PB22051087 YufengPeng

T1

(1) **0101 001 010 1 00100** If the result is 0, machine 2 is busy.

(2) **0101 011 010 1 01100** If the result is 0, machine 2 and 3 are both busy, otherwise 2 and 3 have at least one is available.

(3) **0101 011 010 1 1111** If the result is 0, all machines are busy.

(4) Only one LC-3 instruction **can not** judge if machine 6 is busy or not, because 5 bits can not represent **0000 0000 0100 0000**

T2

We can find that those instructions will do :

x30FF	1110 001 000000001	LEA R1, #1
x3100	0110 010 001 000010	LDR R2, R1, #2
x3101	1111 0000 0010 0101	TRAP x25
x3102	0001 010 001 0 00001	ADD R2, R1, #1
x3103	0001 010 010 0 00010	ADD R2, R2, #2

Because of the instruction stored in [x3101] meas **STOP** the program, so the R2 will end up be **x1482**.

T3

1. LDR R2, R1, #0　　STR R2, R0, #0
2. MAR <- BaseSR + #0　　MDR <- Memory[MAR]
   MAR <- BaseDR + #0　　Memory[MAR] <- MDR

T4

x3000	1001 100 001 11111
x3001	0101 100 100 0 00010
x3002	1001 011 010 11111
x3003	0101 011 011 0 00001
x3004	1101 011 011 000 100

T5

The five addressing modes:

(1) Immediate Mode

(2) PC-Relative Mode

(3) Indirect Mode

(4) Base + Offset Mode

(5) Register Addressing Mode

ADD	Operate instruction	Register Addressing Mode; Immediate Mode
NOT	Operate instruction	Register Addressing Mode
LEA	Data movement instruction	Immediate Mode
LDR	Data movement instruction	Base + Offset Mode
JMP	Control instruction	Register Addressing Mode

T6

1. ADD R4, R5, #0
2. AND R3, R3, #0
3. NOT R1, R7
   ADD R1, R6, R1
4. LD R1, DATA
   ADD R1, R1, R1
   ST R1, DATA
5. ADD R1, R1, #0

T7

- For the JMP instruction, the LC-3 will first access the memory to read this JMP instruction, and then jump to BaseR to read the next instruction, wo it is **2** in total.
- For the ADD instruction, the LC-3 need to access the memoty only **once**.
- For the LDI instruction, the LC-3 need to access the memory **twice**.

T8

1. R6 = x70A4
2. No, I can't. Because if we want to use only one LEA instruction to load x70A4 to R6, we need PC = x3011(x3010 is used to store the LEA instruction) + $Offset$ = x70A4. But the biggest number that 9 bits Offset can represent is 255, which plus PC is far away from x70A4.

T9

Let's check the instructions:

x3000	0101 0000 0010 0000	AND R0, R0, #0
x3001	0101 1111 1110 0000	AND R7, R7, #0
x3002	0001 1100 0010 0001	ADD R6, R0, #1
x3003	0001 1101 1000 0110	ADD R6, R6, R6
x3004	0101 1001 0100 0110	AND R4, R5, R6
x3005	0000 0100 0000 0001	BRz #1
x3006	0001 0000 0010 0011	ADD R0, R0, #3
x3007	0001 1111 1110 0010	ADD R7, R7, #2
x3008	0001 0011 1111 0010	ADD R1, R7, #-14
x3009	0000 1001 1111 1001	BRn #-7
x300A	0101 1111 1110 0000	.HALT

Try to figure out how the program execute, I noticed that the value stored in R0 is related to 7 bits in R5 started from bit 1 to bit 7:

$$xxxx\,xxxx\,\mathbf{yyyyyyy}\,x$$

If one of those **y**s is 0, the value stored in R0 will add by 3. By knowing that at the beginning of the program execution, the value stored in R0 is 0, we can infer that the instruction stored in **x3006** has to be executed for 4 times. So we can infer that there are **only four** 0s in those **y**s. Apart from those **y**s, x could be either 1 or 0, it doesn't matter.

T10

x3000	1001 000 000 111111
x3001	0001 000 000 1 00001
x3002	0101 010 010 1 00000
x3003	0001 011 000 0 00 001
x3004	0000 010 00000 1100
x3005	0001 010 010 1 00001
x3006	0001 110 010 1 10000
x3007	0000 010 000000111
x3008	0101 001 001 1 11111
x3009	0000 100 000000010
x300A	0001 001 001 0 00 001
x300B	0000 111 111110111
x300C	0001 001 001 0 00 001
x300D	0001 001 001 1 00001
x300E	0000 111 111110100
x300F	0101 010 010 1 00000
x3010	0001 010 010 1 11111
x3011	0011 010 000001110
x3012	1111 0000 0010 0101