CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING

UNIVERSITY OF WISCONSIN—MADISON

Prof. Mark D. Hill

TAs: Sujith Surendran, Pradip Vallathol

Midterm Examination 4

In Class (50 minutes)

Wednesday, Dec 11, 2013

Weight: 17.5%

NO: BOOK(S), NOTE(S), OR CALCULATORS OF ANY SORT.

The exam has 10 pages. **Circle your final answers**. Plan your time carefully since some problems are longer than others. **You must turn in the pages 1-8**. Use the blank sides of the exam for scratch work.

Note: LC-3 instruction set is provided on Page 9. Trap Codes and Assembler Directives are provided on the last page.

LAST NAME: ____

FIRST NAME: ____

ID# ____

Problem	**Maximum Points**	**Points Earned**
1	4	
2	8	
3	8	
4	4	
5	6	
Total	30	

Problem 1: Assembly Errors **(4 Points)**

Identify 4 errors in the following Assembly language program:

```
          .ORIG x3601

          LD R0, VALUE
NEXT      TRAP x20
          TRAP x21
          AND R0, R0, R0
          BRzp NEXT
          TRAP x20
          NOT R2, #1                ---- (a)
          ADD R0, R0, FIVE          ---- (b)
          LD R5, R2, #3
          BRp NEXT

NEXT      HALT                      ---- (c)
FIVE      .FILL     x5
VALUE     .ASCII    #9              ---- (d)
          .END
```

(a) NOT cannot have an immediate operand

(b) ADD cannot have a memory argument

(c) Double declaration of NEXT

(d) .ASCII assembler directive does not exist in LC-3.

Problem 2: Two Pass Assembly Process **(8 Points)**

(a) **(2 Points)** Consider the following assembly code. What will be the output on the console if you run this code on PennSim?

```
              .ORIG x3500
              LEA R3, INPUT2
              LD  R1, SIZE
              ADD R3, R3, R1

       NEXT   LDR R0, R3, 0
              TRAP x21
              ADD R3, R3, -1
              ADD R1, R1, -1
              BRp NEXT

              HALT

       INPUT2  .STRINGZ  "QweR"
       STRING2 .BLKW   4
       SIZE2   .FILL   3
              .END
```

Answer: Rew

(b) (4 Points) In the first pass, the assembler creates the symbol table. Fill in the symbol table created by the assembler for the program.

Label	**Address**
NEXT	**3503**
INPUT2	**3509**
STRING2	**350E**
SIZE2	**3512**

(c) **(2 Points)** In the second pass, the assembler creates a binary (.obj) version of the program, using the entries from the symbol table. Assume that there exists another program at 0x3000, whose assembly instructions are as shown below. If the following symbol table entries were generated in the first pass of the assembly for this program, write the binary code generated by the assembler for the two instructions at 0x3000 and 0x3001.

Symbol Table:

Label	**Address**
`INPUT`	`x3025`
`NEXT`	`x3011`

Generated Binary code:

Address	**Instruction**	**Binary Code**
`x3000`	`LD R0, INPUT`	0010 0000 0010 0100
`x3001`	`BRp NEXT`	0000 0010 0000 1111

1) PC'+offset = x3025
   x3000 + 1 + offset = x3025
   ⇨ Offset = x24

2) PC'+offset = x3011
   x3001 + 1 + offset = x3011
   ⇨ Offset = xF

Problem 3: Subroutines and Traps **(8 Points)**

Suppose we want to write a new TRAP subroutine, TRAP x55, which takes a string input from the user. The trap subroutine starts from address x3300 and does the following:

1) It takes a character input from the user
2) It then displays this character (which the user inputs) on the console
3) After that, it stores the user input characters in consecutive memory locations starting from the address location present in register R4. It then repeats (1), (2) and (3) until user inputs ‘Z’.
4) It uses a “callee-save” strategy and ensures that none of the register values are modified by it.
5) It uses R5 to store the ASCII value corresponding to -Z

(a) **(6 Points)** Fill in the missing parts of the trap subroutine.

```
    .ORIG x3300
        ST R0, SAVEREG1
        ST R4, SAVEREG2
        ST R2, SAVEREG3
        ST R7, SAVEREG4
        LD  R2, NEGZ

NEXT    TRAP x20
        TRAP x21
        STR R0, R4, #0
        ADD R4, R4, #1
        ADD R0, R0, R5
        BRnp NEXT

        LD R0, SAVEREG1
        LD R4, SAVEREG2
        LD R5, SAVEREG3
        LD R7, SAVEREG4
        RET

;Data Region
NEGZ        .FILL xFFA6      ;xFFA6 = FFFF – ASCII value of Z
SAVEREG1    .BLKW 1
SAVEREG2    .BLKW 1
SAVEREG3    .BLKW 1
SAVEREG4    .BLKW 1
```

(b) **(2 Points)** Given the following Trap vector table entry:

Address	**Content**
x45	x27

Give the assembly instruction that you would use to call the TRAP routine corresponding to this entry. Provide reasons to justify your answer.

Ans: TRAP x45 because the starting location of trap routine is stored in x45.

Problem 4: I/O **(4 Points)**

(a) **(4 Points)** The following code segment should display the string specified at the "INPUT" label on to the console. Write the missing assembly instructions of the program (without using PUTS/PUTC/TRAP instructions).

