM
9               TRAP  x25
10      A       .FILL x0001
11      B       .FILL x0002
12      C       .FILL x0003
13      D       .FILL x0004
14              .END
```

1. The 1st error is on Line 8: ST R1. SUM. SUM is an undefined label. This error will be detected during assembly time.
2. The 2nd error is on Line 3: ADD R1, R1, R0. R1 was not initialized before it was used; therefore the result of this instruction could be wrong. This error will be detected during run time.

Chapter 8: 8.5, 8.10

8.5 What is the purpose of bit [15] in the KBSR?

Bit[15] in the Keyboard Status Register (KBSR) is the ready bit. The purpose of this bit is to use it as a synchronization mechanism to let the processor know that input from the keyboard has occurred. If KBSR[15] is 0, no key has been struck and the value in the Keyboard Data Register (KBDR) is not valid. If KBSR[15] is 1, the value in KBDR is the ASCII code corresponding to the last key struck.

8.10 What problem could occur if the display hardware does not check the DSR before writing to the DDR?

If DSR[15] is 1, the data contained in the DDR has not been displayed by the monitor. Thus, if the display hardware does not check the DSR before writing to the DDR, the previous value in DDR could be lost.

Chapter 9: 9.2, 9.16

9.2
a. How many trap service routines can be implemented in the LC-3? Why?
b. Why must a RET instruction be used to return from a TRAP routine? Why won't a BR (Unconditional Branch) instruction work instead?
c. How many accesses to memory are made during the processing of a TRAP instruction? Assume the TRAP is already in the IR.

a. The TRAP vector is 8 bits wide; therefore 2^8 = 256 routines. Because the first 8 bits are used for identifying the trap routine, the next 4 bits are zeros (which do nothing), and the last 4 bits are all ones (to identify the TRAP Opcode).
b. RET stores the value of PC (before execution of the service routine) in R7 so that it can return control to the original program after execution of the service routine. A BRnzp would not work because:
    1. The TRAP routine may not be reached by a 9 bit offset.
    2. If TRAP is called multiple times, the computer would not know which LABEL to go to (can change every time)
c. 2 memory accesses are made during TRAP instruction:
    1. 1st access: Instruction in fetch
    2. 2nd access: Trap vector table to get address of TRAP service routine

9.16 The two code sequences *a* and *b* are assembled separately. There is one error that will be caught at assemble time or at link time. Identify and describe why the bug will cause an error, and whether it will be detected at assemble time or link time.

a.

```
        .ORIG x3200
SQRT    ADD     R0, R0, #0
        ; code to perform square
        ; root function and
        ; return the result in R0
        RET
        .END
```

b.

```
        .EXTERNAL SQRT
        .ORIG   x3000
        LD      R0,VALUE
        JSR     SQRT
        ST      R0,DEST
        HALT
VALUE   .FILL   x30000
DEST    .FILL   x0025
        .END
```

The error that will be caught is during assembly time at the VALUE .FILL x30000 instruction because the hex value has 5 numbers (20 bits) instead of 4 (16 bits), which is the max number of bits for the LC-3 instruction set.

Chapter 10: 10.1, 10.8

10.1 What are the defining characteristics of a stack?

The defining characteristics of a stack is the unique specification of how it is to be accessed. Stack follows a LIFO (Last In First OUT) structure. This means that the last node or object that is put in the stack will be the first one to get out from the stack.

10.8 The following operations are performed on a stack:

```
PUSH A, PUSH B, POP, PUSH C, PUSH D, POP, PUSH E,
POP, POP, PUSH F
```

a. What does the stack contain after the `PUSH F`?

b. At which point does the stack contain the most elements? Without removing the elements left on the stack from the previous operations, we perform:

```
PUSH G, PUSH H, PUSH I, PUSH J, POP, PUSH K,
POP, POP, POP, PUSH L, POP, POP, PUSH M
```

c. What does the stack contain now?

OPERATION	STACK
PUSH A:	A
PUSH B:	A B
POP:	A
PUSH C:	A C
PUSH D:	A C D
POP:	A C
PUSH E:	A C E
POP:	A C
POP:	A
PUSH F:	A F

a. The stack contains A and F after the PUSH F operation.

OPERATION	STACK
	A F (From previous operations)
PUSH G:	A F G
PUSH H:	A F G H
PUSH I:	A F G H I
PUSH J:	A F G H I J
POP:	A F G H I
PUSH K:	A F G H I K
POP:	A F G H I
POP:	A F G H
POP:	A F G
PUSH L:	A F G L
POP:	A F G
POP:	A F
PUSH M:	A F M

b. The stack contains the most elements after PUSH J and after PUSH K operations. Both with a number of 6 elements.

c. Now, the Stack contains A F and M, with M being at the top of the Stack.