The Architecture of Computer Hardware and System Software An Information Technology Approach
by Irv Englander

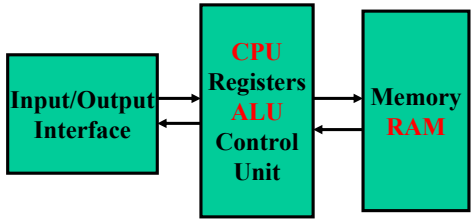
Chapter 7: The CPU and Memory

Outline

- The Components of the CPU
- The Concept of Registers
- The Memory Unit
- The Computer Operating Cycle (Fetch-Execute Instruction Cycle)
- Buses

Computer Hardware Components

Hardware: Input/Output, Memory, Central Processing Unit (CPU or Microprocessor)



System Block Diagram

7.1 The Components of the CPU

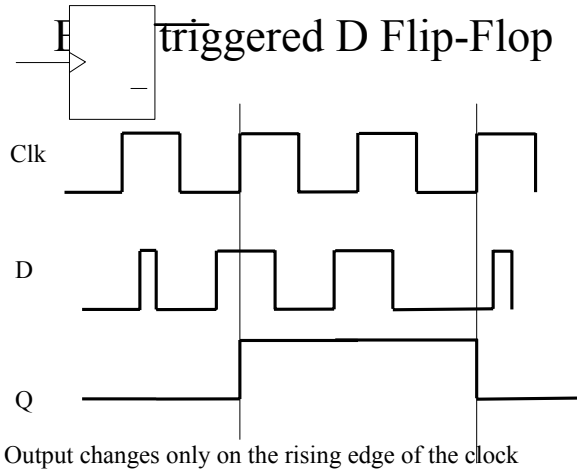
- CPU Consists of Several Integrated Circuits (ICs). For a Single Chip, CPU is Called: Microprocessor
- CPU Consists of
  - Set of Registers, SRs, which are used to store instructions, operands, and the result of the ALU
  - Arithmetic Logic Unit, ALU, which does all the arithmetic and most of the logical operations on operands
  - Control Unit, CU, which sequences the operation of the computer (Computer Cycle).

7.2 The Concept of Registers

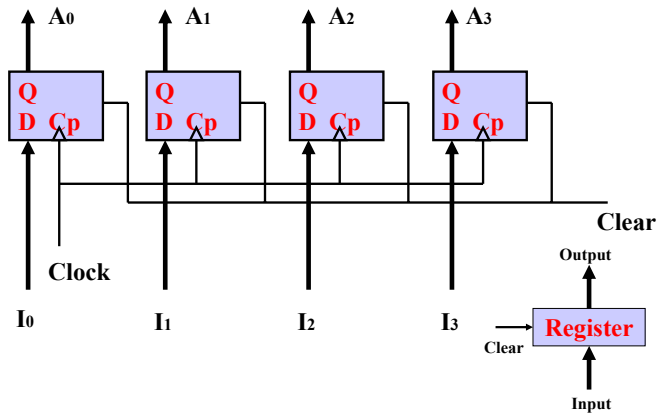
- A Register is a single, permanent storage location within the CPU
- Registers may be small as a single bit or wide as several bytes
- Registers are used to hold data, instructions, I/O address, or special binary codes
- Registers inside the CPU are called: General-Purposes Registers
- There are some specific registers called:
  - Program Counter (PC)
  - Instruction Register (IR)
  - Memory Address Register (MAR)
  - Memory Data Register (MDR) (or MBR)
  - Status Register (ST)
  - I/O Registers

Operations on Registers

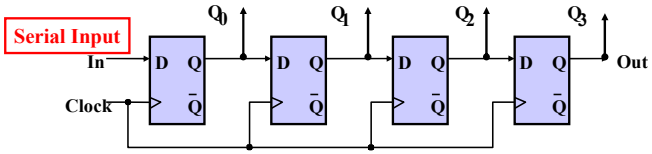
- Data can be loaded from register or memory into a register
- Data in register can be added/subtracted to/from another register
- Data in register can be shifted/rotated right/left by one or more bits
- Data in register can be cleared, set, complemented or incremented/decremented by one



