The von Neumann Model

Abstract

To get a task done by a computer, we need 2 things

- a **computer program**
  A computer program consists of a set of *instructions*, each specifying a well-defined piece of work for the computer to carry out.
- the **computer**
  The von Neumann model consists of 5 parts: *memory, a processing unit, input, output and a control unit*.

1 Basic Components of Computer

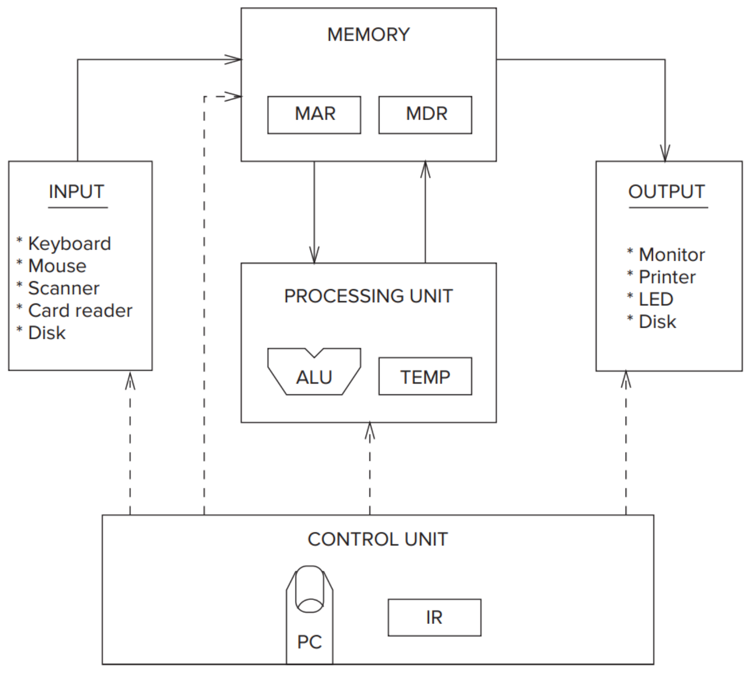


Figure 4.1 The von Neumann model, overall block diagram.

1.1 Memory

Memory is used to store information. Every memory location has an address asscoiated, and store a value in that location. ***e.g.*** the program code

You can turn to Chapter3 to review concepts of address space and addressability, here we just skip them. *In the LC-3, the memory address space is* 2^{16} *and the addressability is 16 bits.*

In order to uniquely reprensent 2^k items, we need at least k bits. In practice, "**byte**" refers to 8 bits and we have 1KB=2^{10}bytes, 1MB=2^{20}bytes, 1GB=2^{30}bytes.

To *access memory* (read or write), we will use 2 registers, **MAR (Memory's Address Register)** and **MDR (Memory's Data Register)**. The process is as follows:

- Read from the memory
  - Load the address you want read to MAR, then interrogate the memory.
  - The information stored in the location having that address will be placed in the MDR
- Write to the memory
  - Load the address you want write to MAR, and the value to be stored in the MDR, then interrogate the memory.

- The information contained in the MDR will be written into the memory location whose address is in the MAR.

1.2 Processing Unit

Processing unit is the actual unit that *carries out the processing of information in the computer*.

Processing unit can be complex, but now we only focus on the simple case. The processing unit consists of 2 parts: **ALU (Arithmetic Logic Unit)** and **Register Files**.

1.2.1 ALU

As the name implie, ALU is for arithmetic and logic operations. ***e.g.*** ADD, AND...

ALU normally proccess data element of a fixed size referred to as the word length of the computer. The data elements are called **words**. Different computer may have different word lengths, the most common nowadays being 64 bits.

In the LC-3, the ALU processes 16-bit words, so it has a word length of 16 bits.

1.2.2 Register Files

As you can see, the processing unit and the memory are not put together, which means if we want access memory when we do arithmetic or logic operation it will cost a lot of time. So we need a space that is located inside the processing unit to store the data that are being processed. It is called register files.

Typically, the size of each register is identical to the size of values processed by the ALU.

In the LC-3, the register files consists of 8 registers $(R0, R1, \dots R7)$, each of which is 16 bits long. Such 8 registers are also called **GPR (General Purpose Registers)**.

1.3 Input and Output

Some devices that get input from users, and some display the result to users. The method of communication between I/O (Input/Output) devices and CPU is covered in Chapter 9.

1.4 Control Unit

Control unit is in charge of making all the other parts of the computer play together. When we describe the step-by-step process of executing a computer program, it is the control unit that keeps track of both *where we are within the process of executing the program* and *where we are in the process of executing each instruction*.

