Course – Computer Organization and Architecture

Course Instructor

Dr. Umadevi V

Department of CSE, BMSCE

Unit-5

Some Fundamental Concepts, Fundamental Basic Processing Unit: Some Fundamental Concepts , Instruction Execution, Hardware Components, Instruction Fetch and Execution Steps, Control Signals, Hardwired Control

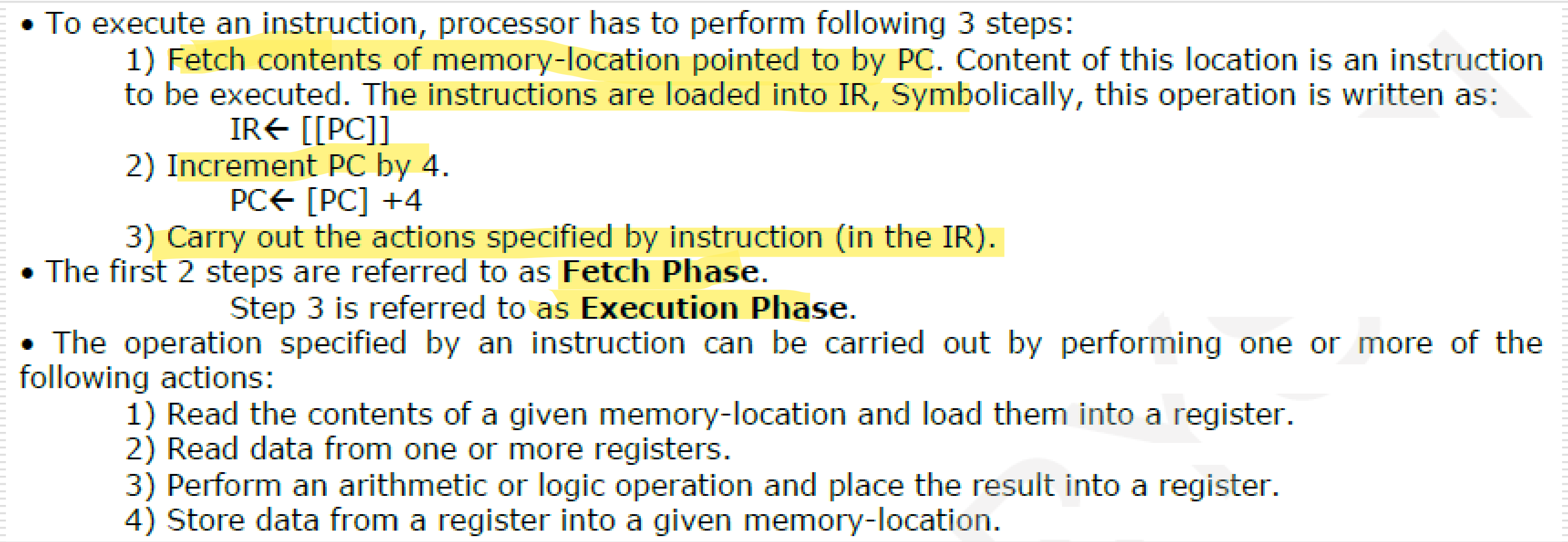
Parallel Computer Architecture: Processor Architecture and Technology Trends, Flynn’s Taxonomy of Parallel Architectures, Memory Organization of Parallel Computers: Computers with Distributed Memory Organization, Computers with Shared Memory Organization, Thread-Level Parallelism: Simultaneous Multithreading, Multicore Processors

Basic Processing Unit

Fundamental Concepts

- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- Instruction Register (IR)