Register is A Group of Flip-Flops

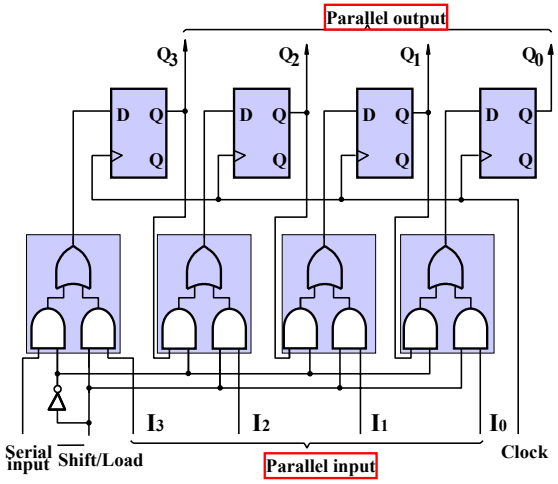


Simple Shift Register

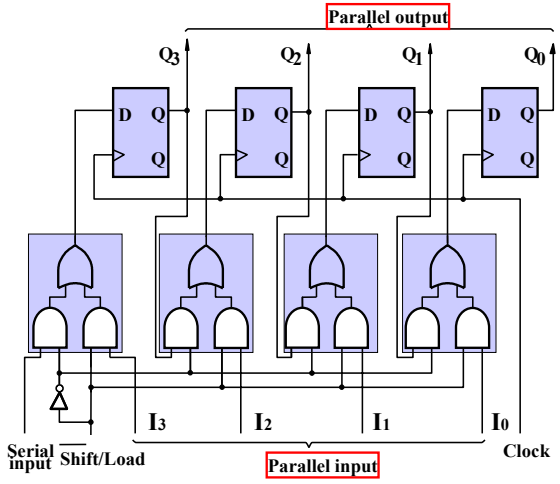


	In	Q_0	Q_1	Q_2	Q_3 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

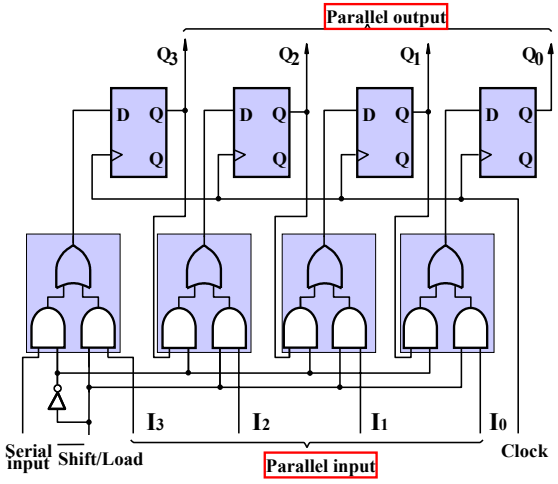
Shift Register with Parallel Load



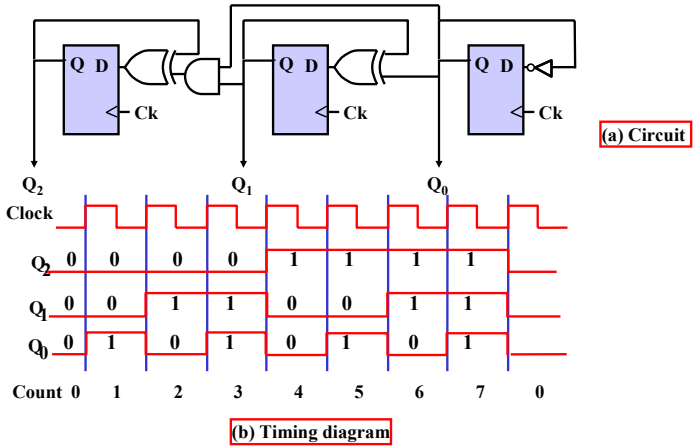
Parallel Load



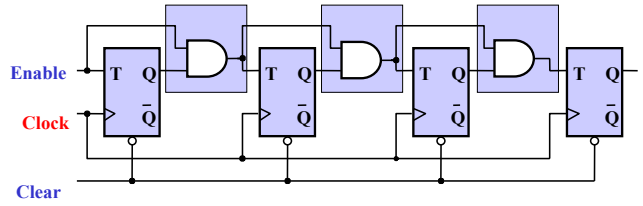
Shift Register



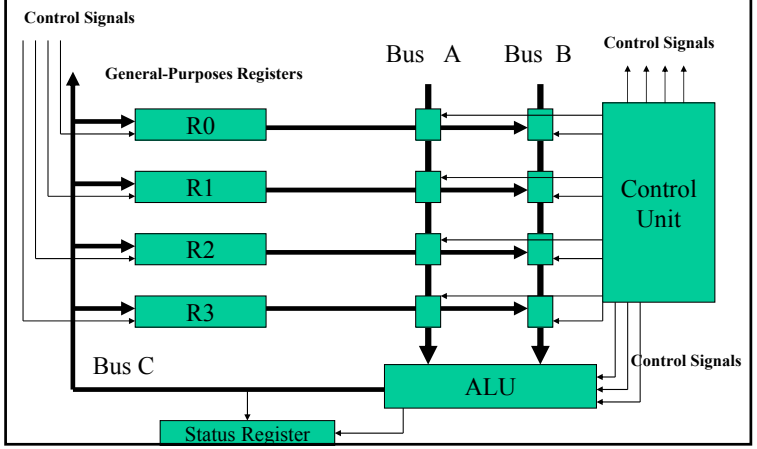
Three-Bit Up-Counter



Four-Bit Synchronous Up-Counter

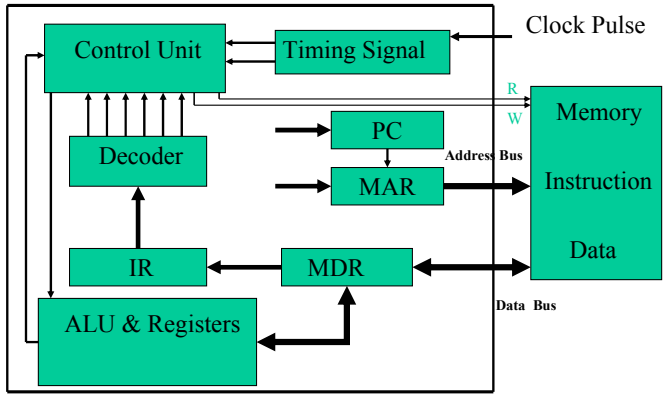


General CPU Bus Organization



Computer Organization
CPU Connection with Memory

PC: Program Counter, DR: Data Register, IR: Instruction Register



7.3 The Memory Unit (RAM)

- A **Memory Unit** is a collection of storage cells together with associated circuits needed to transfer information in and out of storage
- A **Word** is an entity of bits that move in and out of storage as a unit. It may represent a **number**, an **instruction code**, **characters**, or any other **binary-coded information**
- A **Byte** is a group of **eight bits**
- Memory capacity is the total number of bytes that can be stored
- Number of words and the number of bits in each word represent the internal structure of the memory

Random-Access Memory (RAM)

MAR: Memory Address Register holds the address of information to be stored/read from memory.

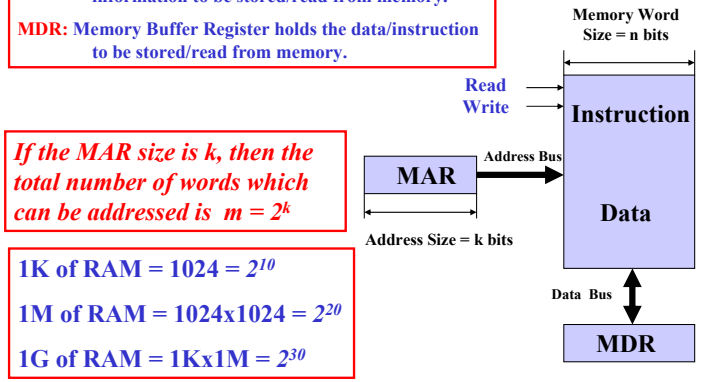
MDR: Memory Buffer Register holds the data/instruction to be stored/read from memory.

If the MAR size is k, then the total number of words which can be addressed is $m = 2^k$

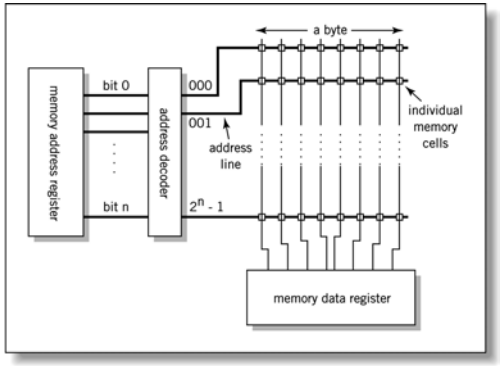
1K of RAM = 1024 = 2^{10}

1M of RAM = 1024x1024 = 2^{20}

1G of RAM = 1Kx1M = 2^{30}

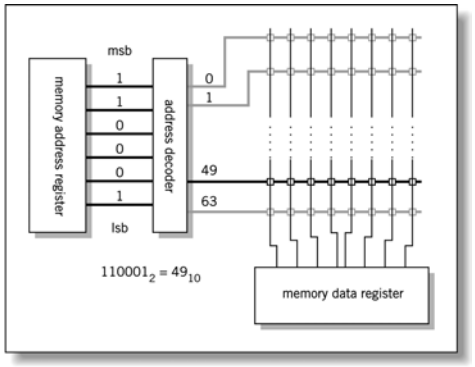


MDR, MAR, and memory

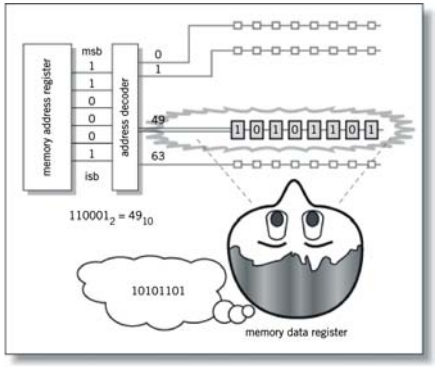


Englander: The Architecture of Computer Hardware and Systems Software, 2nd edition
Chapter 7, Figure 07-04

MAR-MDR example



Englander: The Architecture of Computer Hardware and Systems Software, 2nd edition
Chapter 7, Figure 07-05



Englander: The Architecture of Computer Hardware and Systems Software, 2nd edition
Chapter 7, Figure 07-06

A visual analogy for memory

7.4 The Fetch-Execute Instruction Cycle

- Program to be executed by computer consists of sequence of instructions stored in memory
- The *CPU* executes each instruction of the program according to a predefined steps
- The C*ontrol Unit* of the *CPU* determines these defined steps according to the Instruction Format and the *Computer Architecture*
- A program *instruction* must contains field for *operation code*, field for the *address of the operands*, and field to show where the operands are stored, *address modification* (they will be discussed in Chapter 8)

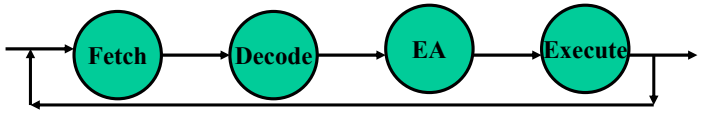
Instruction Codes

- An instruction code is a group of bits that instruct the computer to perform a specific operation.
- It is divided into three parts:
  - 1. The *Operation Code*: ADD, SUB, MULT, DIV, ..etc. The number of bits determines the number of operations. Example: 4-bit size produces 16 Opcodes
  - 2. The *Address Code*: the address of the two operands
  - 3. The *Address Modification Code*: where are the operands stored
- Instruction Format:

Opcode	Operands Address	Address Mod

General Form of A Computer Cycle

- Fetch: Read an Instruction From Memory
- Decode: Decode the Instruction; Obtain the Operation Code and the Operand Address
- Effective Address (EA): Calculate the Address of the Operands
- Execute: Perform the Required Operation and Store the Result in memory/register
- Repeat Until the end of Instructions

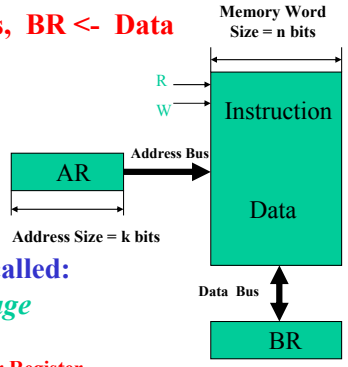


The Instruction must be Fetched from Memory Read and Write Cycle

- **Write to Memory:**
  - **T0 : AR <- Address, BR <- Data**
  - **T1 W: M[AR] <- BR**
- **Read from Memory:**
  - **T0 : AR <- Address**
  - **T1 R: BR <- M[AR]**
- **The Read/Write Cycle is written in a language called:**
  Register Transfer Language