Control unit in LC-3 consists of

- **program counter (PC)**
  PC contains the address of the next instruction.
- **instruction register (IR)**
  IR contains the current instruction.
- FSM
  FSM determines the control signals that are sent to the other parts of the computer. ***e.g.*** For ALU to do ADD, we need to send a signal to ALU to tell it to do ADD operation.

2 Instruction Processing

2.1 The Instruction

The **instruction** is the *smallest piece* of work specified in a computer program. It is made up of 2 parts, the **opcode** (*what the instruction does*) and the **operands** (*who it does it to!*)

There are fundamentally 3 kinds of instructions:

- **operation instructions**: operate on data. ***e.g.*** ADD, AND, NOT
- **data movement**: move information from the processing unit to and from memory and to and from input/output devices. ***e.g.*** load, store
- **control**: altering the sequential processing of instructions. ***e.g.*** BR

2.2 Instruction Example of the LC-3

Here we provide som instruction examples of LC-3. You can turn to the appendix of the textbook for the complete instruction set. If you can not completely understand themnow, don't worry, details of the instruction set will be covered in Chapter 5.

2.2.1 ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0
ADD				R6			R2						R6		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	1	0	0	1	1	0
ADD				R6			R2				imm				

DR <- SR1 ADD SR2 / DR <- SR1 ADD imm5

- bit[15:12] : opcode, 0001 is the opcode of ADD.
- bit[11:9] : DR (Destination Register)
- bit[8:6] : SR1 (Source Register 1)
- bit[5] :
  - bit[5]=0 , the form is SR1 + SR2 -> DR then bit[4:3] is just used to fill the space. bits[3:0] : SR2
  - bit[5]=1 , then form is SR1 + imm5 -> DR then bit[5:0] will be sign-extended to 16 bits as the second operand.

e.g. ADD R2, R1, R0 ADD R2, R1, #1 Here R2 is DR and R1 is SR1.

2.2.2 AND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0
AND				R2			R3				imm				

DR <- SR1 AND SR2 / DR <- SR1 AND imm5
AND is the same as ADD except that its opcode is 0101.
We can set bit[5]=1 and set bit[4:0]=0 , and let SR1=DR . Then the instruction becomes reg AND 0 -> reg , we refer to this technique as *initializeing register to 0*.

e.g. AND R2, R1, R0 AND R2, R1, #1 Here R2 is DR and R1 is SR1.

2.2.3 LD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	0	0	1	1	0	0	0	1	1	0
LD				R2			198								

LD represent load, which means go to a particular memory location, read the value and store it in one of the registers.

LD takes **PC+offset** as its addressing mode, which we will introduce in detail later.

- `bit[15:12]` : opcode, 0010 is the opcode of LD
- `bit[11:9]` : DR, stored the value that will be read from the memory.
- `bit[8:0]` : offset, and it also will be *sign-extended to 16 bits.* Then PC+offset will be the source address. Eventually, `M[PC+offset] -> DR`.

e.g. `LD R2, x3050` Here R2 is DR and 0x3050 is the address that we want to read. (`PC+offset=0x3050`)

2.3 The Instruction Cycle

Instructions are processed under the direction of the control unit in a very systematic, step-by-step manner. The entire sequence of steps needed to process an instruction is called the **instruction cycle**. It consists of 6 sequential phases: fetch, decode, evaluate address, fetch operands, execute, store result.

Note that *the instruction cycle is not the clock cycle* rather made up by it. And *not all instructions need all 6 phases*.

2.3.1 Fetch

Obtain the next instruction from memory and load it into IR (instruction register). The fetch process is as follows:

- Load the MAR with the contents of the PC, and *simultaneously increment* the PC. (in one clock cycle)
- Interrogate memory, resulting in the instructio being placed int the MDR. (at least in one clock cycle)
- Load the IR with the contents of the MDR. (in one clock cycle)

2.3.2 Decode

Examine the instruction in order to figure out what the microarchitecture is being asked to do.(by decoders)

In the LC-3, $IR[15:12]$ will be decoded.

2.3.3 Evaluate Address

Computes the address of the memory location that is needed to process the instruction.

This phase only happens when we go to memory to specify where we will store in or where we will read from. ***e.g.*** LD has this phase.

2.3.4 Fetch Operands

Obtain the source operands including registers and memory.

e.g.

- In the LD example, this phase consists of loading MAR with the address calculated in the EVALUATE ADDRESS phase and reading.
- In the ADD example, this phase consists of loading registers with the contents from register files.

