Question 1: [25 points]

(a) Give the range of integers representable using 4-bit two's complement notation.

(b) Perform the operation:

```
      -8
-   (-1)
=   -  7
```

(c) Perform the operation:

```
      -8
x   (-1)
=     +8
```

using 4-bit two's complement multiplication following the **Booth** algorithm.

(d) Consider the following floating point number format:

```
 Sign        Exponent     Mantissa
1 bit         3 bits        4 bits
```

The exponent is stored in **excess-4** notation. The base is **2**. The mantissa is stored in normalized form **with** a "hidden" 1 bit. There are **no** reserved values of exponents used for special cases such as exact zero, infinity, and Nan.

(i) What decimal value does the number `0 110 1011` represent in this format?

(ii) Express `+6.75` in this floating point format.

(iii) What are the decimal values of the **smallest positive** number and the **next smallest positive** number representable in this format? Give also the binary bit pattern used to represent these numbers in this floating point format.

Question 2: [25 points]

Consider the following truth table which represents the operation of some combinational logic circuit:

Table 1: Truth Table for some "mystery combinational logic circuit"

IN_1	IN_0	OUT_1	OUT_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

(a) Draw the digital logic diagram which corresponds to the above table (the inputs to the circuit are IN_1 and IN_0 and the outputs of the circuit are OUT_1 and OUT_0.). Make use of **ONLY** the following types of logic gates in your diagram: inverters, 2-input AND gates, 2-input OR gates. Your may use as many gates of each type as necessary. Identify clearly in your diagram the inputs and the outputs.

Hint: Here is the hardware for the output OUT_0 (it is simply a wire) :

IN_0 _____ OUT_0

(b) Does the 2-bit parallel adder shown in Figure 1 give the same output as specified by the above truth table?

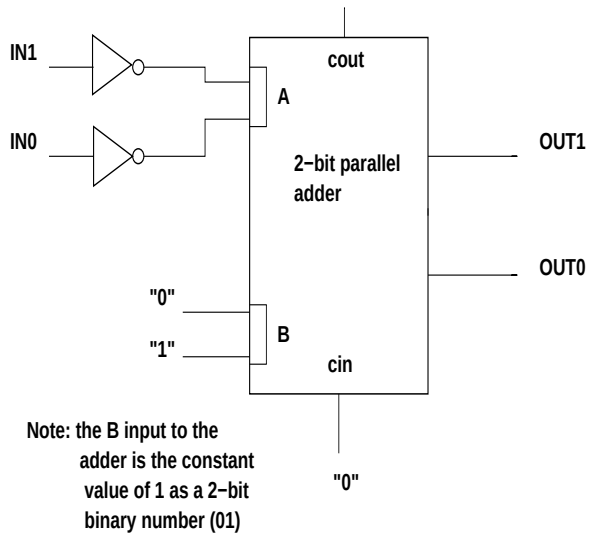

Figure 1: A certain configuration of a two-bit parallel adder.

Question 3: [25 points]

Consider the following assembly language program for the example language explained in class. Recall that in this language, the following conventions are used:

# indicates immediate data
() indicates that indirection is to be used as the addressing mechanism

Instructions are represented in the following format:

```
opcode source , destination
opcode source1, source2, destination
```

```
                        ; This is the program
        ANSWER EQU 1000
        MOV #1,  R0
        MOV #1,  R1
        CLR R2          ; clear the "sum"
        MOV #4,  R3     ; initialize loop counter
        MOV #ANSWER,  R4  ; R4 used as a "pointer" to memory

RABBIT: ADD R0, R1, R2  ; R2 = R0 + R1
        MOV R2, (R4)    ; store R2 into memory
        MOV R1, R0
        MOV R2, R1
        INC R4          ; point to next memory location
        DEC R3          ; decrement loop counter
        BRANCH > 0 RABBIT ; go back and do it again if not
                        ; yet finished the loop
        HALT            ; else we stop the computer and end
```

(a) Complete the following table (in your answer booklets) which shows the contents of the CPU registers R0, R1, R2, R3, and R4 at the **end** of each iteration of the loop designated by the label `RABBIT.`

Table 2: CPU register contents during loop

Register	Initial value before loop	1st iteration	2nd iteration	3rd iteration	4th iteration
R0	1				
R1	1				
R2	0				
R3	4				

Table 2: CPU register contents during loop

Register	Initial value before loop	1st iteration	2nd iteration	3rd iteration	4th iteration
R4	you figure it out				

(b) What values will be stored in the 4 consecutive memory locations starting at address 1000?

(c) Express in a few clear, concise English language sentences what this program does. Here is an example of how you should express your answer:

"This program computes the value of the factorial of the integer stored in memory location ANSWER. The value of ANSWER! is stored in CPU register R2 upon program termination"

Of course, the program actually performs something completely different. **DO NOT** express your answer in terms similar to:

"This program, moves the immediate data 1 into regisdter R0 and R1 and then clears register R2 and moves immediate data 4 into register R3. It then moves the immediate value represented by the symbolic label ANSWER into register R4. The program then enters a loop in which it adds registers R0 and R1 together and stores the resulting sum into register R2 blah blah blah ..."

