Joseph Hyatt
CSE 3100-01/02
007131989

Homework 2

Chapter (4): 4.5 b; 4.9; 4.11

**4.5) b) The binary value within each location can be interpreted in many ways. We have seen that binary values can represent unsigned numbers, 2's complement signed numbers, floating-point numbers,
and so forth.**

(1) Interpret location 0 and location 1 as 2's complement integers.

Location 0 = (0001 1110 0100 0001)
To interpret this as 2’s complement integer, you must see weather it is positive or negative.

If MSB is 0, it’s positive and should be taken as it is.

If MSB is 1, it’s negative and we complement the number and add 1 to the result, and finally convert it to a decimal.

Since the above number is positive, we will take it as it is.
0001 1110 0100 0001 (binary)
7747 (decimal)

At location 1
0001 ---- 1111 0000 0010 0101
Since the above number is negative, we will complement the number and will add 1 to it.

```
1111 0000 0010 0101
0000 1111 1101 1010
                 +1
---------------------------
0000 1111 1101 1010 → 4059 (decimal)
```

(2) Interpret location 4 as an ASCII value.

At location (4)
To convert binary into ASCII, we split the binary number into 8 bits and convert into a decimal, and then we look at ASCII value for their decimal values

(Binary)

0000 0000	0110 0101
0	101

(Decimal)

0 ---> NUL
101 --- e

ASCII value is

NUL e

(3) Interpret locations 6 and 7 as an IEEE floating point number. Location 6 contains number [15:0]. Location 7 contains number [31:16].

At location (6)
0110 ---- 1111 1110 1101 0011 - (15:0)

At location (7)
0111 ---- 0000 0110 1101 1001 - (31:16)

Total number (32 bits)

0	000 0110 1	101 1001 1111 1110 1101 0011

Sign bit ---- 0 (positive number)
Exponent ---- 13 - 127 = -114
Fraition ----

$$1 + \left(1 \cdot 2^{-1}\right) + \left(1 \cdot 2^{-3}\right) + \left(1 \cdot 2^{-4}\right) + \left(1 \cdot 2^{-7}\right) + \left(1 \cdot 2^{-8}\right) + \left(1 \cdot 2^{-9}\right)$$
$$+ \left(1 \cdot 2^{-10}\right) + \left(1 \cdot 2^{-11}\right) + \left(1 \cdot 2^{-12}\right) + \left(1 \cdot 2^{-13}\right) + \left(1 \cdot 2^{-14}\right) + \left(1 \cdot 2^{-16}\right)$$
$$+ \left(1 \cdot 2^{-17}\right) + \left(1 \cdot 2^{-19}\right) + \left(1 \cdot 2^{-22}\right) + \left(1 \cdot 2^{-23}\right)$$
$$\approx 1.6953$$

(4) Interpret

$$\textit{Floating point number} = 1.6953\, X\, 2^{-114}$$

location 0 and location 1 as unsigned integers.

0000 ---- 0001 1110 0100 0011 ---- 7747 (decimal)
0001 ---- 1111 0000 0010 0101 ---- 61477

4.9) The FETCH phase of the instruction cycle does two important things. One is that it loads the instruction to be processed next into the IR. What is the other important thing?

The second most important operation is to load the MAR with the contents of the PC, and at the same time, increase the PC by 1. During the FETCH phase is the loading of the address of the next instruction into the progress counter.

4.11) State the phases of the instruction cycle and briefly describe what operations occur in each phase.

In the FETCH phase:

- First, load the memory address register (MAR) with the contents of the program counter (PC) register. Simultaneously, increment the PC register so that it will point to the next instruction.
- Next, FETCH the actual instruction memory from the address given in MAR and place it into the memory data register (MDR).
- In the end, load the instruction register (IR) with the contents of MDR.

In the DECODE phase:

- Examine the instruction to find out what the processor is being asked to do.
- It decides what opcode needs to be processed among its opcodes.

In EVALUATE ADDRESS phase:

- Compute the address of the memory location required to process this instruction.
- Note that, for some instruction, the address can be fetched directly but for others like load register (LDR) use add the offset to the base to calculate this value.

In FETCH OPERAND phase:

- Obtain the source operands that will be required to process this instruction.
- Note that the operands can either in memory or a register file.
- For some instructions like LDR, this phase involves just two steps and for some, it can be more.
- Performing some of these steps simultaneously can speed up the execution of an instruction.

- Once the opcode and operands are available, the instruction can be executed.
- In the execute phase, the actual execution of the instruction is carried out in the arithmetic and logic unit (ALU).
- Again, it might involve a single step for example, in ADD opcode. It might involve multiple staples, depending on the opcode.

In the STORE RESULT phase:

- Store the result of the execution of the instruction in the designated destination.
- This destination can be a memory location or a register.
- Note that this is the last phase of the instruction cycle.

Chapter (5): 5.1; 5.4; 5.15

5.1) Given instructions ADD, JMP, LEA, and NOT, identify whether the instructions are operating instructions, data movement instructions, or control instructions. For each instruction, list the addressing modes that can be used with the instruction.

The instruction “ADD” is a operate instruction. It’s used to perform instructions process data. The ADD instruction is used to the immediate addressing mode and register addressing mode. In the immediate addressing mode, the operation code in the ADD instruction is immediately followed by the value to be referenced. In the Register addressing mode, a register is used to specify the operand instead of memory location.