2.3.5 Execute

Carries out the execution of the instruction. ***e.g.*** In the ADD example we will add two values.

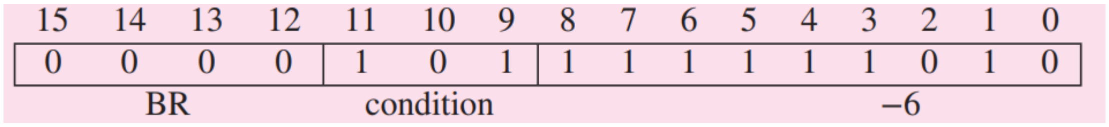
2.3.6 Store Result

Writes the result to its designated destinations. Usually this action is performed during the EXECUTE phase.

In many computers, including LC-3, an ADD instruction can fetch its source operands, perform the ADD in ALU and store the result all in a single clock cycle.

2.4 Changing the Sequence of Execution

In the LC-3, we use the instruction BR to change the sequence of execution.



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	1	1	1	1	0	1	0
BR				condition			−6								

BR (Conditional Branch) may change the contents of the PC.
BR also uses **PC+offset** as its addressing mode.

- `bit[15:12]` : opcode, 0000 is the opcode of BR.
- `bit[11:9]` : condition, is used to decide whehter it will change PC or not, the detail will be cover later.
- `bit[8:0]` : offset, how to calculate the address is the same as that in LD instruction.

Condition code(CC): three single-bit registers.(**N for Negative, Z for Zero, P for Positive**) Individually set or cleared each time one of the 8 GPR is written into as a result of execution of one of the operate instructions or one of the load instructions. ***e.g.*** After ADD instruction, if the result is postive then we will set P and clear N, Z.
Which instructions will modify the value of CC? See instruction set table where there is a `'+'` for those instructions or find the answer from the FSM.

3 Overview of the LC-3 Datapath

Here is an overview of the LC-3 datapath. You can turn to P704 Figure C.3 to see the complete LC-3 datapath. (Details will be discussed in Chapter 5)

