Homework 6 CS/ECE 252: Sec 1 & 2 [Due 11:59AM on Mon, Apr 7]

Primary contact for this hw: Lisa Ossian [ossianli at cs dot wisc dot edu]

Important Notes:

This homework must be submitted electronically to the Learn@UW dropbox. The files to be submitted include **binary code as binary files (*.bin), pseudo-code in text files (*.txt)**, and **README.txt** (see the submission guidelines below). **Do not submit files in hex or assembly! Only machine language for LC-3 is accepted for submission.**

Your programs should **always** start at address **x3000** and end with a HALT instruction (0xF025).

README file:

Download the file: README.txt.

Replace Lastname, UWID, and Section#. Replace ADDRESS with the halt address for the corresponding problem (HALTP1 = halt address for problem 1, HALTP2 = halt address for problem 2, etc.).

Submission Guidelines:

1. Please submit only one compressed or archive file (*.zip or *.tar.gz) to the folder **homework6**.
2. Name the archive file with the following format: Lastname with .zip or .tar.gz as suffix where Lastname is your last name with 1st letter capitalized.
3. Your archive file should contain the following (the files MUST be named exactly like this):
A. hw6_p1.txt – Pseudo-code for problem 1
B. hw6_p1.bin – Binary code for problem 1
C. hw6_p2.txt – Your answers for problem 2
D. hw6_p3.txt – Pseudo-code for problem 3
E. hw6_p3.bin – Binary code for problem 3
F. hw6_p4.txt – Pseudo-code for problem 4
G. hw6_p4.bin – Binary code for problem 4
H. README.txt - Readme file that contains your name, student ID, and section number and the HALT addresses for problems 1, 3, and 4 (one HALT address for each problem).
4. You can submit your code and other files as many times as you would like until the due time on the due date indicated above.

Problem 1 (6 points)

Write a short LC-3 program in PennSim that compares the 2 numbers in R1 and R2, and then puts the smaller number in R0. If R1 is equal to R2, store 50 in R0. Finally, store the result to memory location 0x5000.

a) (2 points) Write the pseudo code for the algorithm. Please submit your pseudo code in a file exactly named as “hw6_p1.txt”, without the double quotes, to dropbox.

Clear R0 (Initialize it to 0)
Get the value of -R2
Store the value of –R2 into R3
Add R1 and R3, store result into R4
If R4 == 0, R0 = 50
If R4 > 0, this means that R2 < R1, so R0 = R2
If R4 < 0, this means that R1 < R2, so R0 = R1
Store the value of R0 to memory location x5000

b) (4 points) Write an LC-3 program based on pseudo-code from part a. Comment each line of the source code and submit the binary code to dropbox. The file name should be exactly “hw6_p1.bin”, without the double quotes.

Binary:

0011000000000000 ; Starting address x3000.
0101000000100000 ; Initialize R0 to 0
1001011010111111 ; R3 contains the 1's complement value of R2
0001011011100001 ; R3 contains the 2's complement value of R2
0001100001000011 ; R4 is the result of R1 - R2
0000010000000010 ; Jump to EQUAL so R0 is updated to 10
0000001000000011 ; Jump to GREATER so R0 is updated to R2
0000100000000100 ; Jump to LESS so R0 is updated to R1
0010000000001000 ; R0 = 50 if R1 == R2
0000001000000011 ; After update, the program should proceed to store result
0001000000000010 ; R0 = R2 if R2 < R1
0000001000000001 ; After update, the program should proceed to store result
0001000000000001 ; R0 = R1 if R1 < R2
0010110000000010 ; Load R6 with address x5000
0111000110000000 ; Store the result into memory location x5000
1111000000100101 ; HALT
0101000000000000 ; Address where the result will be stored.
0000000000110010 ; Data to store when values are equal

Assembly code:

(Note: You were not expect to turn this in for this assignment. This is for future reference.)

```
            .ORIG x3000         ; Starting address of the program
CLEAR_R0    AND R0, R0, #0      ; Initialize R0 to 0
            NOT R3, R2          ; R3 contains the 1's complement value of R2
            ADD R3, R3, #1      ; R3 contains the 2's complement value of R2
            ADD R4, R1, R3      ; R4 is the result of R1 - R2
            BRz EQUAL           ; Jump to EQUAL so R0 is updated to 10
            BRp GREATER         ; Jump to GREATER so R0 is updated to R2
            BRn LESS            ; Jump to LESS so R0 is updated to R1
EQUAL       LD R0, EQDATA       ; R0 = 50 if R1 == R2
            BRp STOP            ; After update, the program should proceed to store result
GREATER     ADD R0, R0, R2      ; R0 = R2 if R2 < R1
            BRp STOP            ; After update, the program should proceed to store result
LESS        ADD R0, R0, R1       ; R0 = R1 if R1 < R2
STOP        LD R6, OFFSET       ; Load R6 with address x5000
            STR R0, R6, #0      ; Store the result into memory location x5000
            HALT
OFFSET      .FILL x5000         ; Address where the result will be stored.
EQDATA      .FILL 50            ; Data to store when values are equal
            .END
```

Problem 2 (6 points)

Load the below LC-3 program in PennSim, and answer the following questions:

Address	Memory Content	Comment
x3000	0010 0100 0000 1010	R2 = 0
x3001	0010 0000 0000 1010	R0 = 7
x3002	0010 0010 0000 1010	R1 = 8
x3003	0000 0100 0000 0011	BRz x3007
x3004	0001 0100 0000 0010	R2 = R2 + R0
x3005	0001 0010 0111 1111	R1 = R1 - 1
x3006	0000 1111 1111 1100	BR x3003
x3007	0010 0110 0000 0010	Load x4000 into R3

x3008	0111 0100 0111 0000	Store 56 at x4000
x3009	1111 0000 0010 0101	HALT
x300A	0100 0000 0000 0000	Data value x4000
x300B	0000 0000 0000 0000	Data value 0
x300C	0000 0000 0000 0111	Data value 7
x300D`	0000 0000 0000 1000	Data value 8

a) (3 points) Fill in the comments column with a summary of what each instruction does.

b) (1 point) How many times does the instruction at address x3003 execute?
9
c) (1 point) What number does this program write to memory and in which location?
56 at location x4000
d) (1 point) What is the purpose of this program? Describe its purpose in one sentence.
This program multiples 7 by 8 and stores the result at memory location x4000.

Problem 3 (9 points)

Write an LC-3 program that multiplies the number in R1 by the number in R2 and stores the result at memory location 0x5000. Furthermore, store a 1 at memory location 0x5001 if the product is odd or a 0 if it is even. (You can assume that the numbers in R1 and R2 will be positive.)

(3 points) Write the pseudo code for the algorithm. Please submit your pseudo code in a file exactly named as “hw6_p3.txt”, without the double quotes, to dropbox.

Clear R3.
Clear R4.
Copy value from R1 to R3.
Break to multiplication loop.

Loop: Add value of R2 for each iteration to R4, which will eventually be the product.
Decrement R3.
Exit the loop once R3 is nonpositive.

Load the address where the product will be stored.

Store the product at the address in R7.
Load hex value x0001, which will be used to check if a number is odd or even.
And R4 and R1 to determine if R4 is odd or even.
Load the address where output is to be stored.
This will store a 1 if product is odd and 0 otherwise.
Exit the program.
Hex number used to check whether product is odd or even.
Address where product is stored.
Address where bit indicating whether output is odd or even is stored.

(6 points) Write an LC-3 program in PennSim based on pseudo-code from part a. Comment each line of the source code and submit the binary code to dropbox. The file name should be exactly “hw6_p3.bin”, without the double quotes.

Binary:

0011000000000000 ; Starting address of the program
0101011011100000 ; Clear R3.
0101100100100000 ; Clear R4.
0001011011000001 ; Copy value from R1 to R3.
0000111000000000 ; Break to multiplication loop.
0001100100000010 ; Add value of R2 for each iteration to R4, which will eventually be the product.
0001011011111111 ; Decrement R3.
0000001111111101 ; Exit the loop once R3 is nonpositive.
0010111000000111 ; Load the address where the product will be stored.
0111100111000000 ; Store the product at the address in R7.
0010001000000100 ; Load hex value x0001, which will be used to check if a number is odd or even.
0101010100000001 ; And R4 and R1 to determine if R4 is odd or even.
0010011000000100 ; Load the address where output is to be stored.
0111010011000000 ; This will store a 1 if product is odd and 0 otherwise.
1111000000100101 ; Exit the program.
0000000000000001 ; Hex number used to check whether product is odd or even.
0101000000000000 ; Address where product is stored.
0101000000000001 ; Address where bit indicating whether output is odd or even is stored.