The instruction “JMP” is a control instruction. It’s used to perform instructions change that is executed. The JMP instruction is used in the register addressing mode. In the Register addressing mode, the address field contains a register reference the cause of a register address location.

The instruction “LEA” is a data movement instruction. It’s used to transfer data between the memory and general registers and between the registers and the input/output devices. The LEA instructions are used in the immediate addressing mode. In the immediate addressing mode, the data on which the operation is to be performed contained in the instruction to move.

The instruction “NOT” is a operate instruction. It’s used to perform a unary operation. The NOT instruction is used in the register addressing mode. In the Register addressing mode, a minor address instruction field is needed in the NOT instruction.

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?

Given Data:
Memory locations = 256
Each location contains 16 bits.
The address contains bits = $\log_2(256)$

= 8 (8 bits are required for the address)

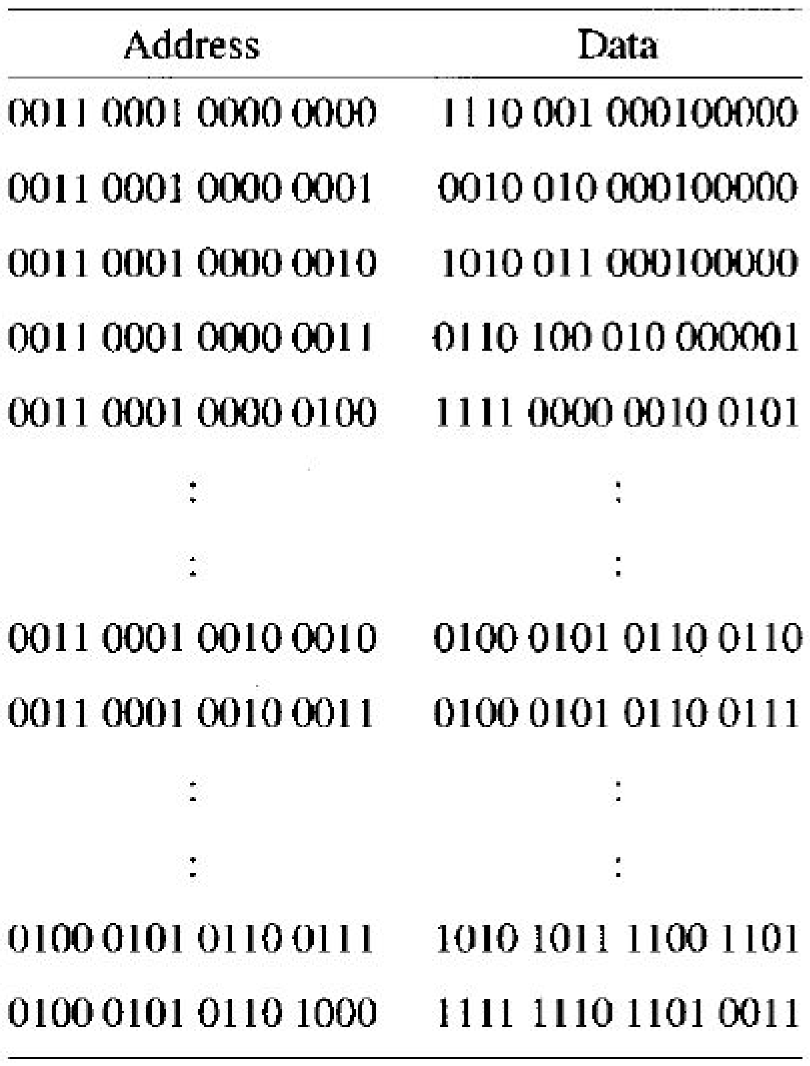
b. If we use the PC-relative addressing mode, and want to allow control transfer between instructions 20 locations away, how many bits of a branch instruction are needed to specify the PC-relative offset?

- The offsets of the PC use 2's compliment representation, which require a minimum of 6 bits.
- The 2's complement representation of the address allows +22 or -21 locations of the LOAD (LD) or Store (ST) instruction because, the PC increments its value before adding the offset. The number of branch instructions is needed to require the CO relative offset is 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.

Given Data:
Old PC relative offset address = 10
Control instruction location = 3
Assuming the control transfer instruction work the same way as in the LC-3, the PC value after incrementing = 3 + 1 = 4
To calculate the PC relative offset of the address is calculated as follows:
New PC - relative offset address = (Old PC - relative offset address) - (Pc value)
= 10 -4
= 6
The PC relative offset address is 6

5.15) State the contents of Rl, 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

Location Address	Data	Contents
x3100	1110 0010 0010 0000	MOVS R0, #0xE2
X3101	0010 0100 0010 0000	MOVS R0, #0x24
X3102	1010 0110 0010 0000	MOVS R0, #0xA6
X3103	0110 1000 1000 0001	STRH R0, [R5, #0xA]
X3104	1111 0000 0010 0101	MOVS R5, #0xF0

The MOVS instruction moves the values to a particular register. The STRH stores the values to a register-based on operations. The immediate values are provided in the hexadecimal number. The next instruction is to halt and shows the contents of all registers.