Homework 6 – Due 12:00PM on Monday, April 9

Primary contact for this homework: Yinggang Huang [yinggang@cs.wisc.edu]

You must do this homework in groups of **two**. This homework must be submitted **online**. No hard copies will be accepted.

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 GroupMembers, UWIDs, 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) per group to the folder **homework6**.
2. Name the archive file with the following format: Lastname1Lastname2 with .zip or .tar.gz as suffix where Lastname1 is the last name with 1st letter capitalized for one of the group members while Lastname2 is the last name for the other.
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 – Pseudo-code for problem 2
   D. hw6_p2.bin – Binary code for problem 2
   E. hw6_p3.txt – Pseudo-code for problem 3
   F. hw6_p3.bin – Binary code for problem 3
   G. hw6_p4.txt – Pseudo-code for problem 4
   H. hw6_p4.bin – Binary code for problem 4
   I. README.txt - Readme file that contains the names, student IDs, and section numbers for all members of your group and the HALT addresses for all the problems (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 (3 points)

Write a short LC-3 program in PennSim that compares the 2 numbers in R1 and R2, and then puts the value 0 in R0 if R1 = R2, 1 if R1 > R2, -1 if R1 < R2. Finally, store the result to memory location 0x4000.

a. (1.5 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.

Solution:

```
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 == 0
If R4 > 0, increment R0 by 1 so R0 == 1
If R4 < 0, decrement R0 by 1 so R0 == -1
Store the value of R0 to memory location x4000
```

b. (1.5 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.

Solution:
Assembly Code for LC-3:

```
;
; This assembly program is implemented to compare the values of R1 and R2, R0 serves as a status
; indicator for the comparison result. R0 is assigned 0 if R1 = R2, 1 if R1 > R2, -1 if R1 < R2.
;

START       .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 STOP            ; Jump to STOP so R0 is still 0, meaning R1 = R2
            BRp GREATER          ; Jump to GREATER so R0 is updated to 1
            BRn LESS            ; Jump to LESS so R0 is updated to -1
GREATER   ADD R0, R0, #1        ; R0 = 1 if R1 > R2
            BRp STOP            ; After update, the program should proceed to store result
LESS        ADD R0, R0, #-1     ; R0 = -1 if R1 < R2
STOP        LD R6, OFFSET       ; Load R6 with address x4000
            STR R0, R6, #0      ; Store the result into memory location x4000
            HALT
OFFSET      .FILL x4000
            .END
```

The corresponding machine code for LC-3:

```
0011 0000 0000 0000; x3000;     .ORIG x3000 (This is PC's initial address, not ST Instruction)
0101 0000 0010 0000; x5020;     R0 is clear to 0
1001 0110 1011 1111; x96BF;     Store the 1's complement value of R2 to R3
0001 0110 1110 0001; x16E1;     Add R3  by 1 and then store the result to R3
0001 1000 0100 0011; x1843;     ADD R1 and R3 then store result in R4
0000 0100 0000 0101; x0405;     If R1 = R2, then proceed to HALT instruction, R0 = 0
0000 0010 0000 0001; x0201;     If R1 > R2, then jump to GREATER
0000 1000 0000 0010; x0802;     If R1 < R2, then jump to LESS
0001 0000 0010 0001; x1021;     If R1 > R2, update R0 to 1
0000 0010 0000 0001; x0201;     After updating R0 to 1, jump to STOP to halt the program
0001 0000 0011 1111; x103F;     If R1 < R2, update R0 to -1 (xFFFF)
0010 1100 0000 0010; x2C02;     Load R6 with address x4000
0111 0001 1000 0000; x7180;     Store the result into memory location x4000
1111 0000 0010 0101; xF025;     Program halts
0100 0000 0000 0000; x4000;     The address that the final result is store
```

Problem 2 (9 points)

Write an LC-3 program that calculates factorial of n where n is a non-negative number. Use the value stored at memory location 0x4000 for input n and memory location 0x4001 to store the result of the computation. You can implement this based on the multiplication code derived in HW5. Assume the output is represented by 16 bits, the word length of LC-3.

a. (4 points) Write the pseudo code for the factorial algorithm. Please submit your pseudo code in a file **exactly** named as “hw6_p2.txt”, without the double quotes, to dropbox.

