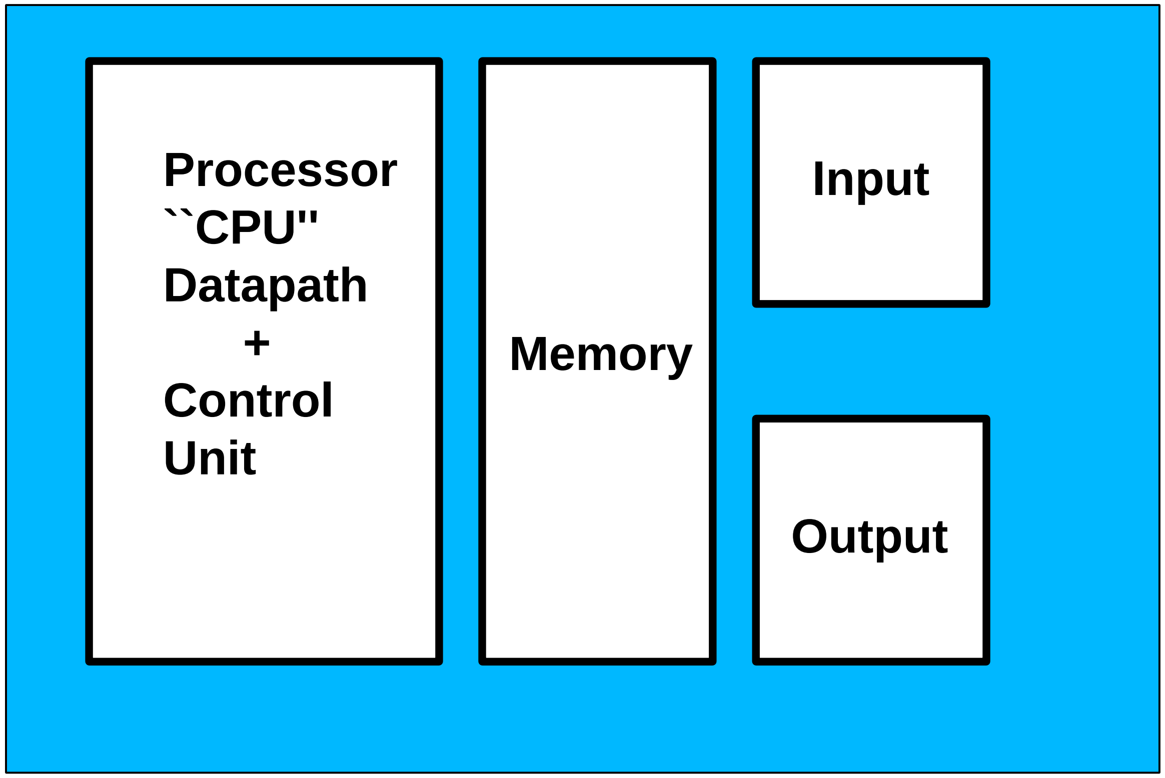
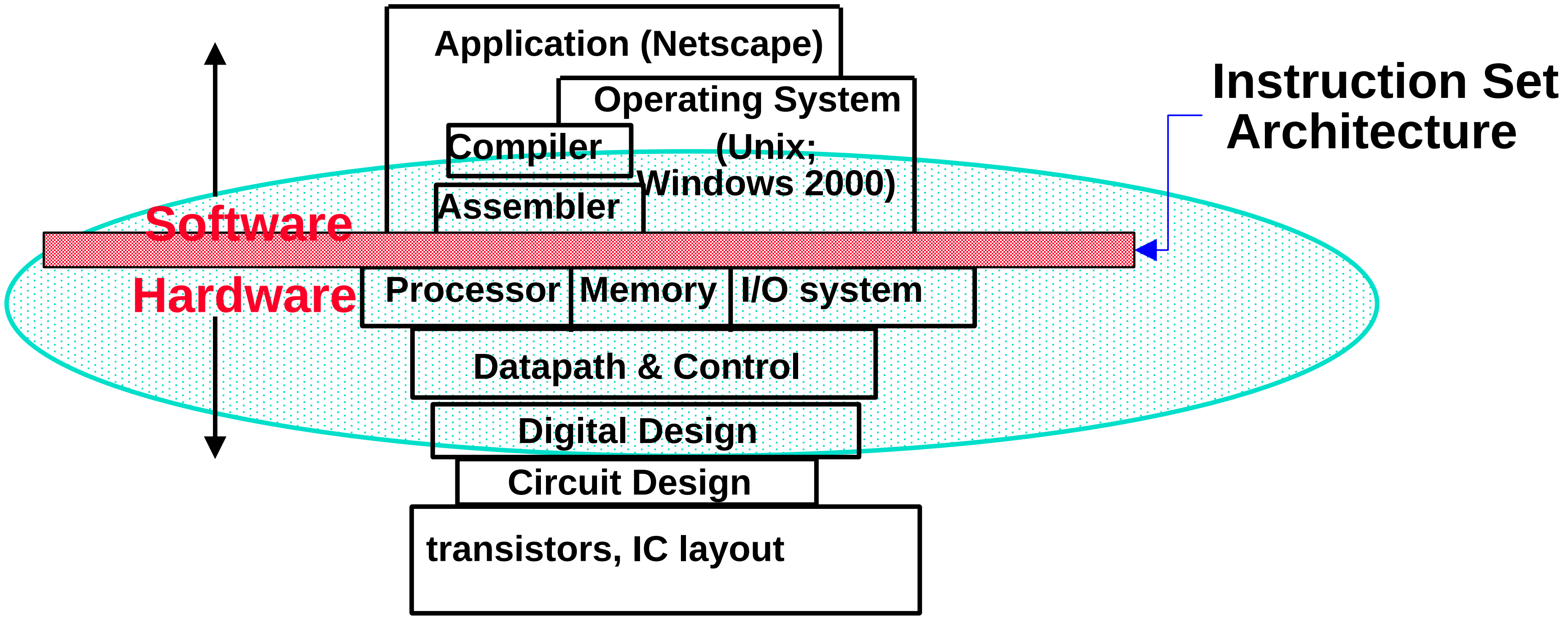
Computer Systems