T0 & T1 are Timing Signals

AR: Address Register, BR: Buffer Register



Register Transfer Language

- ***A Digital System*** **is an interconnection of** ***Digital*** **hardware** ***Modules*** **that accomplish a specific information-processing task**
- ***Digital Modules*** **are best defined by the registers they contain and the operations that are performed on the data stored in them**
- ***Micro-operation*** **is an elementary operation performed on the information stored in one or more registers** ***Examples*** **are** ***shift***, ***clear*** **and** ***load*** **operations**

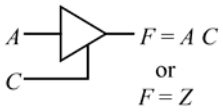
Register Transfer Language *continue*

- **The internal** ***Hardware Organization*** **of a digital computer is best defined by specifying:**
  1. **the** ***set of registers*** **it contains and their function,**
  2. **the sequence of** ***microoperations*** **performed, and**
  3. **the** ***control*** **that initiates the sequence of microoperations**
- **A** ***Register Transfer Language*** **is a system for expressing in** ***Symbolic Form*** **the microoperarion sequences among the registers of a digital system**
- **It is a** ***tool*** **for describing the internal organization of a digital computer in concise and precise manner**

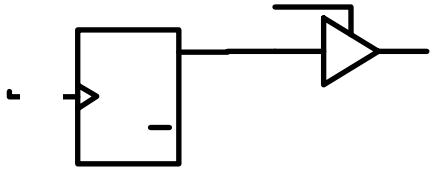
Tri-state Buffer

C	*A*	*F*
0	0	*Z*
0	1	*Z*
1	0	0
1	1	1

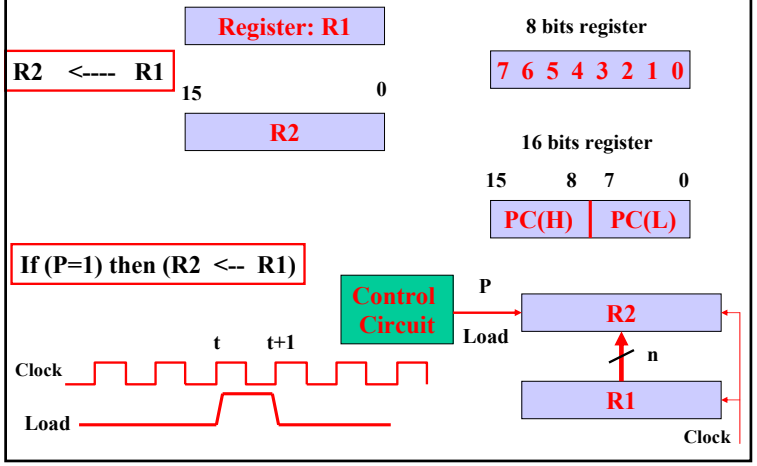
Z: disconnected



In-Out control



Register Transfer



Register Transfer *continue*

T: R2 <-- R1, R1 <-- R2

Symbol	Description	Example
Letters and numerals	Denotes a Register	MAR, R2
Parentheses ( )	Denotes a part of Reg.	R2(0-7),R2(L)
Arrow <---	Denotes transfer of Inf.	R2 <--- R1
Comma,	Separates two microop	R2 <--R1, R1 <-- R2
Colon:	Separate the control from the Operation	

Basic Symbols for Register Transfers

Instruction Execution

- The Program Counter, PC, contains the address of the first instruction of the program
- The first step in the execution of every instruction will be (put the address of the instruction into the memory address register)

  T0 : MAR ← PC
- The second step will be (read the instruction from memory and put it in memory data register)

  T1 : MDR ← M[MAR], PC ← PC + 1
- The third step will be (move the instruction from MDR to the instruction register)

  T2 : IR ← MDR

Instruction Execution *continue*

- The fourth step will be (decode the instruction and determine the address of the operands)

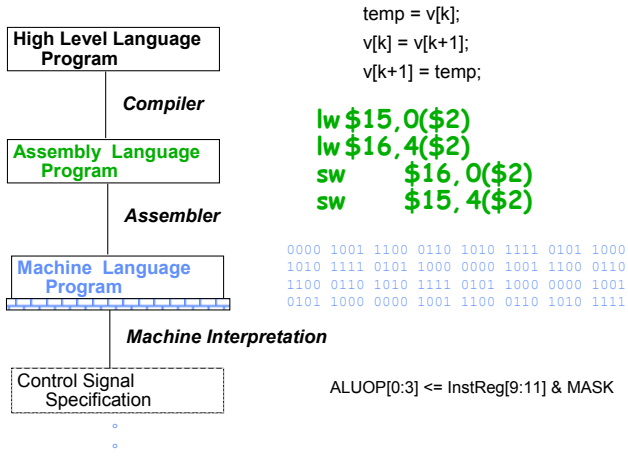
  T3 : Op-Code ← IR [Op-Code], Address ← IR[Address]

- The fifth step will be (execute the instruction which is instruction dependent on the op-code such as Load, Add,… etc). In this case, the operands (data) must be read from memory or they should be stored in registers as shown

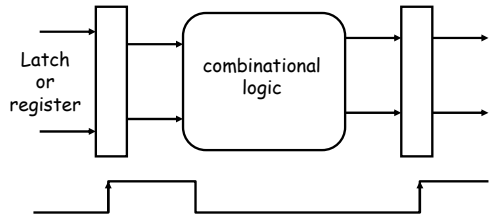
  T4 : A ← M[Address]

- Load the operand from memory and put it in the accumulator register A

Levels of Representation (61C Review)

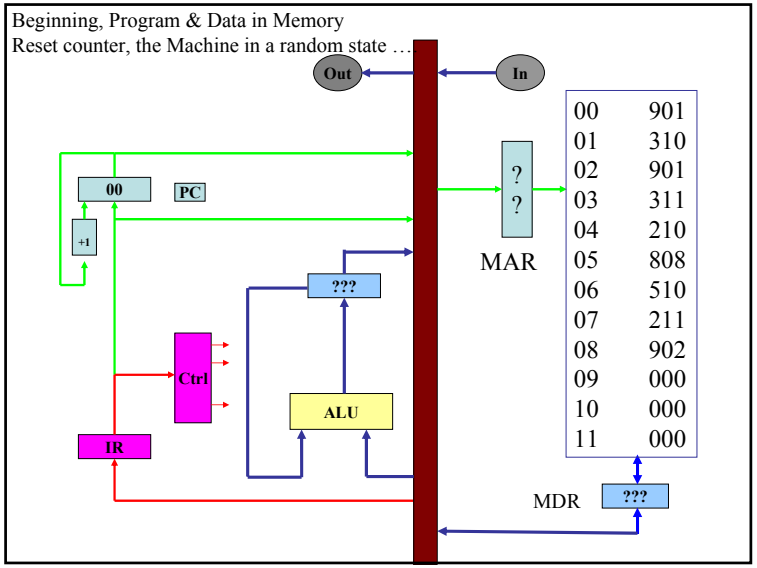


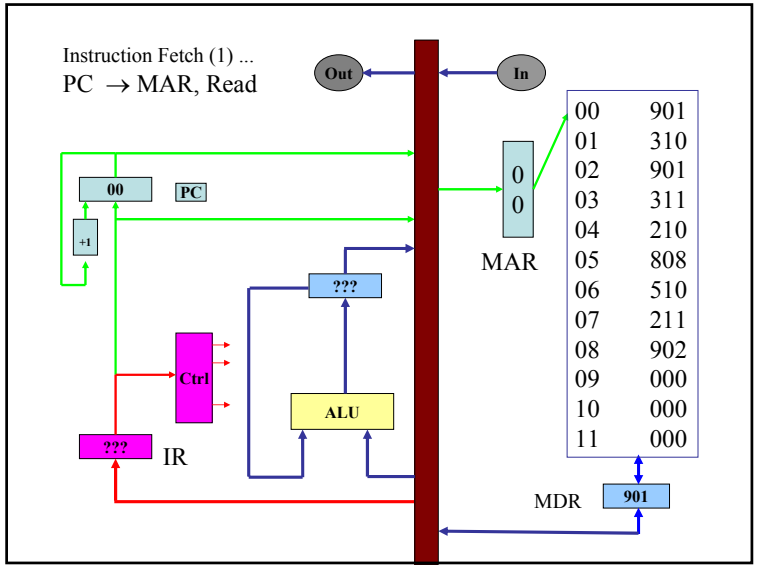
What's a Clock Cycle?

