

COMPSCI 210

Tutorial 4 – The von Neumann Model, LC3

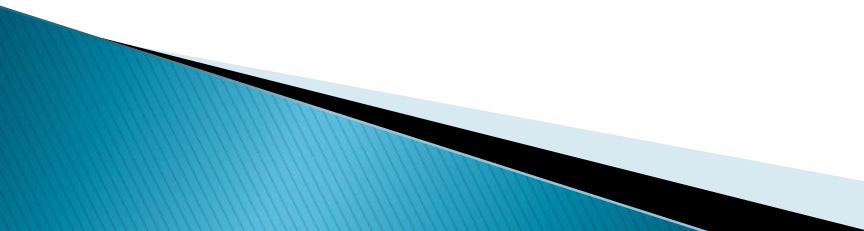


Exercises

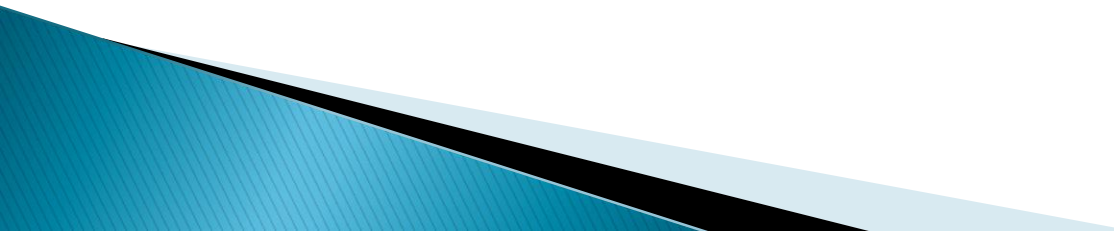
- ▶ 1) Remember that PC is incremented as part of the FETCH phase. This is done *before the EVALUATE ADDRESS stage*. *Why does this matter?*

- ▶ 1) Remember that PC is incremented as part of the FETCH phase. This is done *before the EVALUATE ADDRESS stage*. *Why does this matter?*
- ▶ Answer: To allow for ‘control instructions’ to alter the PC and change the sequence of execution.

- ▶ 4.4) What is the word length of a computer defined as? How does this length affect its computational power in relation to a computer with a larger word size?

- ▶ 4.4) What is the word length of a computer defined as? How does this length affect what a computer can compute in relation to a computer with a larger word size?
 - ▶ Answer: It is the size of the quantities processed by the ALU: 16–bits for the LC–3.
 - ▶ All computers have the same computational ability, the differences are with respect to time and memory requirements.
- 

- ▶ 4.6) What are the two components of an instruction? What information do these two components contain?

- ▶ 4.6) What are the two components of an instruction? What information do these two components contain?
 - ▶ Answer:
 - Opcode – what the instruction does.
 - Operand – what the instruction performs upon.
- 

- ▶ 4.8) Suppose a 32-bit instruction takes the following format:



- ▶ If there are 225 opcodes and 120 registers,
- ▶ a. What is the minimum number of bits required to represent the OPCODE?
- ▶ b. What is the minimum number of bits required to represent the Destination Register (DR)?
- ▶ c. What is the maximum number of UNUSED bits in the instruction encoding?

- ▶ 4.8) Suppose a 32-bit instruction takes the following format:



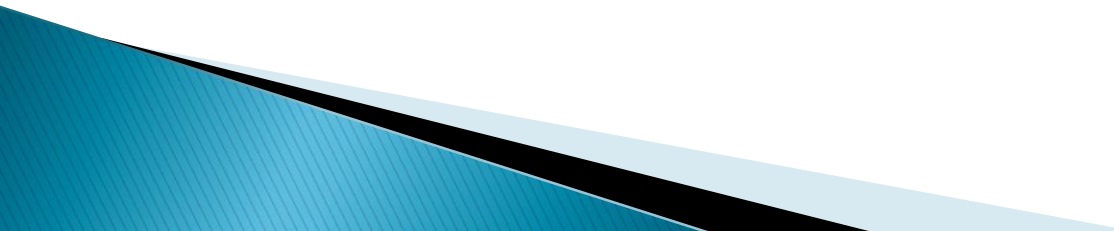
- ▶ If there are 225 opcodes and 120 registers,
- ▶ a. What is the minimum number of bits required to represent the OPCODE? A: 8 bits
- ▶ b. What is the minimum number of bits required to represent the Destination Register (DR)? A: 7 bits
- ▶ c. What is the maximum number of UNUSED bits in the instruction encoding? A: 3 bits

▶ 4.16)

- A. If a machine cycle is 2 nanoseconds (i.e., 2×10^{-9} seconds), how many machine cycles occur each second?
- B. If the computer requires on the average eight cycles to process each instruction, and the computer processes instructions one at a time from beginning to end, how many instructions can the computer process in 1 second?