Since 1946 all computers have had 5 components

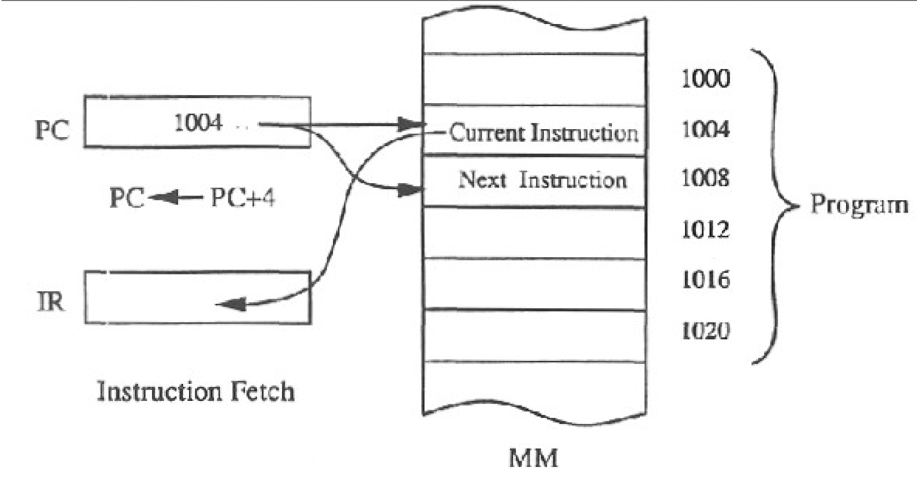


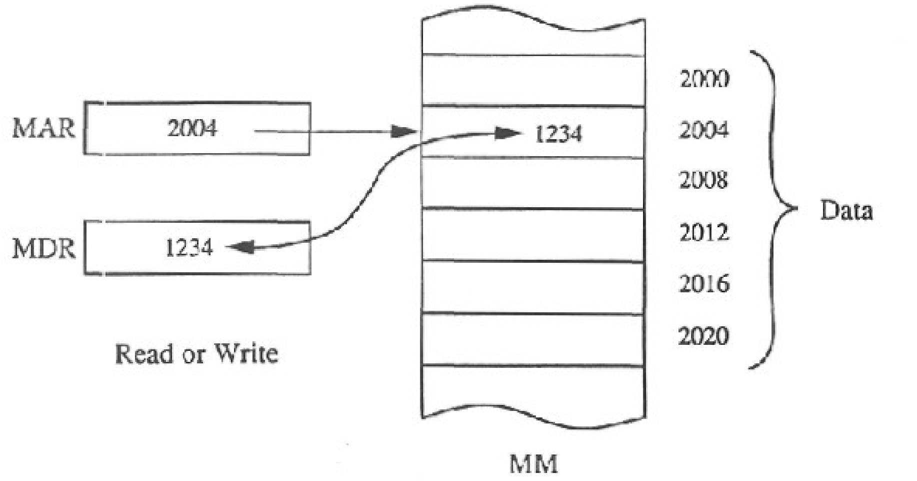
Interconnection Structures (buses)

What is ``Computer Architecture''?



- **Co-ordination of many levels of abstraction**
  - hide unnecessary implementation details
  - helps us cope with enormous complexity of real systems
- **Under a rapidly changing set of forces**
- **Design, Measurement, *and* Evaluation**


PC
1004
PC ← PC+4
IR
Instruction Fetch