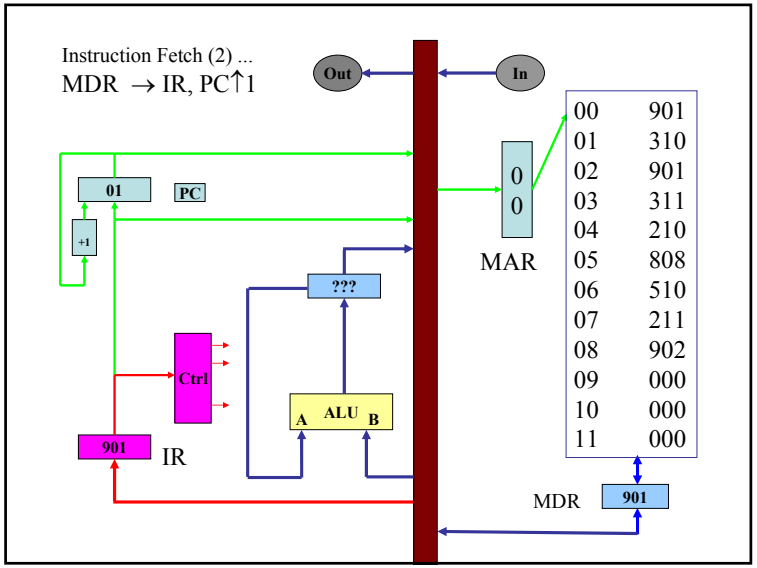


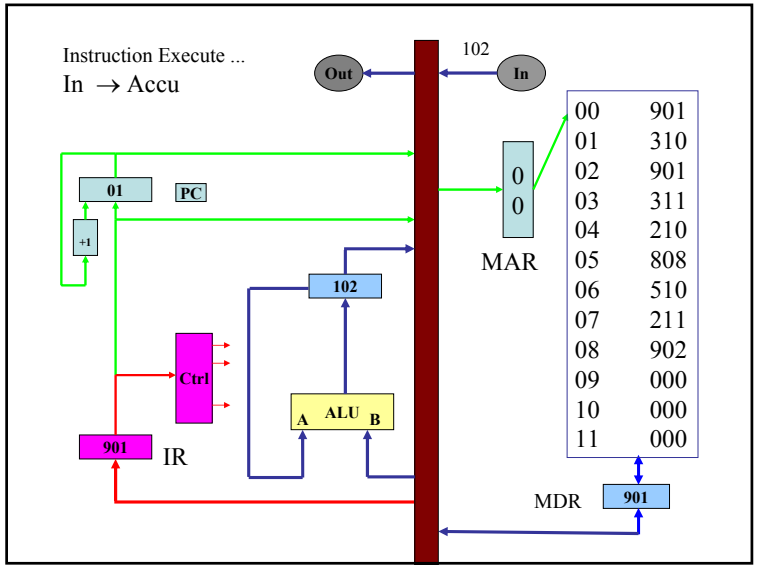
- Old days: 10 levels of gates
- Today: determined by numerous time-of-flight issues + gate delays
  - clock propagation, wire lengths, drivers

- Instruction 00

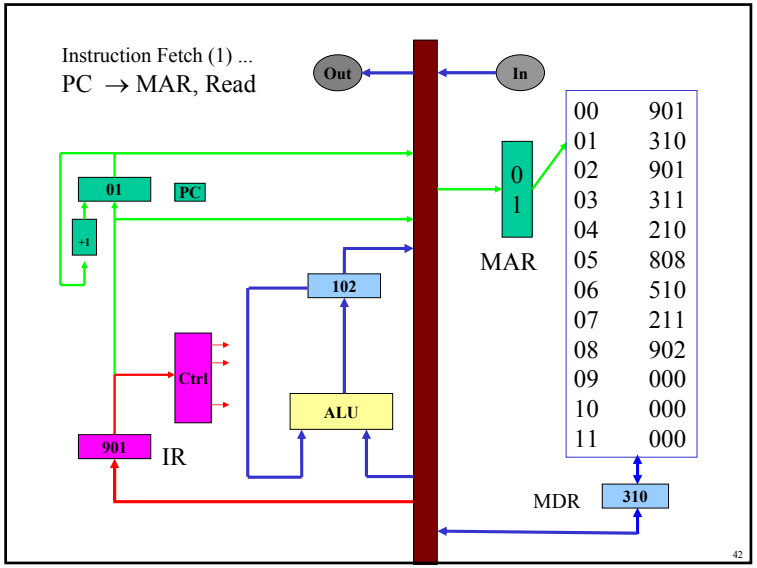

Beginning, Program & Data in Memory
Reset counter, the Machine in a random state …
Out
In
00
PC
+1
???
Ctrl
ALU
IR
? ?
MAR
MDR
???
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 000
11 000

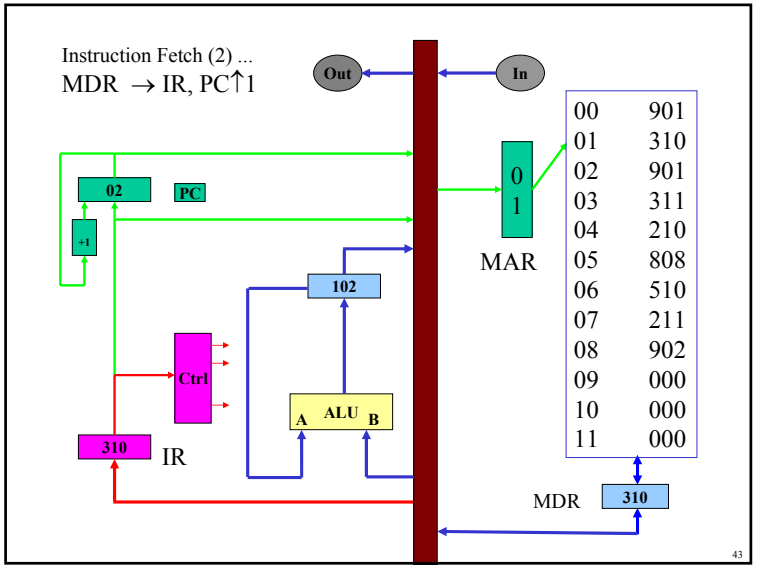

Instruction Fetch (1) ...
PC → MAR, Read
Out
In
00
PC
+1
???
Ctrl
ALU
???
IR
0 0
MAR
MDR
901
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 000
11 000

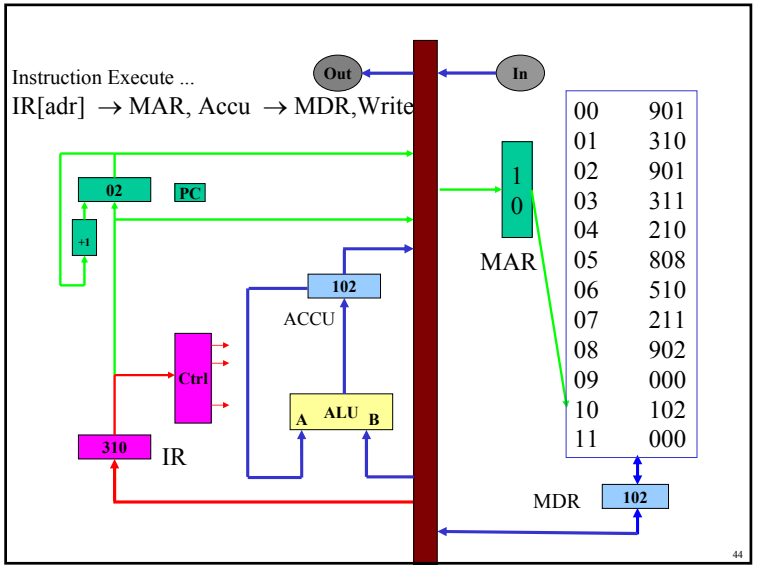

Instruction Fetch (2) ...
MDR → IR, PC↑1
Out
In
01
PC
+1
???
Ctrl
A ALU B
901
IR
0 0
MAR
MDR
901
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 000
11 000


Instruction Execute ...
In → Accu
102
Out
In
01
PC
+1
102
Ctrl
A ALU B
901
IR
0 0
MAR
MDR
901
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 000
11 000