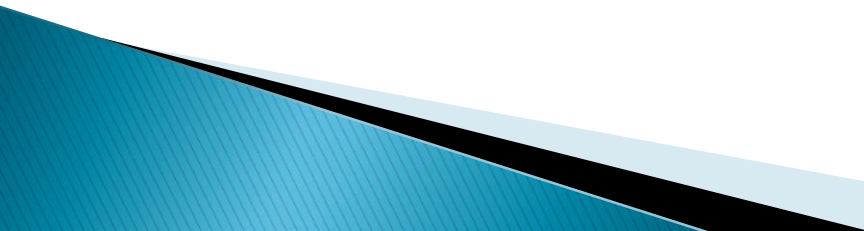
▶ 4.16)

- A. If a machine cycle is 2 nanoseconds (i.e., 2×10^{-9} seconds), how many machine cycles occur each second?
 - Answer: 5×10^8 cycles/sec
- B. If the computer requires on the average eight cycles to process each instruction, and the computer processes instructions one at a time from beginning to end, how many instructions can the computer process in 1 second?
 - Answer: $(5 \times 10^8) / 8 = 0.625 \times 10^8$

- ▶ 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.
- 

- ▶ 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?
 - Answer: 8 bits.
 - 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?
 - Answer: ± 20 gives a range of 40, therefore need 6 bits.
 - 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.
 - Answer: PC counter is incremented to 4, $10 - 4 = 6$.

- ▶ 5.8) We want to increase the number of registers we can specify in the LC-3 ADD instruction to 32. Do you see any problem with that? Explain.

- ▶ 5.8) We want to increase the number of registers we can specify in the LC-3 ADD instruction to 32. Do you see any problem with that? Explain.
 - ▶ Answer: We would need 5 bits to specify 32 registers. There would not be enough bits for the ADD instruction if we specify it in 16 bits. We would therefore need to rewrite the ISA and use a larger instruction size.
- 

- ▶ 5.12) After executing the following LC-3 instruction: `ADD R2, R0, R1`, we notice that `R0[15]` equals `R1[15]`, but is different from `R2[15]`. We are told that `R0` and `R1` contain UNSIGNED integers. Under what conditions can we trust the result in `R2`?

- ▶ 5.12) After executing the following LC-3 instruction: `ADD R2, R0, R1`, we notice that `R0[15]` equals `R1[15]`, but is different from `R2[15]`. We are told that `R0` and `R1` contain UNSIGNED integers. Under what conditions can we trust the result in `R2`?
- ▶ Answer: There is the potential for an overflow condition to occur if the result of the addition of `R0` and `R1` is larger than the available bits. We can only be sure the result is ok if the sum of `R0` and `R1` is less than or equal to 65,535. E.g. if we noticed `R0[15] = R1[15] = 0` then we would know the result would be ok, even when `r2[15] = 1`.

- ▶ 5.14) The LC-3 does not have an opcode for the logical function OR. However, we can write a sequence of instructions to implement OR. The four instruction sequence below performs the OR of the contents of register 1 and register 2 and puts the result in register 3. Fill in the two missing instructions so that the four instruction sequence will do the job:
 - ▶ (1): 1001 100 001 111111
 - ▶ (2):
 - ▶ (3): 0101 110 100 000 101
 - ▶ (4):

- ▶ 5.14) The LC-3 does not have an opcode for the logical function OR. However, we can write a sequence of instructions to implement OR. The four instruction sequence below performs the OR of the contents of register 1 and register 2 and puts the result in register 3. Fill in the two missing instructions so that the four instruction sequence will do the job:
 - ▶ Use DeMorgan's law:
 - ▶ Answer:
 - ▶ (1): 1001 100 001 111111 (NOT R4 R1)
 - ▶ (2): 1001 101 010 111111 (NOT R5 R2)
 - ▶ (3): 0101 110 100 000 101 (AND R6 R4 R5)
 - ▶ (4): 1001 011 110 111111 (NOT R3 R6)