Current Instruction
Next Instruction
1000
1004
1008
1012
1016
1020
Program
MM


MAR
2004
MDR
1234
Read or Write
1234
2000
2004
2008
2012
2016
2020
Data
MM

Single bus CPU

BUS: a collection of wires for transmitting addresses, data and signals.

Decoder: fetch instructions from main memory and determine their type.

ALU: performs performs arithmetic and logical operations.

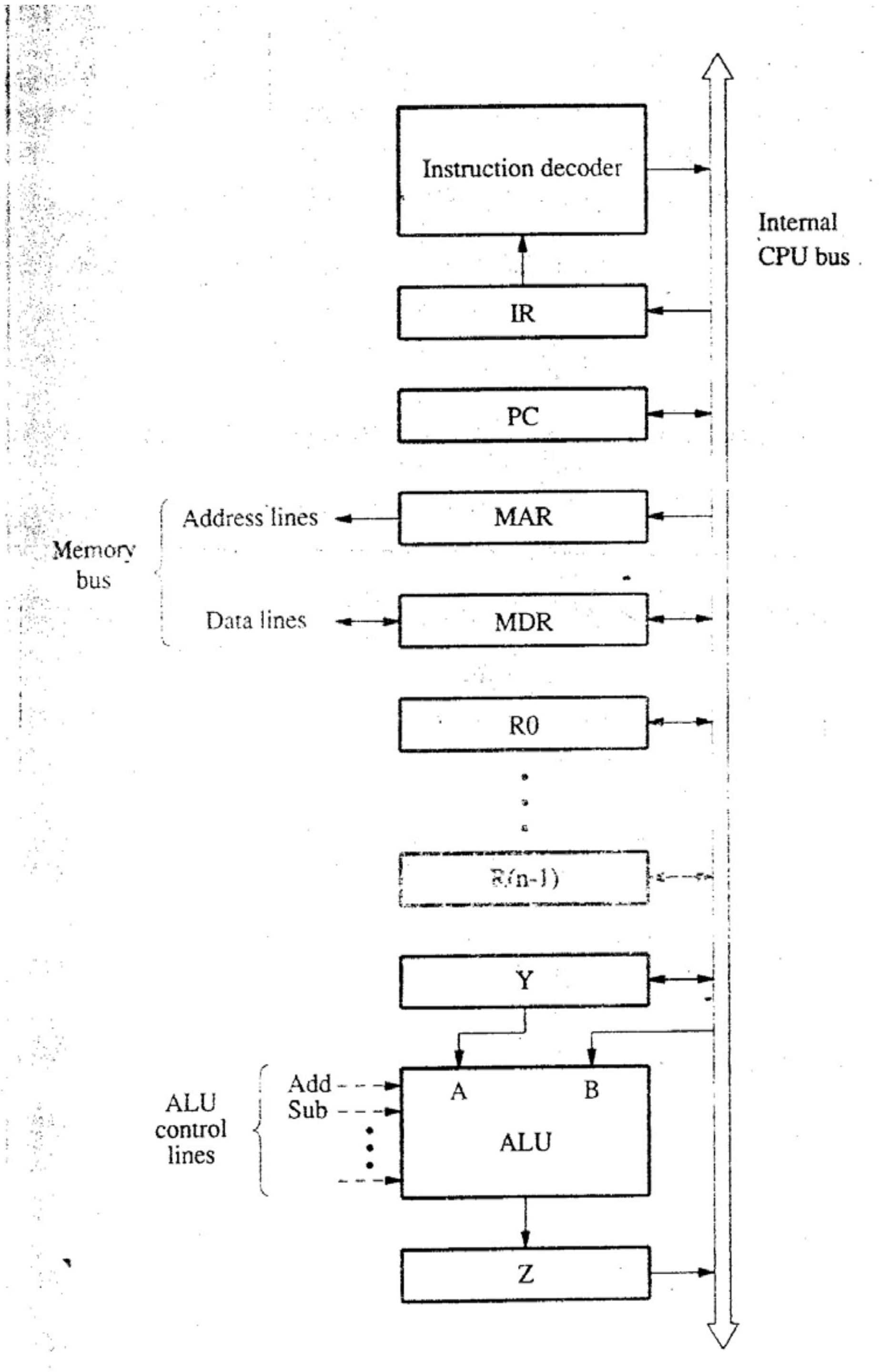
Memory: stores temporary results and control information.