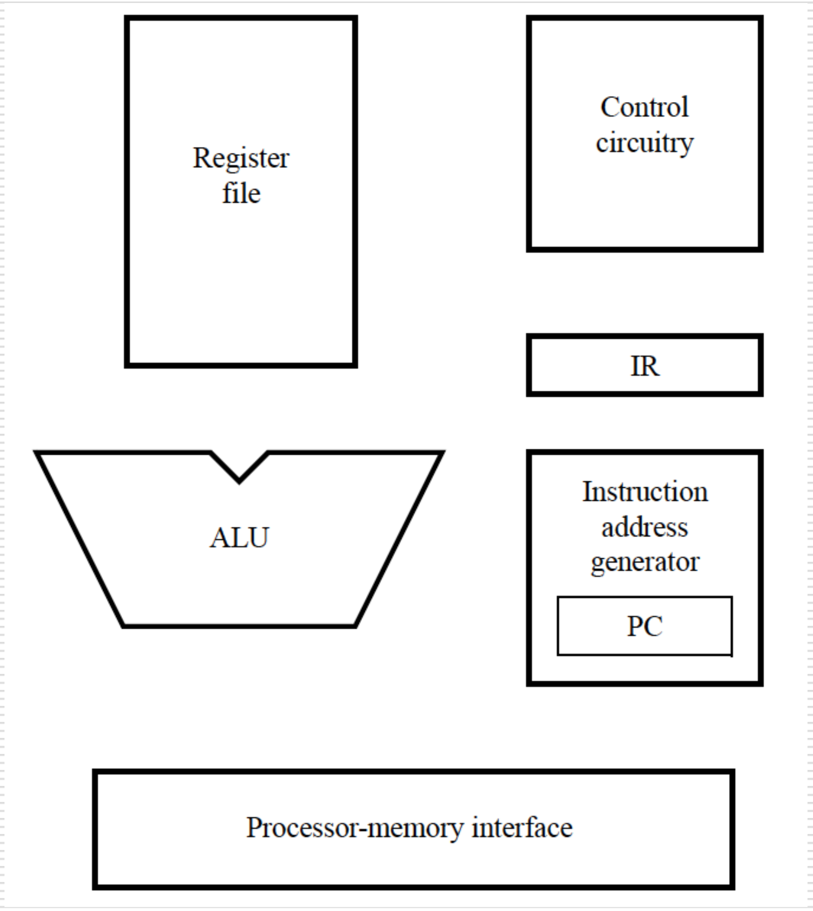
Fundamental Concepts (Contd…)



- To execute an instruction, processor has to perform following 3 steps:
  1) Fetch contents of memory-location pointed to by PC. Content of this location is an instruction to be executed. The instructions are loaded into IR, Symbolically, this operation is written as:
     IR← [[PC]]
  2) Increment PC by 4.
     PC← [PC] +4
  3) Carry out the actions specified by instruction (in the IR).
- The first 2 steps are referred to as **Fetch Phase**.
  Step 3 is referred to as **Execution Phase**.
- The operation specified by an instruction can be carried out by performing one or more of the following actions:
  1) Read the contents of a given memory-location and load them into a register.
  2) Read data from one or more registers.
  3) Perform an arithmetic or logic operation and place the result into a register.
  4) Store data from a register into a given memory-location.

Fundamental Concepts (Contd…)

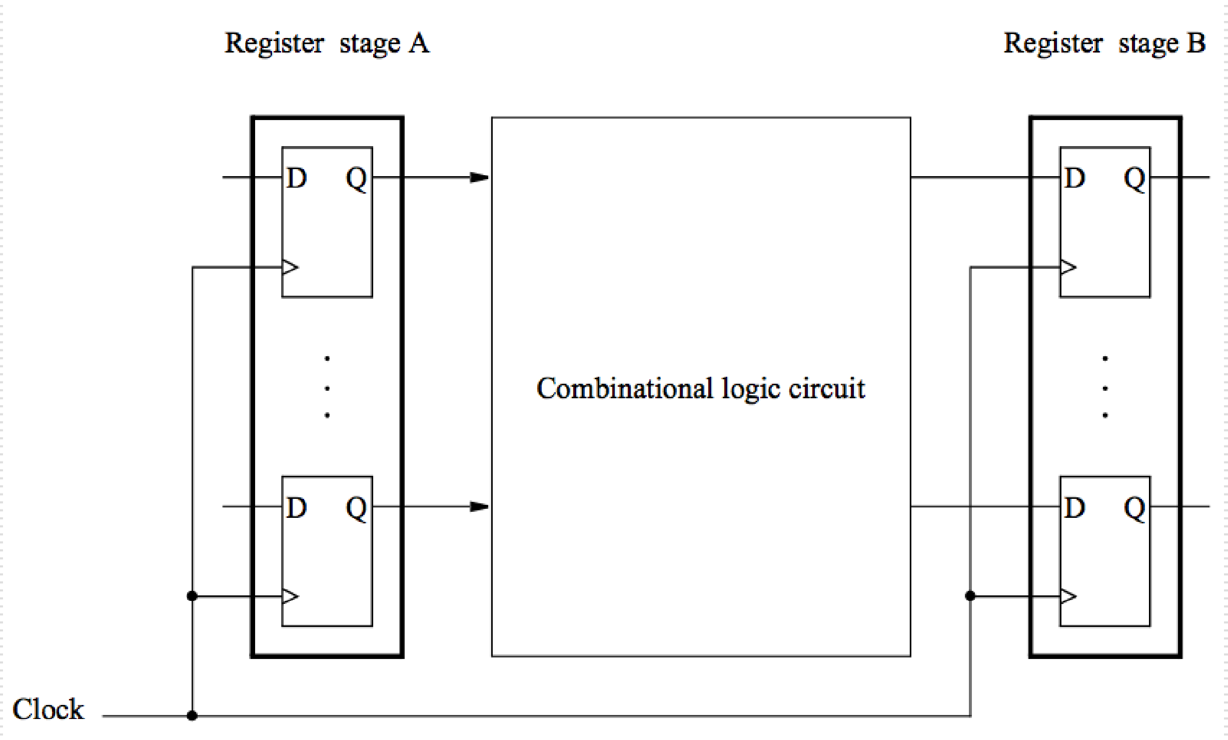
- PC provides instruction address.
- Instruction is fetched into IR
- Instruction address generator updates PC
- Control circuitry interprets instruction and generates control signals to perform the actions needed.



