ICS Homework 5

November 14, 2024

Question 1

What is the purpose of the `.END` pseudo-op? How does it differ from the `HALT` instruction?

Question 2

We would like to have an instruction that does nothing. Many ISAs actually have an opcode devoted to doing nothing. It is usually called `NOP`, for NO OPERATION. The instruction is fetched, decoded, and executed. The execution phase is to do nothing! Which of the following three instructions could be used for `NOP` and have the program still work correctly?

a. 0001 001 001 1 00000

b. 0000 111 000000001

c. 0000 000 000000000

Question 3

Suppose you write two separate assembly language modules that you expect to be combined by the linker. Each module uses the label `AGAIN`, and neither module contains the pseudo-op `.EXTERNAL AGAIN`.

Is there a problem using the label `AGAIN` in both modules? Why or why not?

Question 4

Two students wrote interrupt service routines for an assignment. Both service routines did exactly the same work, but the first student accidentally used `RET` at the end of his routine, while the second student correctly used `RTI`.

There are three errors that arose in the first student's program due to his mistake. Describe all of them.

Question 5

The input stream of a stack is a list of all the elements we pushed onto the stack, in the order that we pushed them. The output stream is a list of all the elements that are popped off the stack in the order that they are popped off.

(1) If the input stream is ABCD, create a sequence of pushes and pops such that the output stream is BDCA.

(2) If the input stream is ABCD, is it posible to create a sequence of pushes and pops such that the output stream is DBCA? Why?

(3) If the input stream is ABCDEF, how many different output streams can be created? Only consider output streams that are 6 characters long.

Question 6

(1) When the following LC-3 program is executed, how many times will the instruction at the memory address labeled `LOOP` execute?

```
        .ORIG x3005
        LEA R2, DATA
        LDR R4, R2, #0
LOOP    ADD R4, R4, #-3
        BRzp LOOP
        TRAP x25
DATA    .FILL x8002
        .END
```

(2) Now if we replace the instruction in x300A with `DATA .FILL x8003`, will your answer be the same?

Question 7

Figure 8.18 in P286 describes a FIB subroutine as shown below:

```
;FIB subroutine
; + FIB(0) = 0
; + FIB(1) = 1
; + FIB(n) = FIB(n-1) + FIB(n-1)
;
; Input is in R0
; Return answer in R1
;
FIB     ADD R6, R6, #-1
        STR R7, R6, #0 ; Push R7, the return linkage
        ADD R6, R6, #-1
        STR R0, R6, #0 ; Push R0, the value of n
        ADD R6, R6, #-1
        STR R2, R6, #0 ; Push R2, which is needed in the subroutine
; Check for base case
        AND R2, R0, #-2
        BRnp SKIP ; Z=0 if R0=0,1
        ADD R1, R0, #0 ; R0 is the answer
        BRnzp DONE
; Not a base case, do the recursion
SKIP    ADD R0, R0, #-1
        JSR FIB ; R1 = FIB(n-1)
        ADD R2, R1, #0 ; Move result before calling FIB again
        ADD R0, R0, #-1
        JSR FIB ; R1 = FIB(n-2)
        ADD R1, R2, R1 ; R1 = FIB(n-1) + FIB(n-2)
; Restore registers and return
DONE    LDR R2, R6, #0
        ADD R6, R6, #1
        LDR R0, R6, #0
        ADD R6, R6, #1
        LDR R7, R6, #0
        ADD R6, R6, #1
        RET
```

(1) Is `R0` caller save or callee save? What about `R2` and `R7`?

(2) The following table shows the instruction cycles used to execute this subroutine for the input n:

n	2	3	4	5	6	7	8	9	10
$T(n)$	55	93	169	283	473	777	1271	2069	3361

Can you define $T(n)$ recursively assuming n is sufficiently large, and explain your definition?

(3) How can you improve the efficiency of this *recursive subroutine*? Do not use iterative methods instead.

Question 8

We have a program with some missing instructions, and we have a table consisting of some information and some missing information associated with five specific clock cycles of the program's execution. Your job is to complete both!

(1) Insert the missing instructions in the program and the missing information in the table. Cycle numbering starts at 1. That is, cycle 1 is the first clock cycle of the processing of `LD R0, A`. Note that we are asking for the value of the registers DURING each clock cycle.

We have not said anything about the number of clock cycles a memory access takes, but you do have enough information to figure that out for yourself.

```
1          .ORIG x3000
2          LD R0, A
3          LD R1, B
4          NOT R1, R1
5          ADD R1, R1, #1
6          AND R2, R2, #0
7  AGAIN   -------------- (a)
8          -------------- (b)
9          BRnzp AGAIN
10 DONE    ST R2, C
11         HALT
12 A       .FILL #5
13 B       .FILL -------- (c)
14 C       .BLKW #1
15         .END
```

Cycle Number	State Number	Information					
(d)	(e)	LD.REG: 1	DRMUX: (f)	GateMDR: (g)			
		LD.CC: (h)	GateALU: (i)	GatePC: (j)			
16	30	LD.MDR: (k)	MDR: (m)	IR: (n)			
		LD.IR: (l)					
50	(o)	LD.REG: 1	MDR: (p) x_4A_	DRMUX: (q)			
		BUS: x0001		GateMDR: (r)			
57	1	PC: (s)	IR: (t) x_040	GateALU: (u)			
		BUS: x0003		GatePC: (v)			
(w)	22	ADDR1MUX: (x)	ADDR2MUX: (y)				
		LD.PC: 1	PC: x3008	PCMUX: ADDER			

(2) Actually, the program was written by a student, so as expected, he did not get it quite right. Almost, but not quite! Your final task on this problem is to examine the code, figure out what the student was trying to do, and point out where he messed up and how you would fix it. It is not necessary to write any code, just explain briefly how you would fix it.