Solution:

```
Load the value to be calculated from memory location x4000, store to R1
Initialize R2 to be (n – 1) to be multiplied by n
Initialize R3 as a flag to for factorial calculation
Clear R0 to 0

Add R0 with R1
Decrement R2 to count how many times R1 has been added
If R2 is still positive, go back 2 steps.

Clear R1 to 0
Store R0 into R1
Clear R0 to 0
Decrement R3, calculation ends if R3 reaches 0
R2 = R3 so that the multiplication can continue if R3 has not reached 0 yet
If R3 is still greater than 0, go back 8 steps.

Finally the factorial result is stored into memory location x4001
```

b. (5 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_p2.bin”, without the double quotes.

Solution:

Assembly Code for LC-3:

```
;
; This assembly program is implemented to calculate factorial of n (1 <= n <= 7), the result can only
; be represented by 16 bits. Initially the value n is loaded from memory location x4000, finally the
; result is stored into memory location x4001.
;

START                   .ORIG x3000             ; Starting address of the program is x3000

                        LD R6, OFFSET           ; Set R6 with value x4000
                        LDR R1, R6, #0          ; Load value n from memory location x4000
                        ADD R2, R1, #-1         ; Initialize R2 as (n-1) that serves as multiplication factor
                        ADD R3, R2, #0          ; Initialize R3 as (n-1) that serves as a flag for factorial
                        AND R0, R0, #0          ; Clear R0 to 0


;
; This innerLoop is an implementation of multiplication operations, R0 contains result of R1 * R2.
;
innerLoop               ADD R0, R0, R1          ; Update R0's value
                        ADD R2, R2, #-1         ; Count the number of times R1 has been added
                        BRp innerLoop           ; If R1 has not been added as many times as R2, loop
                                               ; again
;
; This outerLoop checks whether the factorial is done, it directs back to innerLoop if not.
;
outerLoop               AND R1, R1, #0          ; Clear R1
                        ADD R1, R1, R0          ; Store the current result into R1
                        AND R0, R0, #0          ; Clear R0 to 0
                        ADD R3, R3, #-1         ; Count of iterations of outerLoop, halts if R3 = 0 is
                                                ; reached
                        ADD R2, R2, R3          ; Store value of R3 to R2 so that the multiplication can
                                                ; continue if possible

                        BRp innerLoop           ; Jump back to innerLoop if R3 or R2 is still positive

                        STR R1, R6, #1          ; Finally the factorial is stored into memory location
                                                ; x4001

                        HALT
OFFSET                  .FILL x4000
                        .END
```

The corresponding machine code for LC-3:

```
0011 0000 0000 0000; x3000;            .ORIG x3000 (This is PC’s initial address, not ST Instruction)
```

```
0010 1100 0000 1111; x2C0F;        Set R6 with value x4000
0110 0011 1000 0000; x6380;        Load value n from memory location x4000
0001 0100 0111 1111; x147F;        Initialize R2 as (n-1) that serves as multiplication factor
0001 0110 1010 0000; x16A0;        Initialize R3 as (n-1) that serves as a flag for factorial
0101 0000 0010 0000; x5020;        Clear R0 to 0
0001 0000 0000 0001; x1001;        Update R0's value
0001 0100 1011 1111; x14BF;        Count the number of times R1 has been added
0000 0011 1111 1101; x03FD;        If R1 has not been added as many times as R2, loop again
0101 0010 0110 0000; x5260;        Clear R1
0001 0010 0100 0000; x1240;        Store the current result into R1
0101 0000 0010 0000; x5020;        Clear R0 to 0
0001 0110 1111 1111; x16FF;        Count of iterations of outerLoop, halts if R3 = 0 is reached
0001 0100 1000 0011; x1483;        Store value of R3 to R2 so that the multiplication can continue if
                                   possible
0000 0011 1111 0111; x03F7;        Jump back to innerLoop if R3 or R2 is still positive
0111 0011 1000 0001; x7381;        Finally the factorial result is stored into memory location x4001
1111 0000 0010 0101; xF025;        Program halts
0100 0000 0000 0000; x4000;        The address that the initial value for factorial is stored
```

Problem 3 (10 points)

Write an LC-3 machine language program which divides the number in memory location x4000 by the number in memory location x4001 and stores the quotient at x5000 and the remainder at x5001. Assume both dividend and divisor are positive.

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

Solution:

```
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

Load the address of the quotient into R4
Load the address of the remainder into R5
Store quotient from R2 into x5000
Store remainder from R0 into x5001
```

b. (7 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.