☺ Next instruction: 01



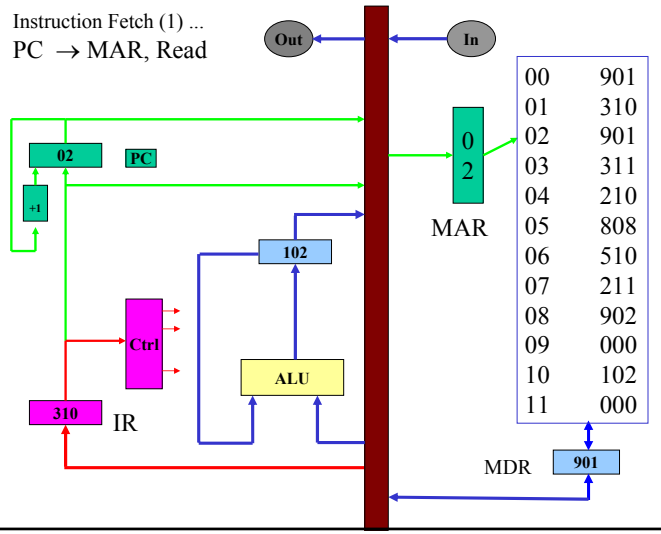




☺ Next instruction: 02

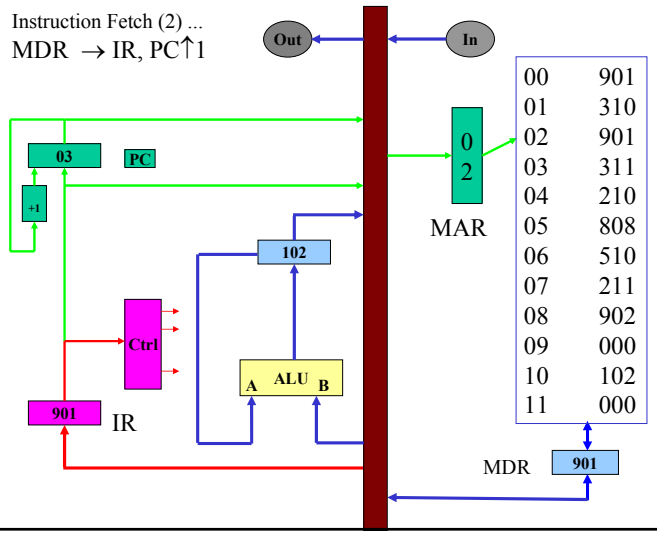
Instruction Fetch (1) ...

PC → MAR, Read



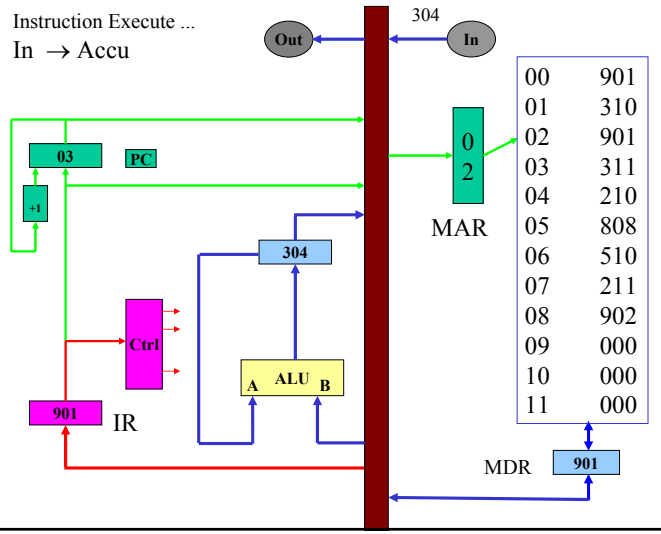
Instruction Fetch (2) ...

MDR → IR, PC↑1



Instruction Execute ...

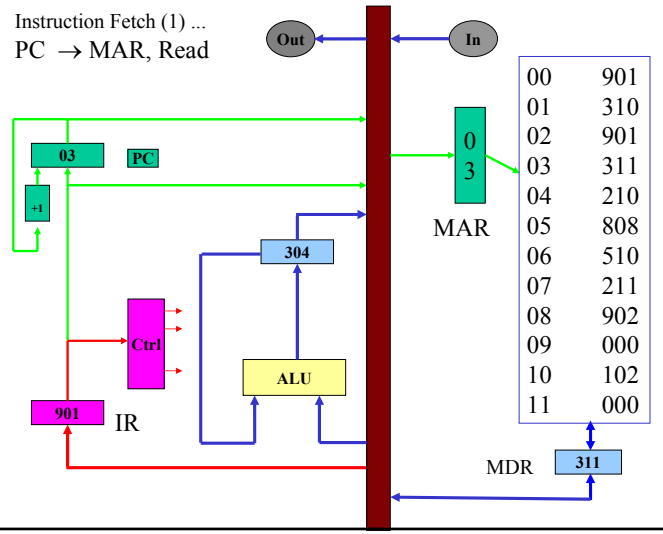
In → Accu



☺ Next instruction: 03

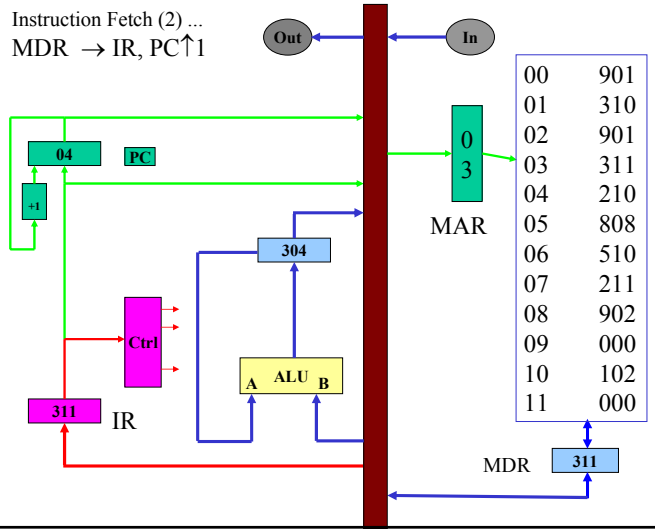
Instruction Fetch (1) ...

PC → MAR, Read



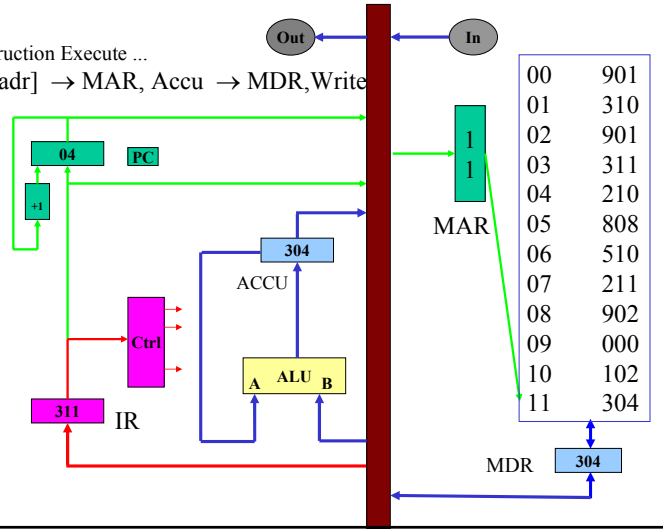
Instruction Fetch (2) ...

MDR → IR, PC↑1



Instruction Execute ...

IR[adr] → MAR, Accu → MDR,Write

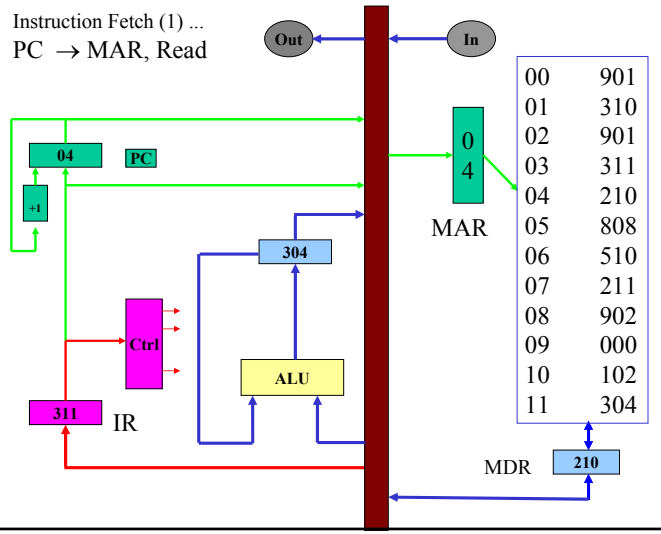


☺ Next instruction: 04

53

Instruction Fetch (1) ...

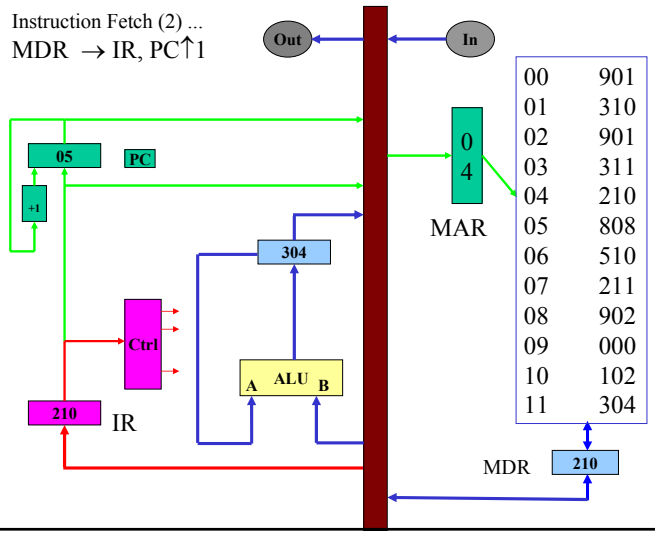
PC → MAR, Read



54

Instruction Fetch (2) ...

