Andrew Hedy

CSE 313 MW 12pm

Professor Hou

11/21/2016

Homework 3

Chapter 7:

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** | **x3027** |
| **SAVE3** | **x3029** |
| **SAVE4** | **x302A** |

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

**The number stored in the immediate value that is too large to be stored. The last 5 bits of the add function are reserved for the immediate value so 25 = 32. Since LC-3 is 2’s complement this means we can store any number from -2k-1 to 2k-1-1. One possible solution would be to perform two add operations that add the value 15 to R3 and store in R3.**

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.

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

**Line 8: Instruction references undefined label ‘SUM’. Instruction references non-addressable memory location. This label has nowhere to put the values after they are stored from R1 into the label ‘SUM’. This error was caught in assembly time.**

**Lines 3, 5, and 7: If R0 isn’t cleared before loading new values than the add instruction will not be correct. This error will be detected at run time.**

7.18.) The following LC-3 program compares two character strings of the same length. The source strings are in the .STRINGZ form. The first string starts at memory location x4000, and the second string starts at memory location x4100. If the strings are the same, the program terminates with the value 0 in R5. Insert instructions (a), (b), and (c) that will complete this program.  (hint: consider using a NOT instruction in line (b))

.ORIG x3000

LD R1, FIRST

LD R2, SECOND

AND R0, R0, #0

LOOP **LDR R3, R1, #0** (a) ; load value from R1 into R3 and offset #0

LDR R4, R2, #0

BRz NEXT

ADD R1, R1, #1

ADD R2, R2, #1

**NOT R4, R4** (b) ; Not R4 and store back into R4

**ADD R4, R4, #1** (c) ; Add value 1 to R4 and store into R4 (increment)

ADD R3, R3, R4

BRz LOOP

AND R5, R5, #0

BRnzp DONE

NEXT AND R5, R5, #0

AND R5, R5, #1

DONE TRAP x25

FIRST .FILL x4000

SECOND .FILL x4100

.END

Chapter 8:

8.2.) Why is a Ready bit not needed if synchronous I/O is used? **Synchronous I/O does input/output in synchronous intervals so it knows exactly when to take from the keyboard and exactly when to output to the monitor. The “flag” also known as the ready bit is not needed because synchronous knows exactly what to do and when to do it.**

8.5.) What is the purpose of bit [15] in the KBSR? **The KBSR[15] controls the synchronization of the slow keyboard and the fast processor.**

8.11.) Which is more efficient, interrupt-driven I/O or polling? Explain. **In polling the CPU keeps checking status register until new data arrives or device ready for next data. In interrupt-driven I/O the device sends a special signal to CPU when new data arrives or when the device is ready for the next set of data. In this scenario the CPU can perform other tasks instead of constantly checking the registers. Between the two transfer controls the interrupt-driven I/O would be more efficient since it can perform other tasks until new data arrives.**

8.14.) An LC-3 load instruction specifies address xFE02. How do we know whether to load from the KBDR or from memory location xFE02? **Addresses from the range xFE00 to xFE02 are reserved for the KBDR and the KBSR. Looking at table A3 in the book it looks as if xFE02 is assigned to the keyboard data register (KBDR).**