Solution:

Assembly Code for LC-3:

```
;
; This program implements division operation. It divides the number in memory location x4000
; by the number in memory location x4001 and stores the quotient at x5000 and the remainder
at ;x5001.
;

START       .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 interations 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

;
; Store the results
;
            LD R4, QUOTIENT             ; Load the address of the quotient into R4
            LD R5, REMAINDER            ; Load the address of the remainder into R5
            STR R2, R4, #0              ; Store quotient into x5000
            STR R0, R5, #0              ; Store remainder into x5001

            HALT

DIVIDEND        .FILL   x4000
DIVISOR         .FILL   x4001
QUOTIENT        .FILL   x5000
REMAINDER       .FILL   x5001

                .END
```

The corresponding machine code for LC-3:

```
0011 0000 0000 0000; x3000;          .ORIG x3000 (This is PC’s initial address, not ST Instruction)
0010 1000 0000 1111; x280F;          Load the address of the dividend into R4
0010 1010 0000 1111; x2A0F;          Load the address of the divisor into R5
0110 0001 0000 0000; x6100;          Load the dividend to R0
0110 0011 0100 0000; x6340;          Load the divisor to R1 which will be complemented
1001 0010 0111 1111; x927F;          NOT(R1) and then store it again into R1
0001 0010 0110 0001; x1261;          R1 is incremented by 1
0101 0100 1010 0000; x54A0;          Clear R2, it will contain the value of quotient finally
0001 0100 1010 0001; x14A1;          LOOP begins: Increment R2 for quotient, until the loop breaks
0001 0000 0000 0001; x1001;          R0 <--- R0 + R1
0001 0110 0000 0001; x1601;          R3 <--- R0 + R1
0000 0111 1111 1100; x07FC;          If R3 is non-negative, go 3 steps back
0010 1000 0000 0110; x2806;          Load the address of the quotient (x5000) into R4
0010 1010 0000 0110; x2A06;          Load the address of the remainder (x5001) into R5
0111 0101 0000 0000; x7500;          Store quotient into x5000
0111 0001 0100 0000; x7140;          Store remainder into x5001
1111 0000 0010 0101; xF025;          Program halts
0100 0000 0000 0000; x4000;          Address of the dividend
0100 0000 0000 0001; x4001;          Address of the divisor
0101 0000 0000 0000; x5000;          Address of the quotient
0101 0000 0000 0001; x5001;          Address of the remainder
```

Problem 4 (8 points)

The LC-3 has no left shift or right shift instruction.

Left shifting means that every bit in a binary string is moved to the left and the right bits are filled in with 0s. For example, the binary string 00110010 when left-shifted by one bit moves every bit one position to the left, which results in 01100100. We can shift a number to the left by one bit position by adding it to itself. For example, when the binary number 0011 is added to itself, the result is 0110 which is exactly one bit left-shift from 0011. Note that lowest bits will be filled with zeros.

But how does right-shifting sound? Let's solve it. Similar to left-shifting, right-shifting means that every bit in a binary string is moved to the right and the left bits are filled in with 0's. For example, 1010 after right-shifting by one bit results in 0101. The algorithm we want you to design should take in a 16-bit value at memory location M1 that contains the number that will have its bits shifted and a value at M2 that contains the number of bits to shift. The 16-bit result should be stored in M3. For instance, suppose M1 = 0x0010 and M2 = 0x0004. After execution of the program, M3 = 0x0100. Note that the highest bits will be filled with zeros. You can assume the MSB of the input is always 0 so that the input value to be right shifted is positive. Also assume if the value in M2 is zero or negative, no shift is performed and original 16-bit value should be stored into M3.

a. (3 points) Write the pseudo code for the right shift algorithm as described above. Please submit your pseudo code in a file **exactly** named as "hw6_p4.txt", excluding the double quotes.

Solution:

```
Load the input value that should be right-shifted to R0
Load the number of right-shifts into R4
If the number of right-shifts is 0 or even negative, shift operation should not be performed,
simply R0 should be stored into M3 as the final result.

Assign -2 to R1 serving as a divisor for right shift
Clear R2 to 0 for containing quotient

innerLoop: increment R2 by 1 until the innerLoop breaks
           R0 is decremented by 2 (R0 <- R0 + R1)
           Forward check if R0 is less than the 2 (R3 <- R0 + R1)
           If R3 is already negative, break the innerLoop, otherwise continue the loop
outerLoop: Assign the then-current quotient to R0 (R0 <- R2 + 0)
           Clear R2 to 0
           R4 is decremented by 1 after one right shift is done
           Go back to innerLoop until R4 is not positive any more

Store the final result to M3
```

b. (5 points) Write a program using LC-3 binary code that implements your algorithm, with the following as your memory locations:

M1: 0x5000

M2: 0x5001

M3: 0x6000

Write comments for each line of code explaining what it does. The binary code named **exactly** “hw6_p4.bin” should be submitted to the dropbox.

