T1

We have $2^6 = 64$ opcodes,which means we need 6 bits to reprsent them.And for the registers,we need $2 \times \lceil \log_2 56 \rceil = 12$ bits.For the 32-bit instruction,we will have $32 - 12 - 6 = 14$ bits for the IMM,whic can reprsent the numbers range from -2^{13} to $2^{13} - 1$.That is:

$$-8192 \leqslant \text{IMM} \leqslant 8191$$

T2

X	Does the program halt?	Value stored in R0
000000010	Yes	2
000000001	Yes	3
000000000	Yes	6
111111111	No	-
111111110	Yes	-32764

T3

Only if the first instruction's result is **not Negtive** which means the result of [ADD R0, R1, R2](0001 000 001 0 00 010) is postive or zero:

$$\text{R}_0 = \text{R}_1 + \text{R}_2 \geqslant 0$$

T4

1. If we reduce the number of registers from 8 to 4, than we only need 2 bits to reprsent the registers. So for ADD(0001) and AND(0101) instructions, we can have a larger range of IMM representation; for the NOT(1001) instruction, there is no obvius benefits.
2. For LD(0010) and ST(0011), the reduction of registers will git us more bits to reprsent the OFFSETS, which means we can address a larger range.
3. For BR(0000) instruction, reducing the number of registers will not bring any obvius benefit, because there is no operation related to registers in the BR instruction.

T5

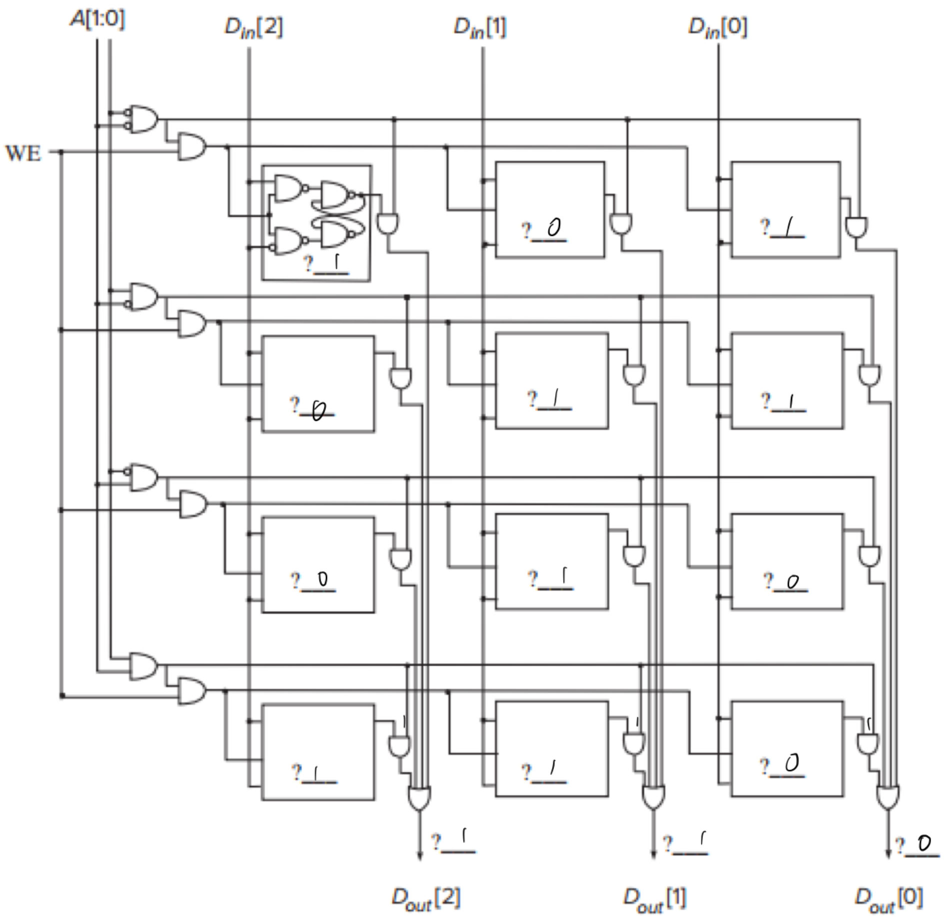

Figure 1: 8 cycle

T6

Operation No.	R/W	MAR	MDR
1	W	x4000	11110
2	R	x4003	10110
3	W	x4001	10110
4	R	x4002	01101
5	W	x4003	01101

Table 1: Operation on Memory

Address	Befor Access 1	After Access 3	After Access 5
x4000	01101	11110	11110
x4001	11010	10110	10110
x4002	01101	01101	01101
x4003	10110	10110	01101
x4004	11110	11110	11110

Table 2: Contents of Memory Locations

T7

1. $$nT = 1 \Rightarrow n = \frac{1}{5 \times 10^{-9}} = 2 \times 10^8 \text{ Cycles}$$
   .

2. $$N \cdot 8T = 1 \Rightarrow N = \frac{2 \times 10^8}{8} = 2.5 \times 10^7 \text{ instructions}$$

3. Using pipeline technique, we can process 1 complete instruction in the first 8 cycles, and then process another instruction in each subsequent cycle, so we can totally process:
   $$\left(2 \times 10^8 - 8\right) + 1 = 2 \times 10^8 - 7$$
   instructions.

T8

Address	instruction
x3000	1001 111 001 111111
x3001	1001 110 010 111111
x3002	0101 101 110 000 010
x3003	0101 100 001 000 110
x3004	1001 001 101 111111
x3005	1001 010 100 111111
x3006	0101 000 001 000 010
x3007	1001 011 000 111111

Table 3: XOR

T9

- 0001 010 001 1 00010 This is **not** a NOP instruction, because it changes the data of R2 to R1 + 2.
- 0000 111 000000000 This instruction **could be** working like a NOP instruction. This is a BR instruction but it does not change the original order of program execution.
- 0000 101 000000100 This is **not** a NOP instruction, because it will cause the program to directly execute the instructions after the fourth step unless the condition is 0.
- 1001 010 111 111111 This is **not** a NOP instruction, because it will do a NOT operation to R7 and store the result in R2.
- 1111 0000 00100011 This is **not** a NOP instruction, it will stop the program.

T10

The limitation of the BR instruction is that we cannot jump to an Address if it is too big for the number PC + OFFSETS. But the JMP instruction can jump to the address stored in a register. So theoretically, we can jump to anywhere we want.