CE 12 Lab Report 3

Writing Assembly and Generating Machine Code

Brandon Gomez

Lab Section: 2

10.29.14

PROCEDURE:
The purpose of this lab is to get familiar writing in machine code and in LC-3 assembly. Our program is storing words into the memory and the program should count the number of 1s in the total number of words. Our program must start at the address x3100 and have the words stored in the address x3200+n. The number of 1s should be stored in x3101. If we were to insert five and two into x3200 and x3201, then the program should count three 1s. In order to do these instructions I will use 11 unique opcodes.

MATERIALS:
Orgin / .ORIG xXXXX / 0110 xXXXX
Tells the program where to start
AND DR, SR1, SR2 / 0101 DR SR1 0 00 SR2
This AND ands SR1 and SR2 and stores the result into DR
AND DR, SR1, imm5 / 0101 DR SR1 1 imm5
This AND ands SR1 and the digit and stores the result into DR
ADD DR, SR1, SR2 / 0001 DR SR1 0 00 SR2
This ADD adds SR1 and SR2 and stores the result into DR
ADD DR, SR1, imm5 / 0001 DR SR1 1 imm5
This ADD adds SR1 and the digit and stores the result into DR
Store Direct / ST SR, LABEL / 0011 SR offset9
Gets whatever data is in SR and stores it into LABEL
Load Direct / LD DR, LABEL / 0010 DR PCoffset9
Loads the data inside of LABEL and puts it in DR
Load Base + Index / LDR DR, BaseR, offset / 0110 DR BaseR offset6
Loads whatever is in BaseR and puts it into DR
Load Effective Address / LEA DrR LABEL / 1110 DR offset9
Loads the address of LABEL and puts it into DR
Conditional Branch / BRx LABEL / 0000 nzp PCoffset9
If the condition is true then it will go to the address LABEL, else it will go to the next line
System Routine / TRAP x25 / 1111 0000 trapvect8
Halts the program

PART A: The registers
We first have to initialize our program at x3100. Then we have to set up our registers: R0 contains the number of words to be counted, R1 contains the count of 1s, R2 contains the shifter to compare each bit with the data in order to check if the bit contains a 1 so we initialize it to 1, R3 will contains the results of a further instruction, R4 will contain the data at the address x3200+n, R4 and R5 will currently store the word to count the bits in, R6 will contain the address we are currently looking at. Before I initialized anything, I made sure to clear each register just incase there was data before I stared using it. In total I will use 6 registers.

```
;; initialize the program at 3100
        ;;.ORIG          x3100
        0011 0001 0000 0000

;; Registers
        ;;AND   R0,R0,#0         ; clear # of words to count
        0101 000 000 1 00000
        ;;ADD   R0,R0,#2         ; initialze to 2
        0001 000 000 1 00010
        ;;AND   R1,R1,#0         ; clear counter to store number of 1s
        0101 001 001 1 00000
        ;;AND   R2,R2,#0         ; clear shifter
        0101 010 010 1 00000
        ;;ADD   R2,R2,#1         ; intialize shifter to 1
        0001 010 010 1 00001
        ;;AND   R3,R3,#0         ; temp
        0101 011 011 1 00000
        ;;AND   R4,R4,#0         ; clear
        0101 100 100 1 00000
        ;;ADD   R4,R4,#2         ; input 3 to be counted
        0001 100 100 1 00011
        ;;AND   R5,R5,#0         ; clear
        0101 101 101 1 00000
        ;;ADD   R5,R5,#7         ; input 7 to be counted
        0001 101 101 1 00111
        ;;AND   R6,R6,#0         ; will store current address
        0101 110 110 1 00000
```

