CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING

UNIVERSITY OF WISCONSIN—MADISON

Prof. Gurindar Sohi, Kai Zhao
TAs: Mohit Verma, Annie Lin, Neha Mittal, Daniel Griffin, Yuzhe Ma

Examination 4
In Class (50 minutes)
Monday, December 12, 2016
Weight: 17.5%

NO: BOOK(S), NOTE(S), CALCULATORS OR ELECTRONIC DEVICES OF ANY SORT.
The exam has **ten pages**. You **must turn in the pages 1-8. Circle your final answers**. Plan your time carefully since some problems are longer than others. Use the blank sides of the exam for scratch work. Feel free to rip out the last two pages for reference.

LAST NAME: _____
FIRST NAME: _____
Section: _____
ID#: _____

Problem	Maximum Points	Points Earned
1	6	
2	7	
3	4	
4	3	
5	3	
6	5	
7	4	
Total	32	

Problem 1 (6 points)

An LC-3 assembly language program is given below:

```
            .ORIG x3000
            LD  R2, NUMBER
            LD  R1, MASK
            LD  R3, PTR2
LOOP        LDR R4, R3, #0
            AND R4, R4, R1
            BRz NEXT
            ADD R0, R0, #1
NEXT        ADD R3, R3, #1
            ADD R2, R2, #-1
            BRp LOOP
            STI R0, PTR1
            HALT
NUMBER      .BLKW 3
MASK        .FILL x8000
PTR1        .FILL x4000
PTR2        .FILL x5000
            .END
```

Symbol	Address (in hex)
LOOP	x3003

(a) A symbol table is created during the first pass of the assembler. Fill in the symbol table above for the preceding program. You may not need to use all rows.

(b) In the second pass, the assembler creates a binary version (.obj) of the program, using the entries from the symbol table shown below. Given that the following symbol table entries were generated in the first pass of assembly (for a different program than the one in part(a)), fill in the binary code generated by the assembler for the two instructions located at x3000 and x300A.

Symbol	Address
ADDRESS	x3012
AGAIN	x3014
PTR3	x3015
DESTINATION	x301A

Address	Instruction (in assembly)	Instruction (in binary)
x3000	`ADD R1, R2, #4`	
x300A	`STI R0, PTR3`	

Problem 2 (7 points)

```
            .ORIG x3000       ; The program begins at x3000
            AND R5, R5, #0    ; R5 ← 0, stores final answer    x3000
            ADD R5, R5, #1    ; R5 ← 1, initialization         x3001
            AND R7, R7, #0    ; R7 ← 0, counter                x3002
CONTINUE    ADD R7, R7, #-1   ; R7 = R7 - 1                    x3003
            ADD R6, R0, R7    ; R6 = R0 + R7                   x3004
            BRn END           ; halt if result negative        x3005
CALL_FUNC   JSR Mult_by_4     ; Subroutine call                x3006
            BRnzp CONTINUE    ;                                x3007
END         HALT              ;                                x3008
Mult_by_4   ADD R6, R5, R5    ; Subroutine to multiply by 4    x3009
            ADD R5, R6, R6    ;                                x300A
            RET               ;                                x300B
SAVE_VAL    .BLKW #1          ; Save data here                 x300C
```

The above assembly program calculates the value of 4^n, where n is the value in register R0, and stores the result in register R5. The code lines have been numbered, as shown above. **Assume R0 = 3, and all other registers (R1-R7) are 0 before the execution begins**. The final value in R5 after the program finishes execution should be $4^3 = 64$.

(a) Write the value (in hex) in register R7 just before the subroutine 'Mult_by_4' is called **for the 1st time.**

(b) Write the value (in hex) in register R7 **just before** the subroutine 'Mult_by_4' **returns.**

(c) The above program does not terminate. Explain why.

(d) Fill in the code provided below to fix the problem mentioned in part 3. The below code REPLACES instructions at x3002-x3004 in the provided code. Explain your solution. **Rest of the code remains unchanged.**

```
            AND ____, ____, #0      ;              x3002
CONTINUE    ADD ____, ____, #-1     ;              x3003
            ADD  R6 , ____, ____    ;              x3004
```

(e) Fill in the code provided below to fix the problem mentioned in part 3 using SAVE_VAL label. The below code REPLACES instructions at x3006-x3007 in the provided code, and adds two more instructions, as shown below. **Rest of the code remains unchanged**.

```
CALL_FUNC ____________________ , SAVE_VAL  ; Save something
          JSR Mult_by_4                    ; Subroutine call
          ____________________ , SAVE_VAL  ; Load something
          BRnzp CONTINUE
```

Problem 3 (4 points)

(a) How would you implement a subroutine using caller-save?