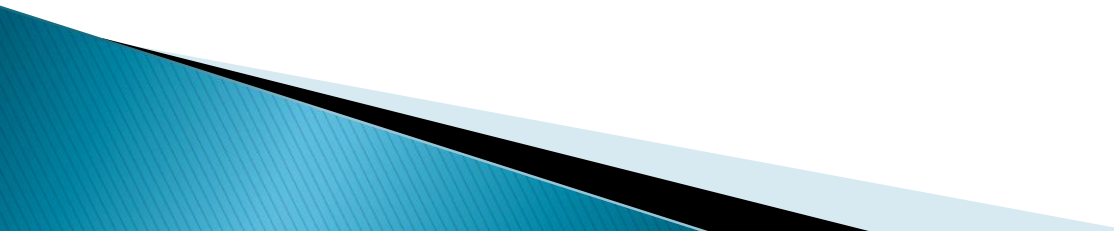
- ▶ 5.16) Which LC-3 addressing mode makes the most sense to use under the following conditions. (There may be more than one correct answer, justify.)
- ▶ A) You want to load one value from an address which is less than $\pm 2^8$ locations away.
- ▶ B) You want to load one value from an address which is more than $\pm 2^8$ locations away.
- ▶ C) You want to load an array of sequential addresses.

- ▶ 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, justify.)
- ▶ A) You want to load one value from an address which is less than $\pm 2^8$ locations away.
 - Answer: PC-relative addressing (9 bits for offset)
- ▶ B) You want to load one value from an address which is more than $\pm 2^8$ locations away.
 - Answer: Indirect (16 bits available)
- ▶ C) You want to load an array of sequential addresses.
 - Answer: PC-relative

- ▶ 5.20) If we made the LC-3 ISA such that we allow the LD instruction to load data only ± 32 locations away from the incremented PC value, how many bits would be required for the PC-relative offset in the LD instruction?

- ▶ 5.20) If we made the LC-3 ISA such that we allow the LD instruction to load data only ± 32 locations away from the incremented PC value, how many bits would be required for the PC-relative offset in the LD instruction?
- ▶ Answer: $\pm 32 = 65$ states \Rightarrow need 7 bits

- ▶ 5.22) The PC contains x3010. The following memory locations contain values as shown:
 - x3050: x70A4
 - x70A2: x70A3
 - x70A3: xFFFF
 - x70A4: x123B
- ▶ The following three LC-3 instructions are then executed, causing a value to be loaded into R6.
- ▶ ->

- ▶ ...what is that value?
 - ▶ x3010 1110 0110 0011 1111
 - ▶ x3011 0110 1000 1100 0000
 - ▶ x3012 0110 1101 0000 0000
-
- ▶ We could replace this three instruction sequence with a single instruction. What is it?
- 

▶ 5.22) Answer:

LEA R3 x3F, PC = x3011, therefore R3 = x3050

LDR R4 R3 0, therefore R4 = x70A4

LDR R6 R4 0, therefore R6 = x123B

Could replace these with LDI R6 000111111,
which takes the value at the address stored in
memory location x3050 (x3011 + x003F)

- ▶ 5.28) It is the case that we REALLY don't need to have load indirect (1010) and store indirect (1011) instructions. We can accomplish the same results using other instruction sequences instead of using these instructions. Replace the store indirect (1011) instruction in the code below with whatever instructions are necessary to perform the same function:

- ▶ x3000 0010 0000 0000 0010
- ▶ x3001 1011 0000 0000 0010
- ▶ x3002 1111 0000 0010 0101
- ▶ x3003 0000 0000 0100 1000
- ▶ x3004 1111 0011 1111 1111

- ▶ It is the case that we REALLY don't need to have load indirect (1010) and store indirect (1011) instructions. We can accomplish the same results using other instruction sequences instead of using these instructions. Replace the store indirect (1011) instruction in the code below with whatever instructions are necessary to perform the same function:
 - ▶ x3000 0010 0000 0000 0010 (LD R0 "x3001"+2)
 - ▶ x3001 1011 0000 0000 0010 (STI R0 "x3002"+2)
 - ▶ x3002 1111 0000 0010 0101
 - ▶ x3003 0000 0000 0100 1000
 - ▶ x3004 1111 0011 1111 1111
 - Answer:
 - Can replace this with:
 - x3001 1110 001 000000011 (LEA R1 3 (its 3 instead of 2 since we have an extra instruction, so we want to write over the memory address with the same contents as in the original code))
 - x3002 0111 000 001 000000 (STR R0 R1 0)