Assembly code:
(Note: You were not expect to turn this in for this assignment. This is for future reference.)

```
.ORIG x3000          ; Starting address of the program
AND R3, R3, #0       ; Clear R3.
```

```
                AND R4, R4, #0      ; Clear R4.
                ADD R3, R3, R1      ; Copy value from R1 to R3.
                BR LOOP             ; Break to multiplication loop.

LOOP            ADD R4, R4, R2      ; Add value of R2 for each iteration to R4, which will
                                    ; eventually be the product.
                ADD R3, R3, #-1     ; Decrement R3.
                BRp LOOP            ; Exit the loop once R3 is nonpositive.

STORE           LD R7, PRODUCT      ; Load the address where the product will be stored.
                STR R4, R7, #0      ; Store the product at the address in R7.

CHECK_LAST
                LD R1, CHECK_LAST_BIT  ; Load hex value x0001, which will be
                                       ; used to check if a number is odd or
                                       ; even.
                AND R2, R4, R1         ; And R4 and R1 to determine if R4 is
                                       ; odd or even.
                LD R3, ODD_OR_EVEN     ; Load the address where output is to be
                                       ; stored.
                STR R2, R3, #0         ; This will store a 1 if product is odd and
                                       ; 0 otherwise.
                HALT                   ; Exit the program.

CHECK_LAST_BIT  .FILL x0001    ; Hex number used to check whether product is
                               ; odd or even.
PRODUCT         .FILL x5000    ; Address where product is stored.
ODD_OR_EVEN     .FILL x5001    ; Address where bit indicating whether output is odd
                               ; or even is stored.

                .END
```

Problem 4 (9 points)

Write an LC-3 program that determines whether the number stored at 0x4000 is divisible by n. Use the value stored at memory location 0x4001 for input n. If the number at 0x4000 is divisible by n, store the number at 0x4000 to 0x5000. If not, store the next smallest number that is divisible by n. (You can assume that both numbers given at 0x4000 and 0x4001 will be positive.)

(3 points) Write the pseudo code for the algorithm. Please submit your pseudo code in a file exactly named as “hw6_p3.txt”, without the double quotes, to dropbox.

Load the address of the dividend into R4

Load the address of the divisor into R5
Load the dividend to R0
Load the divisor to R1 which will be complemented

Get 2's complement value of divisor and place it into R1

Clear R2, it will contain the value of quotient finally

Loop: Increment R2 for quotient, until the loop breaks
 In each iteration, R0 <--- R0 + R1
 Forward check if the intermediate result is already less than the divisor (R3 = R0 + R1)
 Loop breaks if R3 is negative, otherwise continue the iterations

Add 0 to the remainder, so we can break on it.
Break to remain_is_zero code if there is no remainder.
Break to remain_is_pos code if there is a remainder.
If remainder is zero {

Load the address where output will be stored in R6.
Reload the address of the dividend into R7.
Load the dividend to R7.
Store the dividend at the address in R6.
Break to stop to exit the program.
}
Else if remainder is positive {

Reload the address of the divisor into R6.
Load the divisor to R6.
Clear R4 for multiplication.
Set R4 to the quotient of division for multiplication.
Clear R5 for multiplication.

Loop: During each increment add the divisor to a sum at R5.
Decrement the quotient at R4.
Keep going until the quotient is negative.

Load the address where output will be stored in R6.
Store the value of quotient*divisor at the address in R6.
}

Exit the program.

(6 points) Write an LC-3 program in PennSim based on pseudo-code from part a. Comment

each line of the source code and submit the binary code to dropbox. The file name should be exactly “hw6_p3.bin”, without the double quotes.