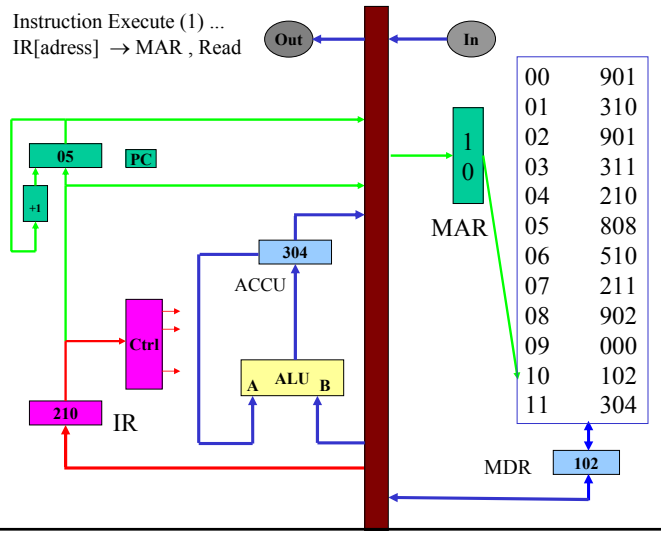
MDR → IR, PC↑1



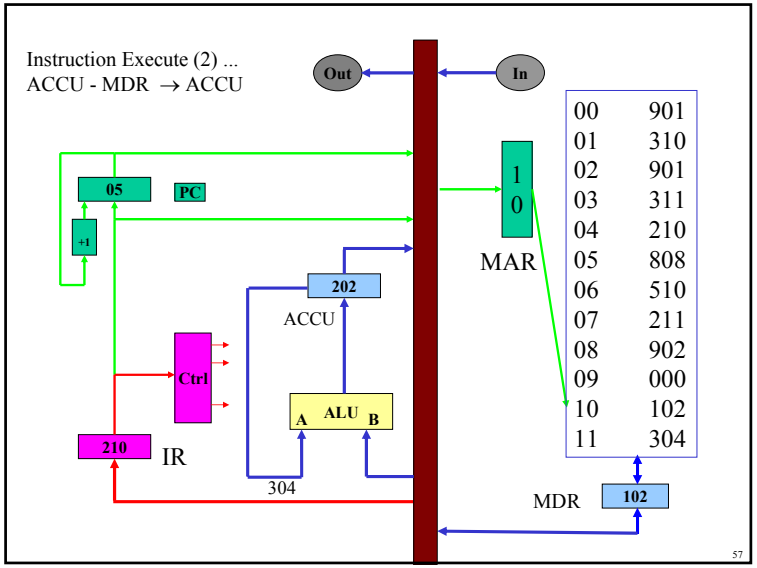
55

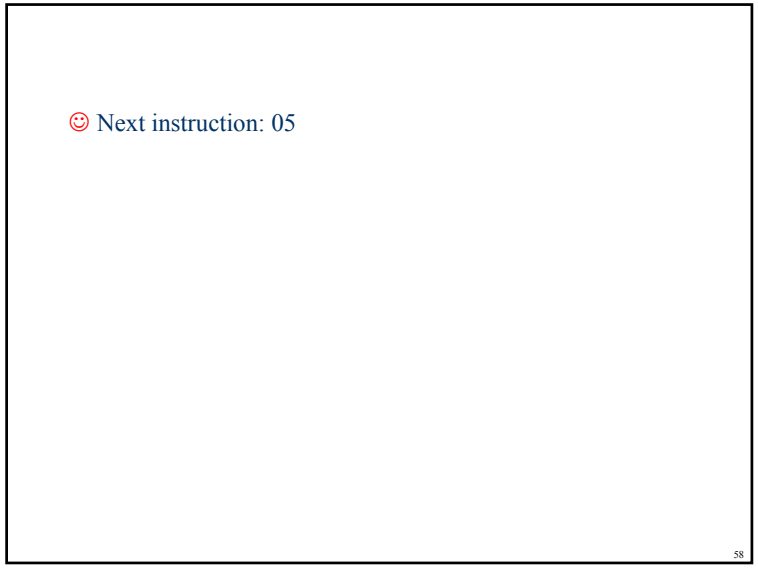
Instruction Execute (1) ...