- **Note:** The instructions which are missing should jump to halt if it is the end of the string. Else, it should print the character onto the console.

```
       .ORIG   x3000

       LEA R5, INPUT

LOOP   LDR R0, R5, #0
       BRz END
POLL   LDI R1, DSR
       BRzp POLL
       STI R0, DDR

       ADD R5, R5, #1     ; Point to the next character
       BR  LOOP

END    HALT


INPUT  .STRINGZ   "Best of Luck!"  ; String to display
DSR    .FILL   xFE04   ; Display status register location
DDR    .FILL   xFE06   ; Display data register location
       .END
```

Problem 5: Short Answer Questions **(6 Points)**

(a) **(2 Points)** An LC-3 assembly program contains the following instruction:

```
INTEGER   LD R5, INTEGER
```

The symbol table entry for INTEGER is x3000. What will be the value of R5 after the execution of the above instruction?

Ans : x2BFF (the binary code for this instruction).

(b) **(2 Points)** Briefly state what happens in Linking and Loading phases for an assembly program?

During the Linking phase, the symbols between different object files which are linked together get resolved.
During the loading phase, the executable image is copied onto the memory.

(c) **(1 Point)** Briefly state a scenario where you would prefer interrupt-driven I/O over polling based I/O?

If the I/O device takes a lot of time to execute the command, then polling consumes a lot of cycles. In these cases, interrupt-driven I/O is preferred.

(d) **(1 Point)** Suppose two I/O devices sends interrupts to the CPU at the same time. How does the CPU decide which interrupt to service first?

The one with a higher priority (ie, at a higher priority level) is executed first.

LC-3 Instruction Set (Entered by Mark D. Hill on 03/14/2007; last update 03/15/2007)

```
PC': incremented PC. setcc(): set condition codes N, Z, and P. mem[A]:memory contents at address A.
SEXT(immediate): sign-extend immediate to 16 bits. ZEXT(immediate): zero-extend immediate to 16 bits.
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ADD DR, SR1, SR2 ; Addition
| 0   0   0   1 |    DR     |    SR1    | 0 | 0   0 |    SR2    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 + SR2 also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ADD DR, SR1, imm5 ; Addition with Immediate
| 0   0   0   1 |    DR     |    SR1    | 1 |       imm5        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 + SEXT(imm5) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ AND  DR, SR1, SR2 ; Bit-wise AND
| 0   1   0   1 |    DR     |    SR1    | 0 | 0   0 |    SR2    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 AND SR2 also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ AND DR,SR1,imm5 ; Bit-wise AND with Immediate
| 0   1   0   1 |    DR     |    SR1    | 1 |       imm5        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 AND SEXT(imm5) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ BRx,label (where x={n,z,p,zp,np,nz,nzp}); Branch
| 0   0   0   0 | n | z | p |             PCoffset9             | GO ← ((n and N) OR (z AND Z) OR (p AND P))
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ if(GO is true) then PC←PC'+ SEXT(PCoffset9)

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JMP BaseR ; Jump
| 1   1   0   0 | 0   0   0 |   BaseR   | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ PC ← BaseR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JSR label ; Jump to Subroutine
| 0   1   0   0 | 1 |                 PCoffset11                |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ R7 ← PC', PC ← PC' + SEXT(PCoffset11)

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JSRR BaseR ; Jump to Subroutine in Register
| 0   1   0   0 | 0 | 0   0 |   BaseR   | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ temp ← PC', PC ← BaseR, R7 ← temp

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LD DR, label ; Load PC-Relative
| 0   0   1   0 |    DR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← mem[PC' + SEXT(PCoffset9)] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LDI DR, label ; Load Indirect
| 1   0   1   0 |    DR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR←mem[mem[PC'+SEXT(PCoffset9)]] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LDR DR, BaseR, offset6 ; Load Base+Offset
| 0   1   1   0 |    DR     |   BaseR   |        offset6        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← mem[BaseR + SEXT(offset6)] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LEA, DR, label ; Load Effective Address
| 1   1   1   0 |    DR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← PC' + SEXT(PCoffset9) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ NOT DR, SR ; Bit-wise Complement
| 1   0   0   1 |    DR     |    SR     | 1 | 1   1   1   1   1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← NOT(SR) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ RET ; Return from Subroutine
| 1   1   0   0 | 0   0   0 | 1   1   1 | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ PC ← R7

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ RTI ; Return from Interrupt
| 1   0   0   0 | 0   0   0   0   0   0   0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ See textbook (2nd Ed. page 537).

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ST SR, label ; Store PC-Relative
| 0   0   1   1 |    SR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[PC' + SEXT(PCoffset9)] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ STI, SR, label ; Store Indirect
| 1   0   1   1 |    SR     |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[mem[PC' + SEXT(PCoffset9)]] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ STR SR, BaseR, offset6 ; Store Base+Offset
| 0   1   1   1 |    SR     |   BaseR   |        offset6        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[BaseR + SEXT(offset6)] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ TRAP ; System Call
| 1   1   1   1 | 0   0   0   0 |           trapvect8           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ R7 ← PC', PC ← mem[ZEXT(trapvect8)]

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ; Unused Opcode
| 1   1   0   1 |                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ Initiate illegal opcode exception
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
```

TRAP CODES

Code	*Equivalent*	*Description*
`HALT`	`TRAP x25`	Halt execution and print message to console.
`IN`	`TRAP x23`	Print prompt on console, read (and echo) one character from keybd. Character stored in R0[7:0].
`OUT`	`TRAP x21`	Write one character (in R0[7:0]) to console.
`GETC`	`TRAP x20`	Read one character from keyboard. Character stored in R0[7:0].
`PUTS`	`TRAP x22`	Write null-terminated string to console. Address of string is in R0.

ASSEMBLER DIRECTIVES

Opcode	*Operand*	*Meaning*
`.ORIG`	address	starting address of program
`.END`		end of program
`.BLKW`	n	allocate n words of storage
`.FILL`	n	allocate one word, initialize with value n
`.STRINGZ`	n-character string	allocate n+1 locations, initialize w/characters and null terminator