Homework 4

2023.07.15 沈韵沨 3200104392

5.2 A memory's addressability is 64 bits. What does that tell you about the size of the MAR and MDR?

- The size of `MDR` should be 64 bit.
  The *addressability* represents the number of bits stored at each memory location and 64 bits addressability means each memory location is 64 bits long.
  As the MDR has the same size as the memory location, it should be 64 bits long.
- The size of `MAR` could not be determined.
  The size of MAR is related with the *address space* (the **total number** of memory locations). We can't determine its length from the given information.

5.4 Say we have a memory consisting of 256 locations, and each location contains 16 bits.

a. How many bits are required for the address?
b. If we use the PC-relative addressing mode, and want to allow control transfer between instructions 20 locations away, how many bits of a branch instruction are needed to specify the PC-relative offset?
c. If a control instruction is in location 3, what is the PC-relative offset of address 10? Assume that the control transfer instructions work the same way as in the LC-3.

a.

$$\because log_2(256) = 8$$
$$\therefore 8\ bits\ are\ required\ for\ the\ address$$

b. If we want to allow control transfer between instructions 20 locations away, then a branch instruction needs at least **6 bits** to specify PC-relative offset. The reasons are as follows:

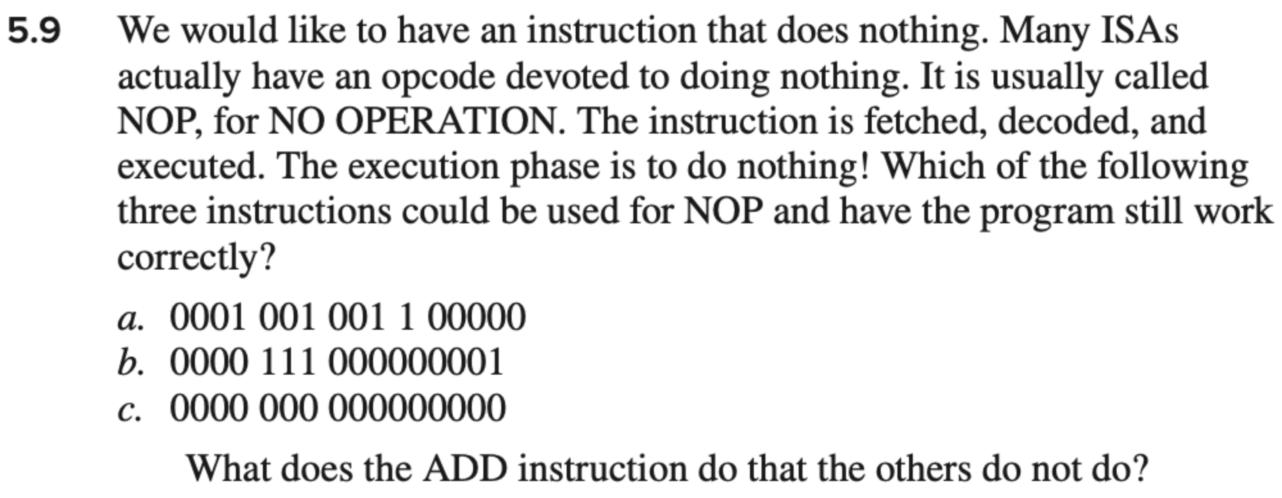
(1) PC-offsets shold be represented by 2's complement.

(2) Since PC has be incremented in FETCH phase, k bits offsets could transfer between $[-(2^{k-1}-1), +2^{k-1}]$ (relavated to current instruction).

$$According\ to\ (2),\ we\ have \begin{cases} 2^{k-1} \geq 20 \\ \left|-(2^{k-1}-1)\right| \geq 20 \end{cases}$$
$$\Rightarrow k \geq \lceil log_2(21) \rceil = 6$$

c. Since PC has be incremented in FETCH phase:

$$10 = PC + offset = (curLocation + 1) + offset$$
$$\Rightarrow offset = 10 - (curLocation + 1) = 10 - (3 + 1) = 6$$



5.9 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

What does the ADD instruction do that the others do not do?

Instruction (c) could be used for NOP, the reason are as follows:

- Instruction (a) means `ADD R1, R1, #0`. It doesn't change the value stored in R1, but would affect the condition code.
  => If we do nothing in EXECUTION phase, it might affect the result of some BR instruction after if.
- Instruction (b) means `BRnzp #1`, which means an **unconditionally** branch.
  => If we do nothing in EXECUTION phase, it would certainlly affect the control flow.
- Instruction (c) means `BR #0`.
  => Since every integer must be negative/zero/positive, the branch would **never taken**, thus this instruction could be used for NOP & the program still work correctly.