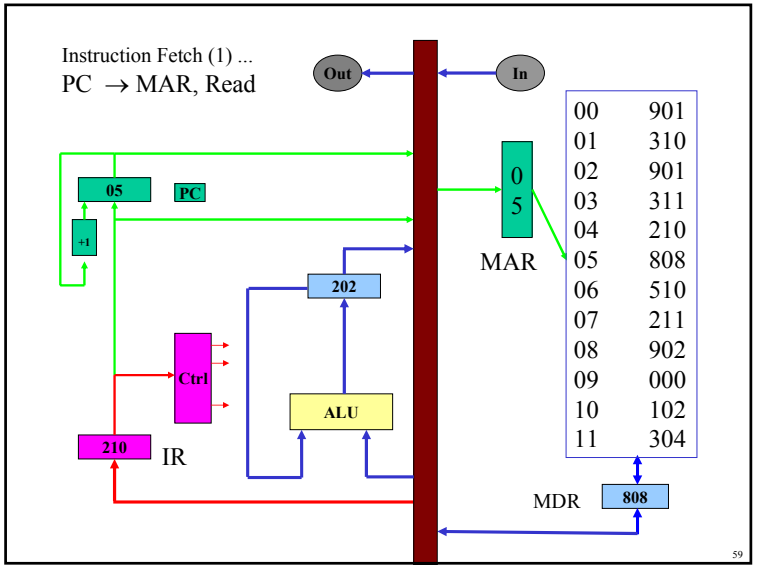
IR[adress] → MAR , Read

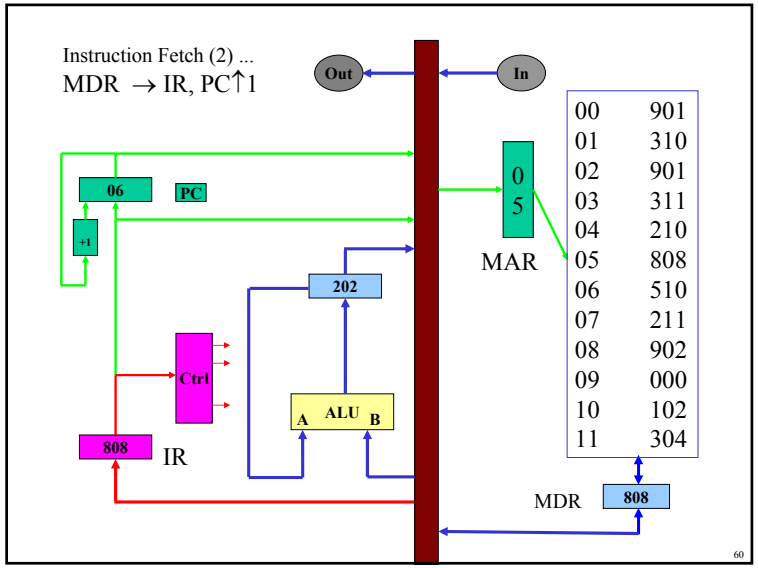


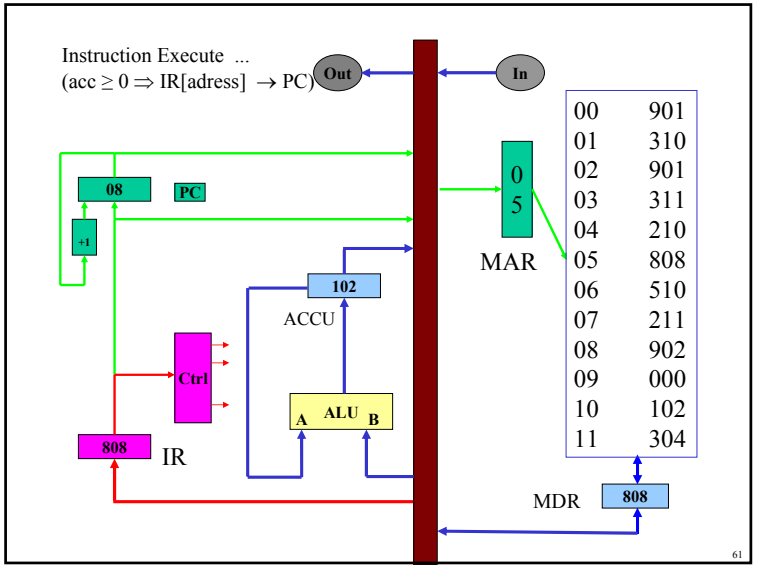
56

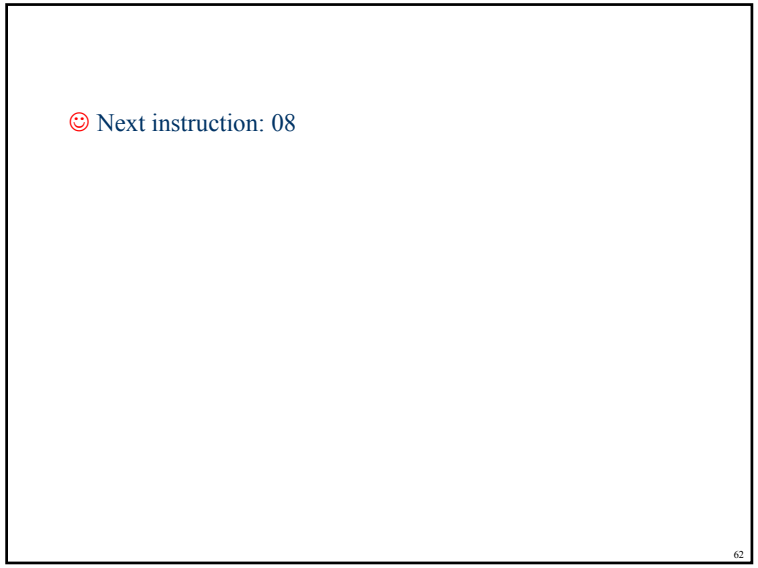

Instruction Execute (2) ...
ACCU - MDR → ACCU
Out
In
05
PC
+1
202
ACCU
Ctrl
A ALU B
210
IR
304
1
0
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
102
57

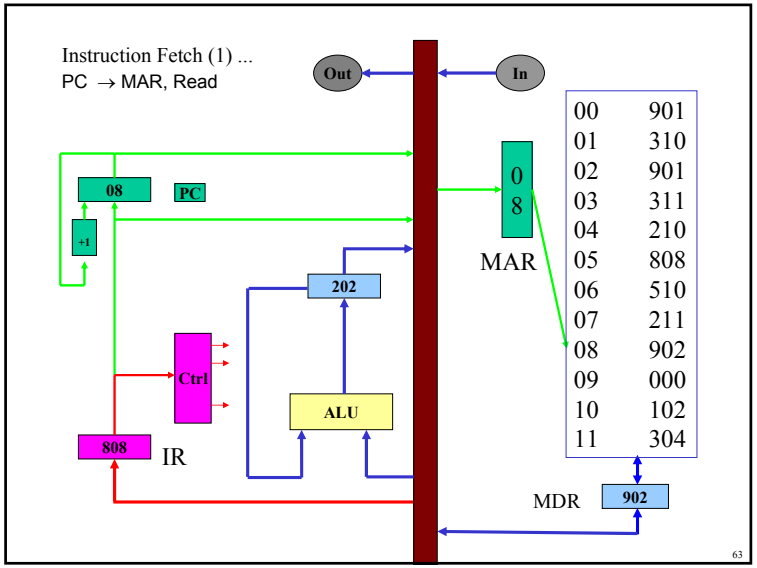

☺ Next instruction: 05
58

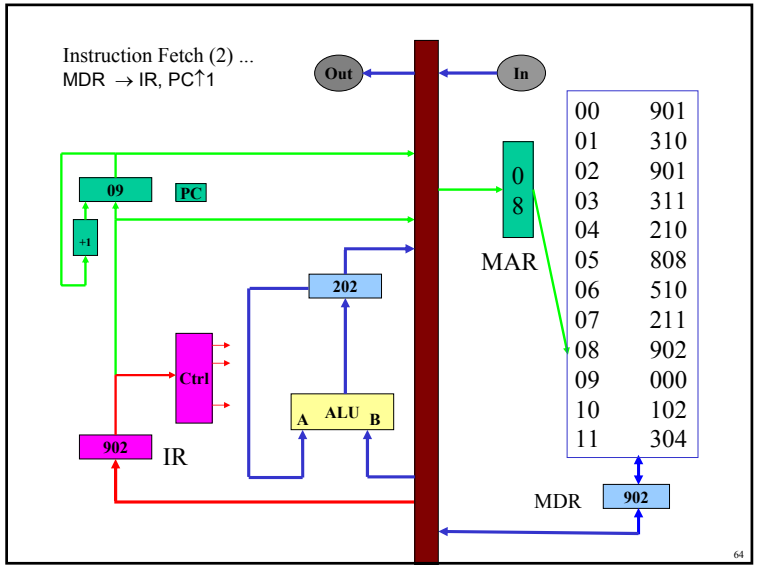

Instruction Fetch (1) ...
PC → MAR, Read
Out
In
05
PC
+1
202
Ctrl
ALU
210
IR
0
5
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
808
59

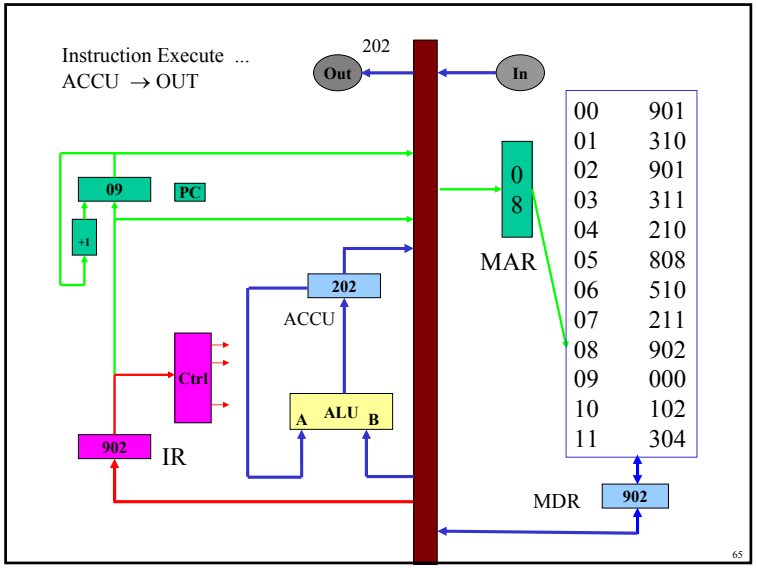

Instruction Fetch (2) ...
MDR → IR, PC↑1
Out
In
06
PC
+1
202
Ctrl
A ALU B
808
IR
0
5
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
808
60

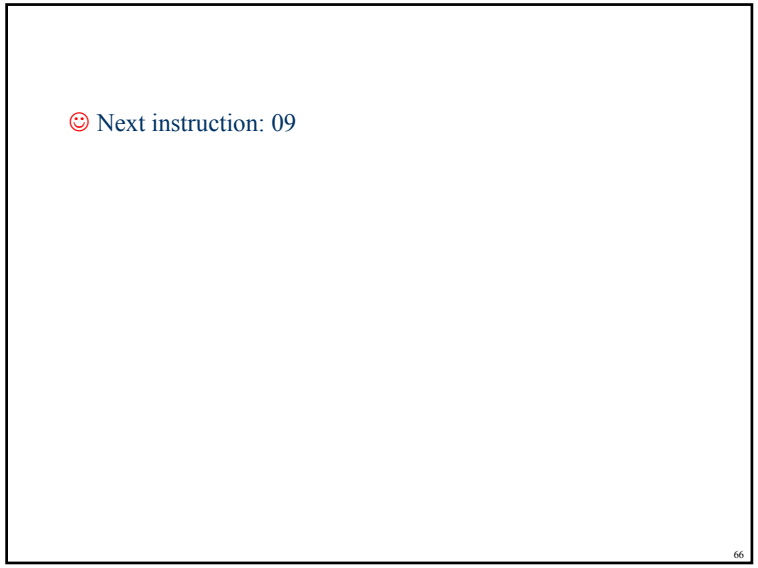

Instruction Execute ...
(acc ≥ 0 ⇒ IR[adress] → PC)
Out
In
08
PC
+1
102
ACCU
Ctrl
A ALU B
808
IR
05
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
808
61