PART B: The outer word loop

Before we get to the outer loop we first have to store our words into x3200+n. Then after that we are going to load x3200 into R6 and then load the data inside address x3200, which is inside R6, into R4. Now inside the loop we first have to clear and initialize the shifter, R2, because we have to reset the shifter for every new word. We then go inside of the counter loop (look at PART C). After the program is done counting the number of 1s inside the word it will then increment the address and load that address data into R4, if the word count, R0, is positive it will go back to the top of the loop, else it will continue with the program where it stores the counter into x3101 and then halt the program.

```
;; store words at x3200+n
        ;;ST    R4,x3200
        0011 100 011110100
        ;;ST    R5,x3201
        0011 101 011110100

        ;;LEA   R6,xF5
        1110 110 011110010      ; load address number x3200 into R6
        ;;LD    R4,xF5
        0010 100 011110001      ; store data x3200 into R4

        ;; LOOP if still words left

        ;;AND   R2,R2,#0         ; clear R2 shifter
        0101 010 010 1 00000
        ;;ADD   R2,R2,#1
        0001 010 010 1 00001    ; intialize shift back to 1 for loop
```

INNER LOOP LOOK AT PART C

```
;;ADD R6,R6,#1          ; increment address x32nn+1
0001 110 110 1 00001
;;LDR R4,R6             ; load data into R4
0110 100 110 000000
;;ADD R0,R0,#-1         ; decrement words left
0001 000 000 1 11111
;;BRp   LOOP            ; if still words left do outer loop again
0000 001 111110101

;;ST R1
0011 001 111100111      ; store counter into x3101
;;.END
;; TRAP x25             ; stop the processor
1111 0000 00100101
```

PART C: The inner counter loop

The compares the shifter and the data and if the shifter sees a 1 it will then increment the counter. If the shifter does not see a one it will then skip the increment, continue, and double the shifter. Here is an example if the word is 0000 0101:

Word: 0000 0000 0000 0101
Shifter: 0000 0000 0000 0001
Word AND Shifter = 0000 0000 0000 0001
Since the result is not zero, it will increment the counter and double the shifter and start the loop again.
Word: 0000 0000 0000 0101
Shifter: 0000 0000 0000 0010
Word AND Shifter = 0000 0000 0000 0000
Since the result is zero, it will skip the increment but still double the shifter.
This will continue until it compares every bit and this is when the shifter is negative because the front bit in the shifter will be a 1. In total, it will take 16 cycles in order to count the bits in a single word.

```
;; counter if still bits left
;; WHILE_STILL_ONES
        ;;AND   R3,R4,R2         ; store result in the temp
        0101 011 100 0 00 010
        ;;BRz   SKIP             ; skip increment count if it is zero
        0000 010 000000001
        ;;ADD   R1,R1,#1         ; increment count
        0001 001 001 1 00001
        ;;SKIP  ADD     R2,R2,R2; double right shifter
        0001 010 010 0 00 010
        ;;BRp   WHILE_STILL_ONES; do inner loop again if positive
        0000 001 111111011
```

RESULTS:

Lets do an example of the program with the numbers 3 and 7.
Orgin = x3100
Clear R0
ADD 2 to R0
Clear R1
Clear R2

ADD 1 to R2
Clear R3
Clear R4
ADD 3 to R4
Clear R5
ADD 7 to R5
Clear R6
Store 3 in the address x3200
Store 7 in the address x3201
Load the address number x3200 into R6
Load the data inside address x3200 into R4
INSIDE OUTER WORD LOOP
    Clear R2
    ADD 1 to R2
    INSIDE INNER COUNTER LOOP
        AND R4 (3) and R2 (1) and store result into R3
        Since not zero increment count so R1 = 1
        Double R2
        Since R2 is positive loop back up
        .
        .
        .
        (R0 = 2)
        (R1 = 1, R2 = 2, R4 = 3, R3 = 1)
        (R1 = 2, R2 = 4, R4 = 3, R3 = 2)
        (R1 = 2, R2 = 8, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 16, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 32 R4 = 3, R3 = 0)
        (R1 = 2, R2 = 64, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 128, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 256, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 512, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 1024, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 2048, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 4096, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 8192, R4 = 3, R3 = 0)
        (R1 = 2, R2 = 16384, R4 = 3, R3 = 0)
        (R1 = 2, R2 = -32768, R4 = 3, R3 = 0)

    END OF INNER COUNTER LOOP
    Increment R6 (x3200 -> 3201)
    Load data of x3201 into R4
    Decrement word count R0 (2 -> 1)
    Since R0 is not zero go back to start of outer loop