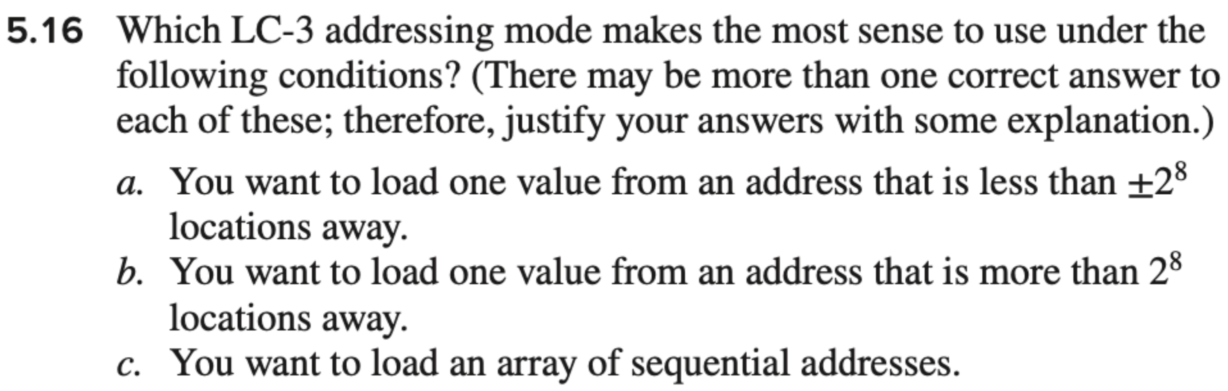
5.15 State the contents of R1, R2, R3, and R4 after the program starting at location x3100 halts.

Address	Data
0011 0001 0000 0000	1110 001 000100000
0011 0001 0000 0001	0010 010 000100000
0011 0001 0000 0010	1010 011 000100000
0011 0001 0000 0011	0110 100 010 000001
0011 0001 0000 0100	1111 0000 0010 0101
:	:
:	:
0011 0001 0010 0010	0100 0101 0110 0110
0011 0001 0010 0011	0100 0101 0110 0111
:	:
:	:
0100 0101 0110 0111	1010 1011 1100 1101
0100 0101 0110 1000	1111 1110 1101 0011

Address(Hex)	Instruction	Desc
`x3100`	`LEA R1, #32`	R1 = x3101 + x20 = x3121
`x3101`	`LD R2, #32`	R2 = M[x3102 + x20] = M[x3122] = x4566
`x3102`	`LDI R3, #32`	R3 = M[M[x3103 + x20]] = M[M[x3123]] = M[x4567] = xABCD
`x3103`	`LDR R4, R2, #1`	R4 = M[R2 + x1] = M[x4567] = xABCD
`x3104`	`TRAP #37`	**HALT**
...		
`x3122`	x4566	
`x3123`	X4567	
...		
`x4567`	xABCD	
`x4568`	xFED3	

According to the table above, the contents of each register after the program starting at `x3100` halts are:

Register	Content
R1	`x3121`
R2	`x4566`
R3	`xABCD`
R4	`xABCD`



5.16 Which LC-3 addressing mode makes the most sense to use under the following conditions? (There may be more than one correct answer to each of these; therefore, justify your answers with some explanation.)

a. You want to load one value from an address that is less than $\pm 2^8$ locations away.

b. You want to load one value from an address that is more than 2^8 locations away.

c. You want to load an array of sequential addresses.

a. `PC-Relative` Mode

- PC-Relative mode use 9 bits offset, which cover the range $[-(2^8-1), +2^8]$. Since the destination is less than $\pm 2^8$ away, this mode can satisfy the request.
- Although `Indirect` mode also works, the `PC-Relative` mode is faster since it only visit memory ONCE (`Indirect` mode need TWICE, 1 for address & 1 for data).

=> Thus, `PC-Relative` Mode is the most sense one under this situation.

b. `Indirect` Mode

Mode	Range
`Base + offset`	Base + $[-(2^5-1), +2^5]$
`PC-Relative`	curInstruction + $[-(2^8-1), +2^8]$
`Indirect`	Anywhere in the Memory

=> Since only the `Indirect` mode can satisfy the request, it must be the most sense one.

> In fact, `Base + offset` mode also allow

c. `Base + offset` Mode

- If the destination address is close, then `PC-Relative` mode is faster.

- Buf if the destination address is far away, `Base + offset` mode works better —— it can just modify offset in the instruction to visit those sequential address.
  `Indirect` mode requires us to modify the content of DR, which needs more instructions.