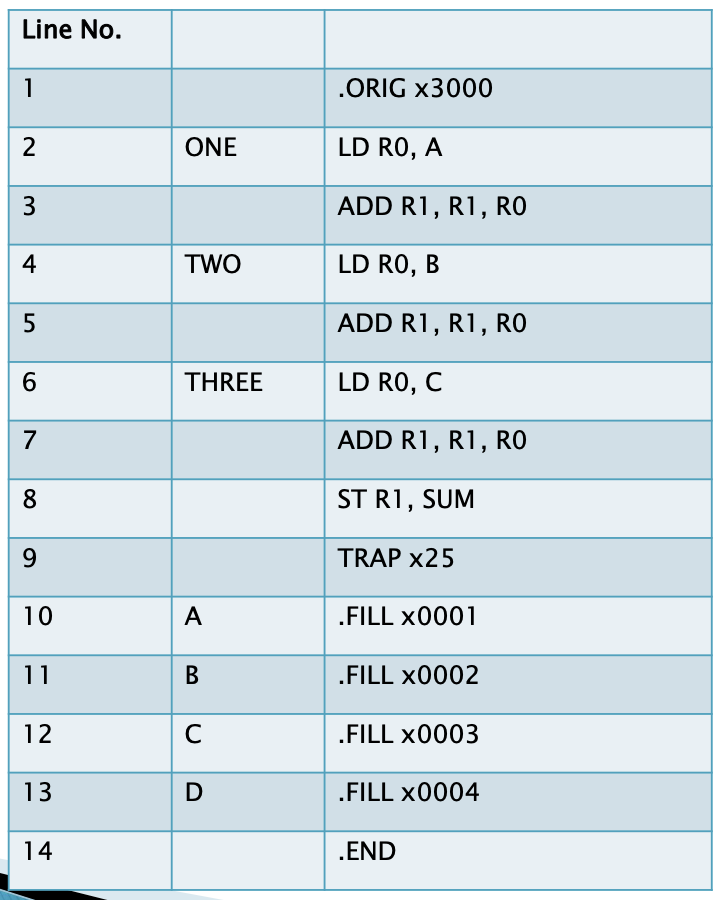
Assignment 4

Computer Organization

Deadline: 11:55pm, Dec 19, 2024

Student ID:\_\_\_2330016056\_\_\_\_\_ Name:\_\_\_\_\_\_\_Bohan YANG\_\_\_\_\_\_\_\_\_\_\_

1. The following program adds the values stored in memory locations A, B, and C, and stores the result into memory. There are **two errors** in the code. For each, describe the error and indicate whether it will be detected at assembly or run time. (10 points)



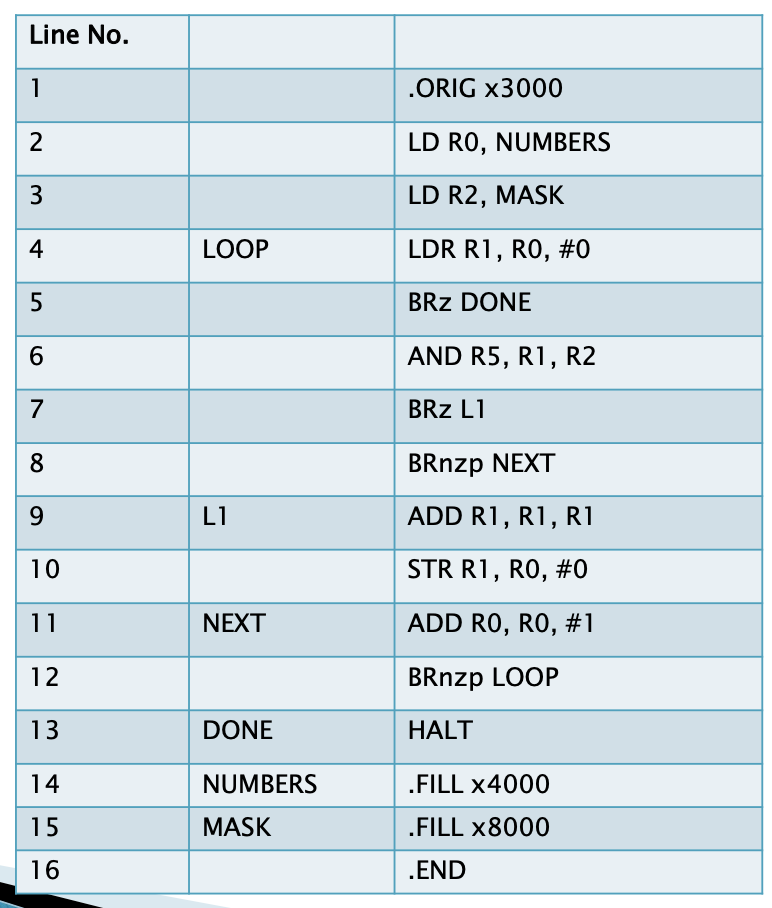
Answer:

The first error is: label SUM not defined. This will be detected at assembly. It should be SUM .BLKW x1

The second error is: Register R1 Uninitialized. It should be AND R1, R1, #0 ; Initialize R1 to 0

|  |
| --- |
| .ORIG x3000    AND R1, R1, #0 ; Initialize R1 to 0    ONE LD R0, A  ADD R1, R1, R0  TWO LD R0, B  ADD R1, R1, R0  THREE LD R0, C  ADD R1, R1, R0    ST R1, SUM  TRAP x25    A .FILL x0001  B .FILL x0002  C .FILL x0003  D .FILL x0004  SUM .BLKW x1 ; Reserve space for SUM    .END |

1. The following is an LC-3 program that performs a function. Assume a sequence of integers is stored in consecutive memory locations, one integer per memory location, starting at the location x4000. The sequence of numbers terminates with the number x0000. What does the following program do? (10 points)



Answer:

The program processes a sequence of integers stored consecutively in memory starting at x4000. For each non-zero integer:

It checks whether the integer is positive or negative by using a bitwise AND operation with the mask x8000 (which checks the sign bit).

If the integer is positive, the program doubles it (shifts it left by 1 bit) and stores it back into memory.

The program continues processing integers in memory until it encounters the terminator value x0000, which ends the sequence.

1. Write a program (div.asm) to perform a positive integer division algorithm. Your program will have two inputs: the dividend and divisor and have two outputs: the quotient and remainder. Assume, you will be given only positive values and the divisor will be always greater than zero. Your inputs and outputs are as follows: (20 points)

Register R1 = Divisor

Register R6 = Dividend

Register R2 = Quotient

Register R3 = Remainder.

Example: R1 = 30, R6 = 100. Therefore 100/30, R2 = 3 and R3 = 10.

You can consider to use the follow program framework:

.ORIG x3000

; below is your code

HALT

DIVISOR .FILL #30

DIVIDND .FILL #100

.END

Answer:

|  |
| --- |
| .ORIG x3000  LD R1, DIVISOR  LD R6, DIVIDND  NOT R4, R1  ADD R4, R4, #1  LOOP  ADD R6, R6, R4  BRn DONE  ADD R2, R2, #1  BRnzp LOOP  DONE ADD R3, R6, R1  HALT  DIVISOR .FILL #30  DIVIDND .FILL #100  .END |

1. What is the benefit of using interrupts to handle I/O? When might you use polling for handling I/O? (5 points)

Answer:

Comparing to interrupts, polling consumes a lot of cycles checking the ready bit, especially for rare I/O event – these cycles can be used for more computation. And with interrupts, the CPU can immediately respond to an I/O event as soon as it occurs, ensuring timely processing of critical tasks.

Polling is preferable in certain scenarios, such as Time-Critical Applications: If the CPU must frequently check for data (e.g., in systems where the device status changes rapidly and predictably), polling can reduce the latency caused by interrupt handling overhead.

1. What is the last instruction of an interrupt service routine? What does it do? (5 points)

Answer:

The last instruction is RTI (Return from Interrupt).

RTI restores the CPU registers (Program Counter, Condition Codes, etc.) to the state they were in before the interrupt occurred. This ensures the interrupted program can resume execution without any disruption.

After restoring the state, the CPU continues executing the next instruction of the program that was interrupted, as if the interrupt had never happened.