Register: can hold one number, be read/written at high speed.

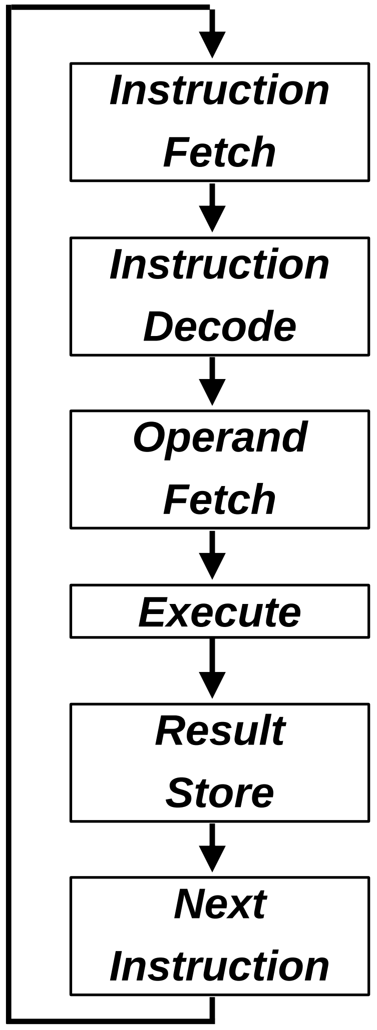
PC: points to the next instruction to be fetched.

MAR: Memory address register

MDR: Memory data register



Execution Cycle



Obtain instruction from program storage

Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use

Determine successor instruction

Fetch and store operations

FETCH

Memory location address in R1 and memory data in R2

1. **MAR ← [R1]**
2. **Read**
3. **Wait for MFC signal**
4. **R2 ← [MDR]**

STORE

For writing a word into a given memory location:

1. **MAR ← [R1]**
2. **MDR ← [R2], Write**
3. **Wait for MFC**

MFC: Memory-function-completed (control signal)