    Clear R2

ADD 1 to R2
INSIDE INNER LOOP
(R0 = 1)
(R1 = 3, R2 = 2, R4 = 7, R3 = 1)
(R1 = 4, R2 = 4, R4 = 7, R3 = 2)
(R1 = 5, R2 = 8, R4 = 7, R3 = 4)
(R1 = 5, R2 = 16, R4 = 7, R3 = 0)
(R1 = 5, R2 = 32 R4 = 7, R3 = 0)
(R1 = 5, R2 = 64, R4 = 7, R3 = 0)
(R1 = 5, R2 = 128, R4 = 7, R3 = 0)
(R1 = 5, R2 = 256, R4 = 7, R3 = 0)
(R1 = 5, R2 = 512, R4 = 7, R3 = 0)
(R1 = 5, R2 = 1024, R4 = 7, R3 = 0)
(R1 = 5, R2 = 2048, R4 = 7, R3 = 0)
(R1 = 5, R2 = 4096, R4 = 7, R3 = 0)
(R1 = 5, R2 = 8192, R4 = 7, R3 = 0)
(R1 = 5, R2 = 16384, R4 = 7, R3 = 0)
(R1 = 5, R2 = -32768, R4 = 7, R3 = 0)
END OF INNER COUNTER LOOP
Increment R6 (x3201 -> 3202)
Load data of x3202 into R4
Decrement word count R0 (1 -> 0)
Since R0 is zero we break out of outer loop

END OF OUTER LOOP
Since R1 contains the count, we need to store R1 into x3101
HALT

CONCLUSION:
In conclusion, I found it easier to first draft the entire program on a piece of paper and then write each one of the instructions in assembly language and then convert it into machine code. I found that if you were to write the offset for the branch right away, their will get a great chance that you will have to change the offset if you add more code. The types of branches I used were BRz and BRp. I calculated the offsets by counting the number of lines in between the branch and the destination. If the branch goes back up the program then I will have to use a negative binary number and if destination is lower in the program then I will use a positive binary number to achieve this. If you need more than 8 registers you can store the data in some address and then clear the registers and use them again. In order to make my program more efficent is to use call and return functions. My program is 29 lines and 1990 bytes.

By stepping through the hello.bin program I was able to see what PUTS trap does. First, Load what is in the data of register R0 into R1, then checks to see R0 is zero, if it is skips the whole process and goes to the next instruction. If R0 is not zero then it will load indirect to R2 and check to see if is negative. It will then store indirect to R1 and increment R0 that contains the address. If there is still letters, it will repeat this process.

```
x0454  0110001000000000  x6200          LDR    R1, R0, #0
x0455  0000010000000101  x0405          BRZ    x045B
x0456  1010010000001001  xA409          LDI    R2, x0460
x0457  0000011111111110  x07FE          BRZP   x0456
x0458  1011001000001000  xB208          STI    R1, x0461
x0459  0001000000100001  x1021          ADD    R0, R0, #1
x045A  0000111111111001  x0FF9          BRNZP  x0454
```

By stepping through the hello.bin program, I was able to see what the HALT trap does. First, It stores register data into the address and then it prints the halting statement. It then loads data into the registers and retires the forever loop.

```
xFD70  0011111000001110  x3E0E          ST     R7, xFD7F
xFD71  0011001000001100  x320C          ST     R1, xFD7E
xFD72  0011000000001010  x300A          ST     R0, xFD7D
xFD73  1110000000001100  xE00C          LEA    R0, xFD80
xFD74  1111000000100010  xF022          TRAP   PUTS
xFD75  1010001000101111  xA22F          LDI    R1, xFDA5
xFD76  0010000000101111  x202F          LD     R0, xFDA6
xFD77  0101000001000000  x5040          AND    R0, R1, R0
xFD78  1011000000101100  xB02C          STI    R0, xFDA5
xFD79  0010000000000011  x2003          LD     R0, xFD7D
xFD7A  0010001000000011  x2203          LD     R1, xFD7E
xFD7B  0010111000000011  x2E03          LD     R7, xFD7F
xFD7C  1100000111000000  xC1C0          RET
```