(b) What is the difference between asynchronous and synchronous I/O?

(c) In **interrupt**-driven I/O, if a program is running at PL3 and the I/O device at PL1, can an interrupt successfully occur? Explain why or why not.

(d) How are KeyBoard Status Register (KBSR) and KeyBoard Data Register (KBDR) used when TRAP x20 (GETC) is called?

Problem 4 (3 points)

Find 3 syntax errors in the following assembly language code.

```
          .ORIG x3000
          AND R1, R1, #12
          NOT R3, R1, R5
          ADD R2, R3, #-1
          BRz END
          BRnzp DONE
END       STI R9, ADD2
          HALT
ADD1      .FILL x4000
          .FILL x4600
ADD2      .FILL x5000
          .END
```

Problem 5 (3 points)

Consider the following program.

```
          .ORIG x3000
          LD   R1, NUM
          LD   R2, INCRE
INPUT     GETC
          ADD  R0, R0, R2
          OUT
          ADD  R1, R1, #-1
          BRz  STOP
          BRnzp INPUT
STOP      HALT
INCRE     .FILL x5
NUM       .FILL x10
          .END
```

(a) Fill in the following TRAP instruction that corresponds to symbol 'INPUT':

1	**1**	**1**	**1**	**0**	**0**	**0**	**0**								

(b) Shown below is part of the trap vector table (or System Control Block). Specify the PC value after the trap instruction

Memory Address	**Content**
x0020	x0400
x0021	x0430
x0022	x0450
x0023	x04A0

Problem 6 (5 points)

(i) What is the purpose of bit [15] of KBSR?

(a) It is set to 1 when the keyboard receives a new character.
(b) It is set to 1 when the device is ready to display a new character.
(c) It is set to 0 when TRAP x25 is called to halt program execution.
(d) It is set to 0 when there is input data stored at R7.

(ii) Assume that a LC-3 processor receives interrupts from 3 I/O devices (A, B and C) simultaneously. The priority levels for the interrupts are given below:

A: PL2 B: PL0 C: PL6

Which of the above interrupts is serviced first?

(a) B
(b) A
(c) C
(d) Any selected at random

(iii) Which of the following conditions must be satisfied for an I/O device to be able to successfully interrupt a processor? Circle the correct option.

A: The I/O device must be able to request service.
B: The processor must be able to poll the I/O device.
C: The priority of the I/O device request must be higher than the current executing process on the processor.

(a) A and C
(b) B and C
(c) A, B and C
(d) Only B

(iv) The LC-3 Trap Mechanism performs 3 operations. Possible operation sequences are given below. Circle the correct sequence of operations.

A: Return (JMP R7)
B: Lookup service routine starting address
C: Check the control registers
D: Transfer to service routine

(a) A, C, D
(b) C, B, D
(c) B, D, A
(d) D, A, B

(v) How many trap service routines can be defined in LC-3?

(a) 128
(b) 64
(c) 356
(d) 256

Problem 7 (4 points)

Consider the following program for converting a string of uppercase letters (A~Z) to the lowercase. The string is input from the keyboard one character by one character with an end of '#' (ASCII x23). The result is stored in memory location starting at x5000. Fill in the missing part of the program according to the comments (some may not be given).

```
                .ORIG x3000
                LD    R1, ENDC          ;  Load '#' into R1
                LD    R4, addition
                LD    R5, addr
Next            (1) _________________   ;  Get the next character
                NOT   R2, R1
                ADD   R2, R2, x1
                (2) _________________   ;  Test if is '#'
                BRz   LAST
                (3) _________________   ;  Convert to lowercase and store the result into R3
                STR   R3, R5, #0
                (4) _________________
                BRnzp Next
LAST            HALT
ENDC            .FILL x23
addr            .FILL x5000
addition        .FILL x20
                .END
```

(1) _____

(2) _____

(3) _____

(4) _____

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
```

Assembler Directives

Pseudo-operations

- do not refer to operations executed by program
- used by assembler
- look like instruction, but "opcode" starts with dot

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

7-8

Trap Codes

LC-3 assembler provides "pseudo-instructions" for each trap code, so you don't have to remember them.

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.

7-9