F.4 Chapter 4 Solutions

4.1 Components of the Von Neumann Model:

(a) Memory: Storage of information (data/program)

(b) Processing Unit: Computation/Processing of Information

(c) Input: Means of getting information into the computer. e.g. keyboard, mouse

(d) Output: Means of getting information out of the computer. e.g. printer, monitor

(e) Control Unit: Makes sure that all the other parts perform their tasks correctly and at the correct time.

4.2 The communication between memory and processing unit consists of two registers: Memory Address Register (MAR) and Memory Data Register (MDR).

- To read, the address of the location is put in MAR and the memory is enabled for a read. The value is put in MDR by the memory.
- To write, the address of the location is put MAR, the data is put in MDR and the Write Enable signal is asserted. The value in MDR is written to the location specified.

4.3 The program counter does not maintain a count of any sort. The value stored in the program counter is the address of the next instruction to be processed. Hence the name 'Instruction Pointer'is more appropriate for it.

4.4 The size of the quantities normally processed by the ALU is referred to as the word length of the computer.

The word length does not affect what a computer can compute. A computer with a smaller word length can do the same computation as one with a larger word length; but it will take more time.

For example, to add two 64 bit numbers,

```
word length = 16 takes 4 adds.
word length = 32 takes 2 adds.
word length = 64 takes 1 add.
```

4.5 (a) Location 3 contains 0000 0000 0000 0000
Location 6 contains 1111 1110 1101 0011

(b) i. Two's Complement -
Location 0: 0001 1110 0100 0011 = 7747
Location 1: 1111 0000 0010 0101 = -4059

ii. ASCII - Location 4: 0000 0000 0110 0101 = 101 = 'e'

iii. Floating Point -
Locations 6 and 7: 0000 0110 1101 1001 1111 1110 1101 0011
Number represented is 1.10110011111111011010011 x 2^{-114}

iv. Unsigned -
Location 0: 0001 1110 0100 0011 = 7747
Location 1: 1111 0000 0010 0101 = 61477

(c) Instruction - Location 0: 0001 1110 0100 0011 = Add R7 R1 R3

(d) Memory Address - Location 5: 0000 0000 0000 0110 Refers to location 6. Value stored in location 6 is 1111 1110 1101 0011

4.6 The two components of and instruction are:

Opcode: Identifies what the instruction does.

Operands: Specifies the values on which the instruction operates.

4.7 60 opcodes = 6 bits
32 registers = 5 bits
So number of bits required for IMM = 32 - 6 - 5 - 5 = 16
Since IMM is a 2's complement value, its range is -2^{15} ... $(2^{15} - 1)$ = -32768 .. 32767.

4.8 a) 8-bits
b) 7-bits
c) Maximum number of unused bits = 3-bits

4.9 The second important operation performed during the FETCH phase is the loading of the address of the next instruction into the program counter.

4.10 Refer to the following table:

	Fetch Instruction	Decode	Evaluate Address	Fetch Data	Execute	Store Result
PC	0001, 0110, 1100				1100	
IR	0001, 0110, 1100					
MAR	0001, 0110, 1100			0110		
MDR	0001, 0110, 1100			0110		

4.11 The phases of the instruction cycle are:

(a) Fetch: Get instruction from memory. Load address of next instruction in the Program Counter.

(b) Decode: Find out what the instruction does.

(c) Evaluate Address: Calculate address of the memory location that is needed to process the instruction.

(d) Fetch Operands: Get the source operands (either from memory or register file).

(e) Execute: Perform the execution of the instruction.

(f) Store Result: Store the result of the execution to the specified destination.

4.12 Considering the LC3 instruction formats

ADD

Fetch: Get instruction from memory. Load next address into PC. Decode: It is here that it is determined that the instruction is an add instruction. Evaluate Address: No memory operation so NOT REQUIRED. Fetch Operands: Get operands from register file. Execute: Perform the add operation. Store Result: Store result in the register file.

LDR

Fetch: Get instruction from memory. Load next address into PC. Decode: It is here that it is determined that the instruction is a Load Base+offset instruction. Evaluate Address: Calculate the memory address by adding the Base register with the sign extended offset.

Fetch Operands: Get value from the memory. Execute: No operation needed so NOT REQUIRED. Store Result: Store the value loaded into the register file.

JMP

Fetch: Get instruction from memory. Load next address into PC. Decode: It is here that it is determined that the instruction is a Jump instruction. Evaluate Address: No memory operation so NOT REQUIRED. Fetch Operands: Get the base register from register file. Execute: Store the value in PC. Store Result: NOT REQUIRED.

** Since we are considering a non pipelined implementation, the instruction phases where no operation is performed may not be present in its execution cycle.

4.13

	F	D	EA	FO	E	SR	
x86: ADD [eax] edx	100	1	1	100	1	100	= 303
LC3: ADD R6, R2, R6	100	1	-	1	1	1	= 104

4.14 JMP: 1100 0000 1100 0000

Fetch: Get instruction from memory. Load next address into PC.

Decode: It is here that it is determined that the instruction is JMP.

Evaluate Address: No memory operation, so NOT required.

Fetch Operands: Get the base register from the register file.

Execute: Load PC with the base register value, x369C.

4.15 Once the RUN latch is cleared, the clock stops, so no instructions can be processed. Thus, no instruction can be used to set the RUN latch. In order to re-initiate the instruction cycle, an external input must be applied. This can be in the form of an interrupt signal or a front panel switch, for example.

4.16 (a) $1/(2 * 10^{-9}) = 5 * 10^8$ machine cycles per second.

(b) $5 * 10^8/8 = 6.25 * 10^7$ instructions per second.

(c) It should be noted that once the first instruction reaches the last phase of the instruction, an instruction will be completed every cycle. So, except for this initial delay (known as latency), one instruction will be completed each machine cycle (assuming that there are

no breaks in the sequential flow). If we ignore the latency, the number of instructions that will be executed each second is same as the number of machine cycles in a second = 5 * 10^8.