Register transfers

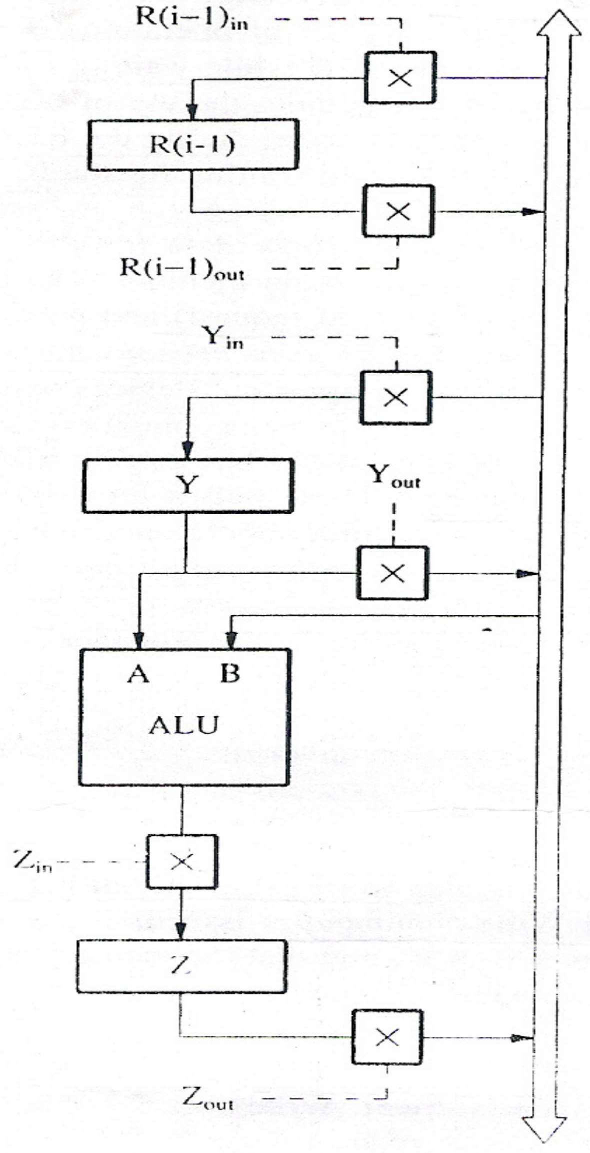
Data transfer requires input and output gates. These are controlled in and out signals. For register Ri:

Ri_in = 1 => data available on common bus is loaded into Ri

Ri_out = 1 => Contents of Ri are placed on the bus.

To transfer contents of R1 to R4

- **Set R1_out to 1 => contents of R1 placed on CPU bus**
- **Set R4_in to 1 => data loaded from the CPU bus into R4**



Program Execution in CPU

Instructions constituting a program to be executed are loaded in sequential locations in main memory. The CPU fetches one instruction at a time and performs the functions specified.

The CPU keeps track of address of the next instruction through a dedicated CPU register called program counter (PC).

Execution of each instruction requires the following steps:

1. Fetch the contents of the memory location pointed to by PC. Contents of this location are interpreted as an instruction to be executed. Hence, they are loaded into the instruction register (IR), i.e. $IR \leftarrow [[PC]]$
2. Increment the contents of the PC by 1, i.e. $PC \leftarrow [PC] + 1$
3. Carry out the actions specified by the instructions in the IR.

If an instruction occupies more than one word, steps 1 and 2 can be repeated as many times as required. These two steps are called the *fetch phase*; step 3 constitutes the *execution phase*.

Most of the operations in Steps 1 to 3 above can be carried out by performing the following functions in a pre-specified sequence:

1. Fetch the contents of a given memory location and load them into a CPU register.
2. Store a word of data from a CPU register into a given memory location.
3. Transfer a word of data from one CPU register to another or to the ALU
4. Perform an arithmetic or logic operation and store the result into a CPU register.

Instructions

- **An instruction is a word containing a bit pattern defining a basic machine instruction and its operands**
    - Add values in registers r1 and r2 and store sum in r0
        - `add r0, r1, r2`
- **Compare an instruction in a high level language**
    - `c = a + b`
    - A compiler takes this line of text and typically converts it to several machine instructions
        - (Assume addresses of a, b, c already in registers r0, r1, r2)

```
ld r3, r0    ;Get value of a into r3
ld r4, r1    ;Get value of b into r4
add r3,r3,r4  ;Add values in r3, r4 – store sum in r3
st r3, r2    ;Store sum in c
```