Binary:

```
0011000000000000 ; Starting address of the program
0010100000011101 ; Load the address of the dividend into R4
0010101000011101 ; Load the address of the divisor into R5
0110000100000000 ; Load the dividend to R0
0110001101000000 ; Load the divisor to R1 which will be complemented
1001001001111111 ; Get 2's complement value for the divisor
0001001001100001 ; therefore R1 has the value of -divisor
0101010010100000 ; Clear R2, it will contain the value of quotient finally
0001010010100001 ; Increment R2 for quotient, until the loop breaks
0001000000000001 ; In each iteration, get the result from the subtraction
0001011000000001 ; Forward check if the intermediate result is already less than the divisor.
0000011111111100 ; If R3 is non-negative, go 3 steps back
0001000000100000 ; Add 0 to the remainder, so we can break on it.
0000010000000001 ; If there is no remainder, output the dividend.
0000001000000101 ; If there is a remainder, output quotient*divisor.
0010110000010001 ; Load the address where output will be stored in R6.
0010111000001110 ; Reload the address of the dividend into R7.
0110111111000000 ; Load the dividend to R7.
0111111110000000 ; Store the dividend at the address in R6.
0000111000001010 ; Break to stop to exit the program.
0010110000001011 ; Reload the address of the divisor into R6.
0110110110000000 ; Load the divisor to R6.
0101100100100000 ; Clear R4 for multiplication.
0001100100000010 ; Set R4 to the quotient of division for multiplication.
0101101101100000 ; Clear R5 for multiplication.
0001101101000110 ; Add divisor to R5 for every 1 in the quotient.
0001100100111111 ; Subtract 1 from the quotient.
0000011111111101 ; Keep going until quotient is negative.
0010110000000100 ; Load the address where output will be stored in R6.
0111101110000000 ; Store the value of quotient*divisor at the address in R6.
1111000000100101 ; HALT
0100000000000000 ; Address where dividend is stored.
0100000000000001 ; Address where divisor is stored.
0101000000000000 ; Address where output will be stored.
```

Assembly code:
(Note: You were not expect to turn this in for this assignment. This is for future reference.)

```
                .ORIG x3000          ; Starting address of the program
                LD R4, DIVIDEND      ; Load the address of the dividend into R4
                LD R5, DIVISOR       ; Load the address of the divisor into R5
                LDR R0, R4, #0       ; Load the dividend to R0
                LDR R1, R5, #0       ; Load the divisor to R1 which will be complemented

                NOT R1, R1           ; Get 2's complement value for the divisor
                ADD R1, R1, #1       ; therefore R1 has the value of -divisor

                AND R2, R2, #0       ; Clear R2, it will contain the value of quotient finally

; Loop starts. The loop iterates over and over until the updated value is less than the divisor.
; After all iterations end, R2 contains value of the quotient while R0 contains the value of
; remainder.

LOOP            ADD R2, R2, #1       ; Increment R2 for quotient, until the loop breaks
                ADD R0, R0, R1       ; In each iteration, get the result from the subtraction
                ADD R3, R0, R1       ; Forward check if the intermediate result is already less
                                      ; than the divisor

                BRzp LOOP            ; If R3 is non-negative, go 3 steps back

                ADD R0, R0, #0              ; Add 0 to the remainder, so we can break on it.
                BRz REMAIN_IS_ZERO          ; If there is no remainder, output the dividend.
                BRp REMAIN_IS_POS           ; If there is a remainder, output quotient*divisor.

REMAIN_IS_ZERO  LD R6, OUTPUT        ; Load the address where output will be stored in
                                     ; R6.
                LD R7, DIVIDEND      ; Reload the address of the dividend into R7.
                LDR R7, R7, #0       ; Load the dividend to R7.
                STR R7, R6, #0       ; Store the dividend at the address in R6.
                BRnzp STOP           ; Break to stop to exit the program.

REMAIN_IS_POS   LD R6, DIVISOR       ; Reload the address of the divisor into R6.
                LDR R6, R6, #0       ; Load the divisor to R6.
                AND R4, R4, #0       ; Clear R4 for multiplication.
                ADD R4, R4, R2       ; Set R4 to the quotient of division for multiplication.
                AND R5, R5, #0       ; Clear R5 for multiplication.

; Loop starts. This loop iterates over and over until the quotient equals 0. During each iteration
; the divisor is added to a sum. The sum at the end of the iterations is the quotient*divisor.
```

```
MULT_LOOP       ADD R5, R5, R6          ; Add divisor to R5 for every 1 in the
                                        ; quotient.
                ADD R4, R4, #-1         ; Subtract 1 from the quotient.
                BRzp MULT_LOOP          ; Keep going until quotient is negative.

                LD R6, OUTPUT           ; Load the address where output will be
                                        ; stored in R6.
                STR R5, R6, #0          ; Store the value of quotient*divisor at the
                                        ; address in R6.

STOP            HALT                    ; Exit the program.

DIVIDEND .FILL x4000                ; Address where dividend is stored.
DIVISOR .FILL x4001                 ; Address where divisor is stored.
OUTPUT .FILL x5000                  ; Address where output will be stored.

                .END
```