Solution:

Assembly Code for LC-3:

```
;
; This program implements right shifting algorithm on 16-bit values. Every bit in a binary string
; is moved to the right and the left bits are filled in with 0’s. It should take in a 16-bit value
; at memory location M1 that contains the number that will have its bits shifted and a value at M2
; that contains the number of bits to shift. The 16-bit result should be stored in M3. NOTE: if the
; value stored in M2 is a zero or negative number, no shift is performed and in this case the value
; stored in M1 will be directly stored into M3.
;

START       .ORIG x3000             ; Starting address of the program
            LD      R5, M1          ; Load the address of the input value into R5
            LDR R0, R5, #0          ; Load the input value to R0

            LD      R5, M2          ; Load the address to R5 where the number of right-
                                    ; shifts is stored
            LDR     R4, R5, #0      ; Load the number of right-shifts into R4
            BRnz UNCHANGE           ; If the number of right-shifts is 0 or even negative, shift
                                    ; operation is not performed
```

```
                                                  ;
                AND R1, R1, #0          ; Clear R1 to 0
                ADD R1, R1, #2          ; Assign 2 to R1
                NOT R1, R1              ; Take 1's complement value of R1 and then assign to
                                        ; itself
                ADD R1, R1, #1          ; Take 2's complement value, now R1 is -2

                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 interations in the innerLoop end, R2 contains the right shifted bits. The outerLoop
; prompts another inner loop (for another time of right shifting) depending on the number stored
; in M2.
;
innerLoop       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 innerLoop          ; If R3 is non-negative, go 3 steps back

outerLoop       ADD R0, R2, #0          ; Assign the right shifted bits to R0 for another right
                                        ; shifting if necessary
                AND R2, R2, #0          ; Clear R2 to 0, now only R0 contains current right-
                                        ; shifted bits
                ADD R4, R4, #-1         ; R4 should be decremented by 1 after 1-time of right-
                                        ; shift is done
                BRp innerLoop                   ; If R4 is still positive, go back to innerLoop

;
; Store the results
;
UNCHANGE        LD  R5, M3                      ; Load the address to which the result should be stored
                                                ; into R5
                STR R0, R5, #0                  ; Store final result to M3

                HALT

M1              .FILL           x5000
M2              .FILL           x5001
M3              .FILL           x6000

                .END
```

The corresponding machine code for LC-3:

```
0011 0000 0000 0000; x3000;         .ORIG x3000 (This is PC’s initial address, not ST Instruction)
0010 1010 0001 0100; x2A14;         Load the address of the value to be right shifted into R5
0110 0001 0100 0000; x6140;         Load the input value to R0
0010 1010 0001 0011; x2A13;         Load address to R5 where the number of right-shifts is stored
0110 1001 0100 0000; x6940;         Load the number of right-shifts into R4
0000 1100 0000 1101; x0C0D;         If the number of right-shifts is 0 or even negative, skip over the
                                    right shift algorithm
0101 0010 0110 0000; x5260;         Clear R1 to 0
0001 0010 0110 0010; x1262;         Assign 2 to R1
1001 0010 0111 1111; x927F;         Take 1’s complement of R1 into R1
0001 0010 0110 0001; x1261;         Take 2's complement value, now R1 is -2
0101 0100 1010 0000; x54A0;         Clear R2 to 0
0001 0100 1010 0001; x14A1;         innerLoop begins: Increment R2 for quotient, until the loop
                                    Breaks
0001 0000 0000 0001; x1001;         R0 <- R0 + R1
0001 0110 0000 0001; x1601;         Forward check if the intermediate result is already less
                                    than the divisor
0000 0111 1111 1100; x07FC;         If R3 is non-negative, go back to innerLoop
0001 0000 1010 0000; x10A0;         outerLoop begins: Assign the right shifted bits to R0 for
                                    another right shift if necessary
0101 0100 1010 0000; x54A0;         Clear R2 to 0
0001 1001 0011 1111; x193F;         R4 is decremented by 1 after 1-time of right shift is done
0000 0011 1111 1000; x03F8;         If R4 is still positive, go back to innerLoop
0010 1010 0000 0100; x2A04;         Load M3 (address) to R5
0111 0001 0100 0000; x7140;         Store final shifted result to M3
1111 0000 0010 0101; xF025;         Program halts
0101 0000 0000 0000; x5000;         M1, the address of the input value to be right shifted
0101 0000 0000 0001; x5001;         M2, the address of the number of right shifts
0110 0000 0000 0000; x6000;         M3, the address into which the final result should be stored
```