In other words, do not simply perform a literal translation from assembly language to English. **Answers of this type will result in a grade of 0.**

Question 4: [25 points]

For the 3-bus CPU organization presented in class and shown in Figure 2, complete the given steps (by filling in the blanks) expressed in register-transfer notation necessary to **execute** the following instruction:

```
SUM X
```

where X refers to a location in main memory. This instruction performs the following operation:

M[X] <--- M[X] + R1

The contents of register R1 are added with the contents of memory location X and the resulting sum is stored back into memory location X.

You may assume that the instruction is stored in one main memory location.

T_i: BUS A <--- _____ , F = A , _____ <--- BUS C

T_{i+1}: _____ , MDR <--- _____

T_{i+2}: BUS A <--- _____ , BUS B <--- _____ , F = A + B ,
_____ <--- BUS C

T_{i+3}: Data_in_of_memory <--- Bus A , _____ <- MDR , Write, END

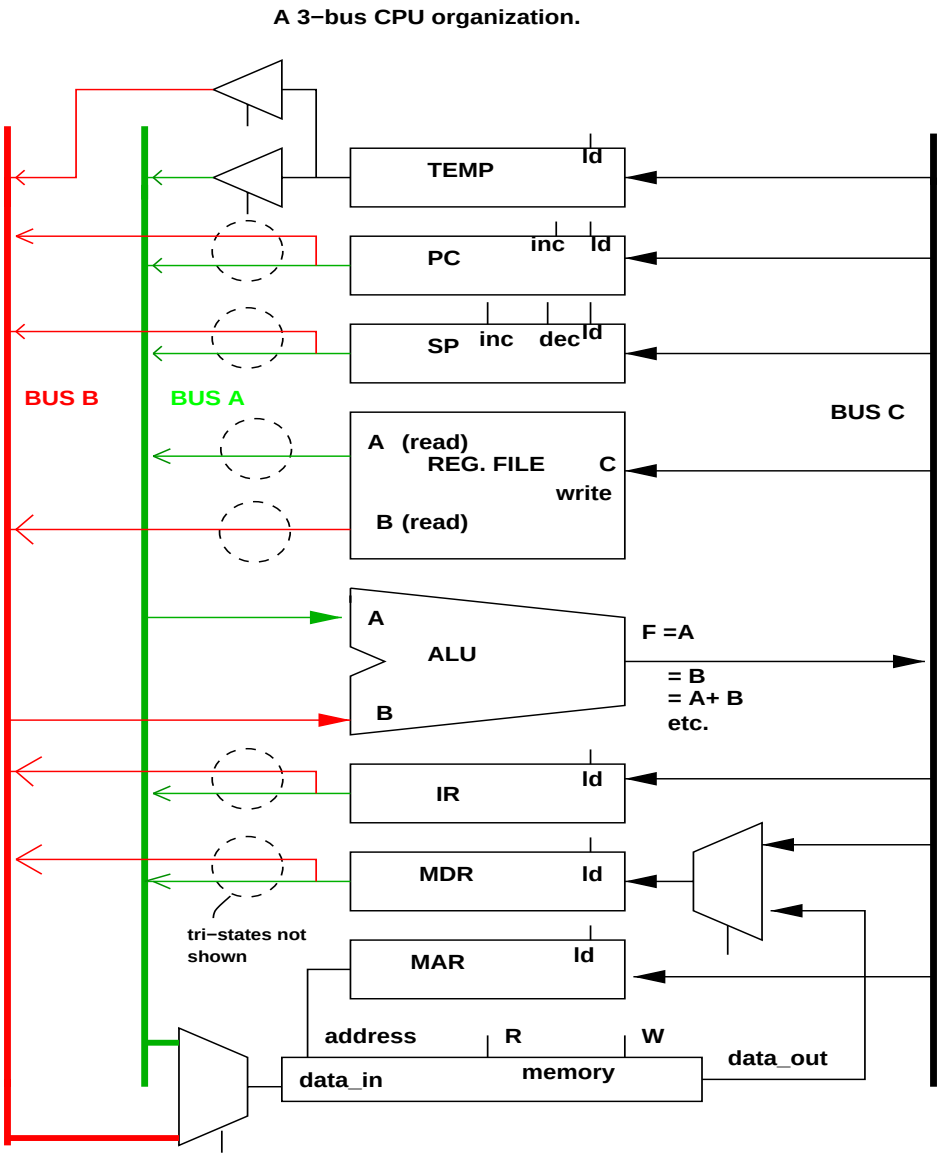

Figure 2: 3-bus CPU organization.