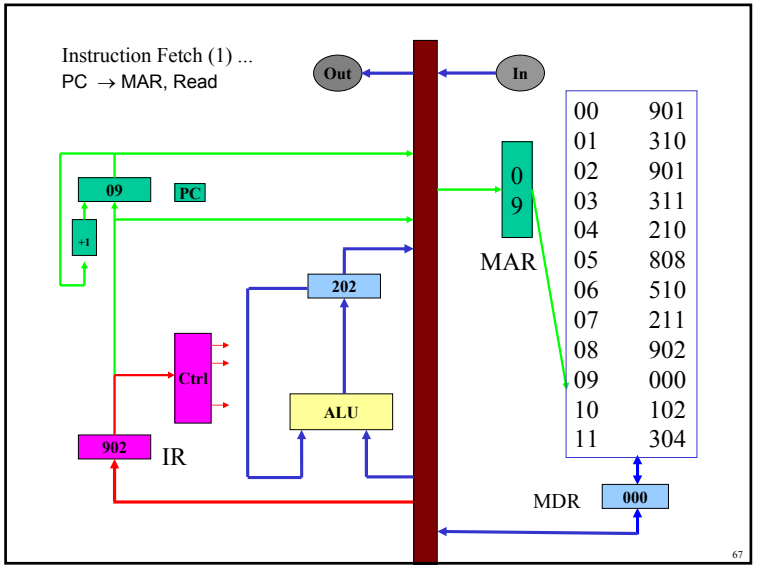

☺ Next instruction: 08
62

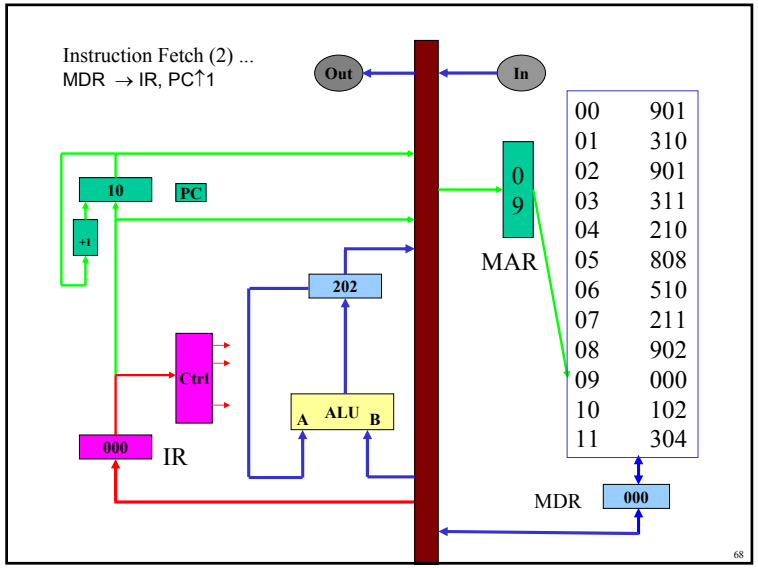

Instruction Fetch (1) ...
PC → MAR, Read
Out
In
08
PC
+1
202
Ctrl
ALU
808
IR
08
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
902
63

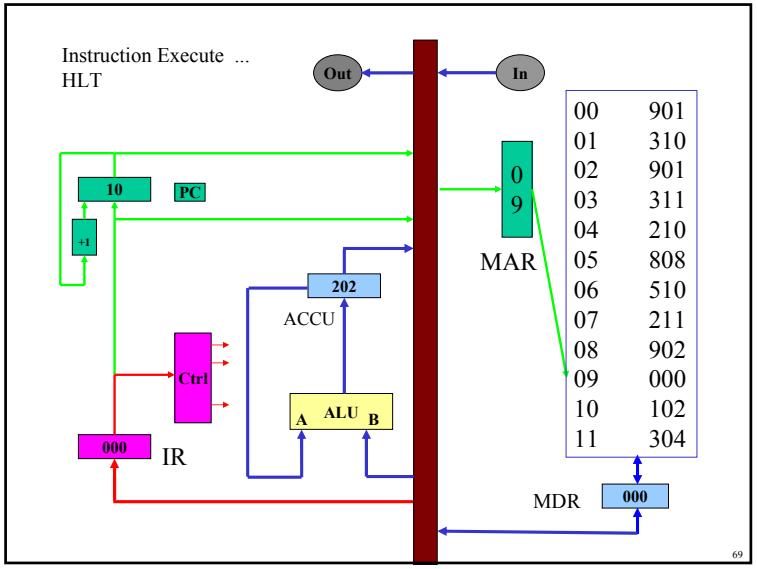

Instruction Fetch (2) ...
MDR → IR, PC↑1
Out
In
09
PC
+1
202
Ctrl
A ALU B
902
IR
08
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
902
64


Instruction Execute ...
ACCU → OUT
202
Out
In
09
PC
+1
202
ACCU
Ctrl
A ALU B
902
IR
0
8
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
902
65


☺ Next instruction: 09
66


Instruction Fetch (1) ...
PC → MAR, Read
Out
In
09
PC
+1
202
Ctrl
ALU
902
IR
0
9
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
000
67

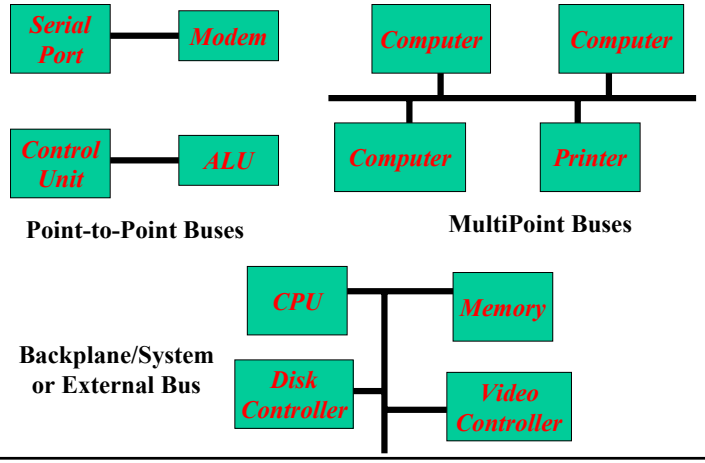

Instruction Fetch (2) ...
MDR → IR, PC↑1
Out
In
10
PC
+1
202
Ctrl
A ALU B
000
IR
0
9
MAR
00 901
01 310
02 901
03 311
04 210
05 808
06 510
07 211
08 902
09 000
10 102
11 304
MDR
000
68



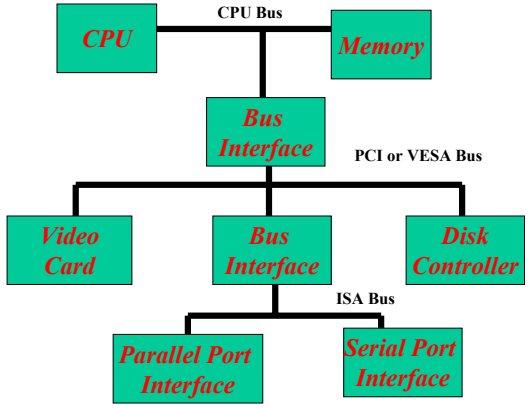
7.5 Buses

- A Bus is a group of electrical conductors suitable for carrying computer signals from one location to another
- Buses are used to transfer data/instructions among computer peripherals, memory and the CPU
- The lines on a bus can be grouped into four categories:
  - Data,
  - Addressing,
  - Control, and
  - Power

Hardware Components Interconnected with Buses



Typical PC Interconnections



Different Buses for Connecting Different parts of the Computer

- The buses internal to the CPU don't have a names
- The CPU bus, *Peripheral Connect Interface* (PCI) or *Video Electronic Standards Association* (VESA) local bus, and *Industry Standard Architecture* (ISA) bus are all part of the Backplane as shown
- The PCI, VESA local, also known as VL, and ISA buses are examples of popular modern external buses
- A *bus protocol* is an agreement between two or more entities that establish a clear, common path of communication and understanding between them
- Buses can be characterized by:
  - *Throughput* (Data Transfer Rate, bit/second),
  - the *Data Width* in bits,
  - the distance between the two end points,
  - the type of control required,
  - the type of bus,
  - the addressing capacity, and
  - the dedicated line or shared line inside the bus

7.6 Timing Issues

- The time of the events at the CPU is synchronized to the pulses of an electronic clock
- The clock provides a master control as to when each step in the instruction cycle takes place
- The pulses in the clock is separated sufficiently to assure that each step has enough time to complete, with the data settled down, before the results of that step are required by the next step
- Wait state is provided when the clock pulse is not wide enough (slow down the speed of the computer)
- Each clock pulse is used to control one step in the sequence of the instruction execution
- Sometime, additional pulses might be used to control different details within a single step