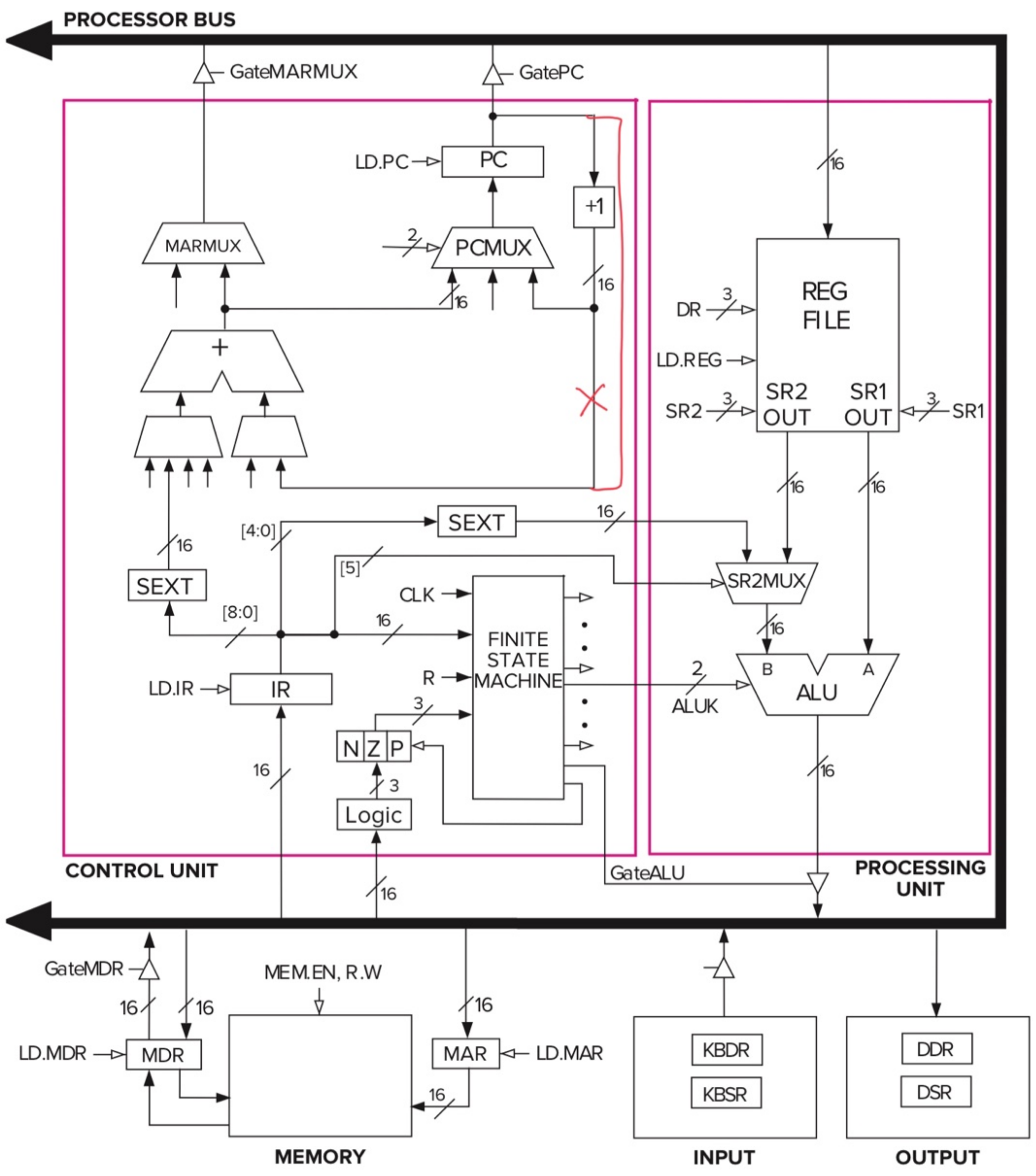


Figure 4.3 The LC-3 as an example of the von Neumann model.

It's woth noting that the in the diagram above, *filled-in arrowheads* denote *data elements* that flow along the correspounding paths. *Not-filled-in arrowheads* denote *control signals* that control the processing of the data elements.

3.1 LC-3 As an Example of The von Neumann Model

According to datapath above, we can find 5 parts of the von Neumamm model.

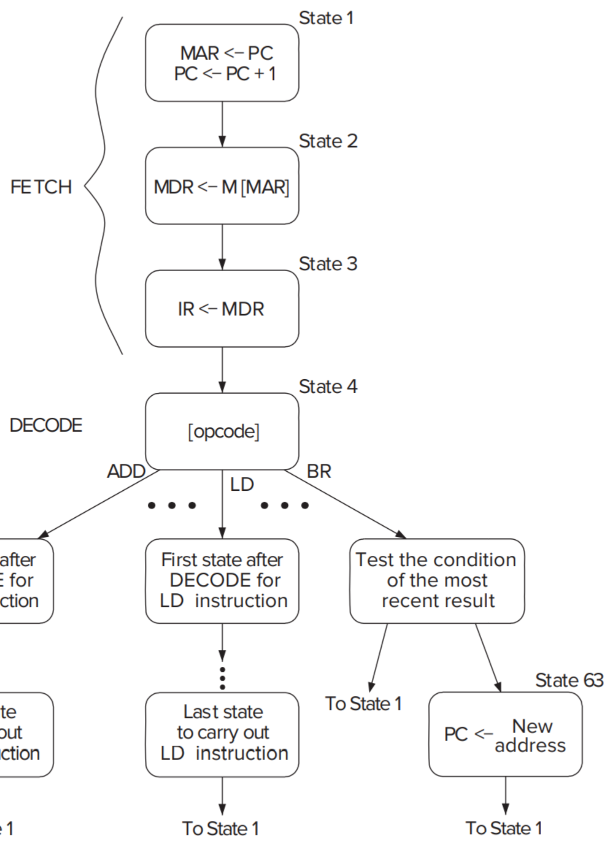
- I/O
- Memory
- Processing Unit: contains ALU and Register Files

- Control Unit: FSM, PC and IR.
  All the control signals are generated by the FSM.

3.2 Control of the Instruction Cycle

The instruction cycle is controlled by a synchrounous FSM.

Here we use an abbreviated state diagram of the LC-3. You can turn to P702 Figure C.2 to see the complete state diagram.



State 1
MAR <- PC
PC <- PC + 1
FETCH
State 2
MDR <- M [MAR]
State 3
IR <- MDR
State 4
DECODE
[opcode]
ADD
LD
BR
First state after DECODE for ADD instruction
First state after DECODE for LD instruction
Test the condition of the most recent result
To State 1
State 63
Last state to carry out ADD instruction
Last state to carry out LD instruction
PC <- New address
To State 1
To State 1
To State 1

- Fetch
  - MAR is loaded with the content of PC and PC simultaneously increment. (GatePC, LD.MAR, LD.PC)
  - MDR get the data from memory
  - The instruction is transferred from MDR to IR. (GateMDR, LD.IR)