Main hardware components of a processor

Data Processing Hardware

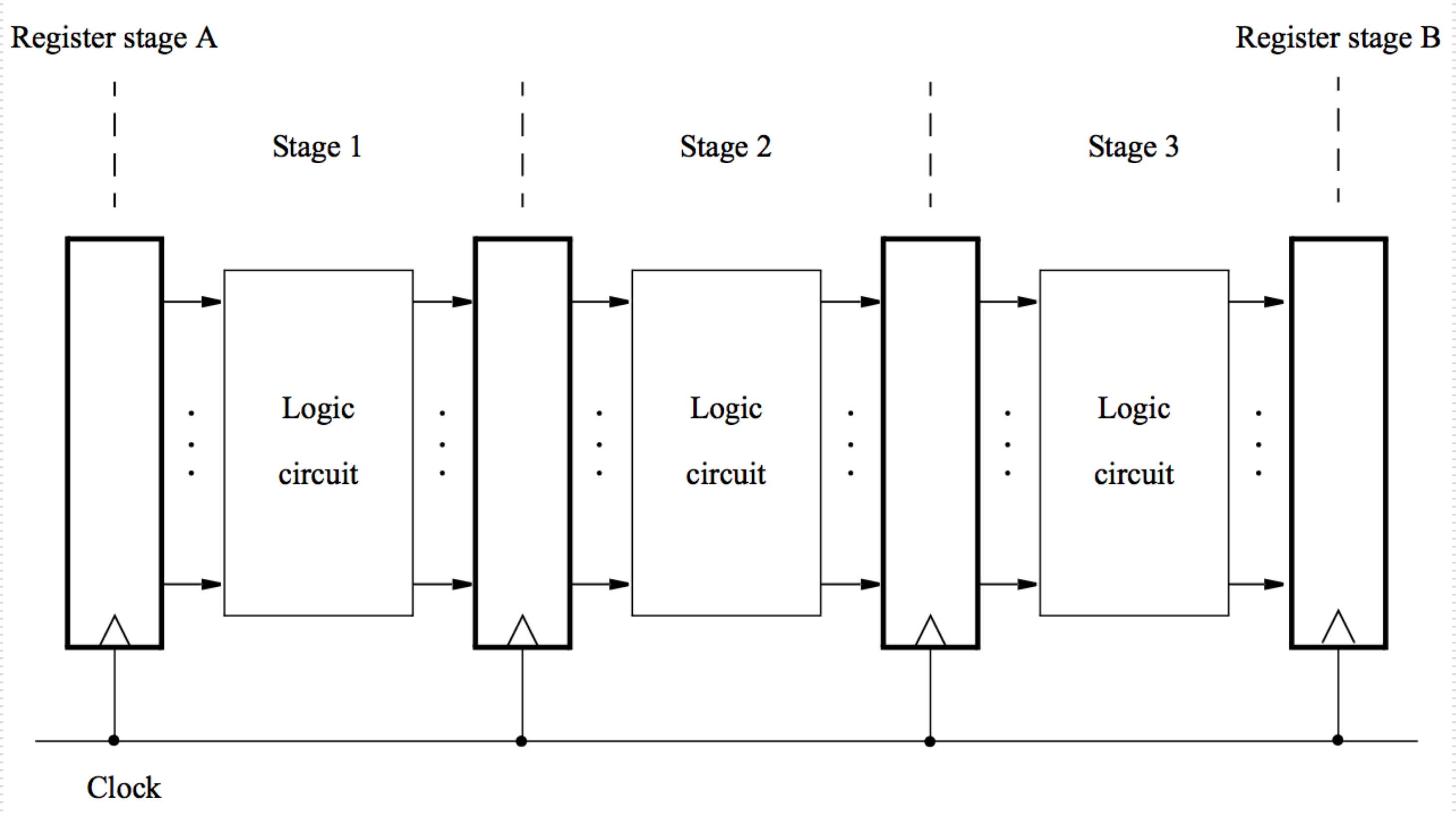
Contents of register A are processed and deposited in register B.



Basic structure for data processing

Data Processing Hardware (Contd…)

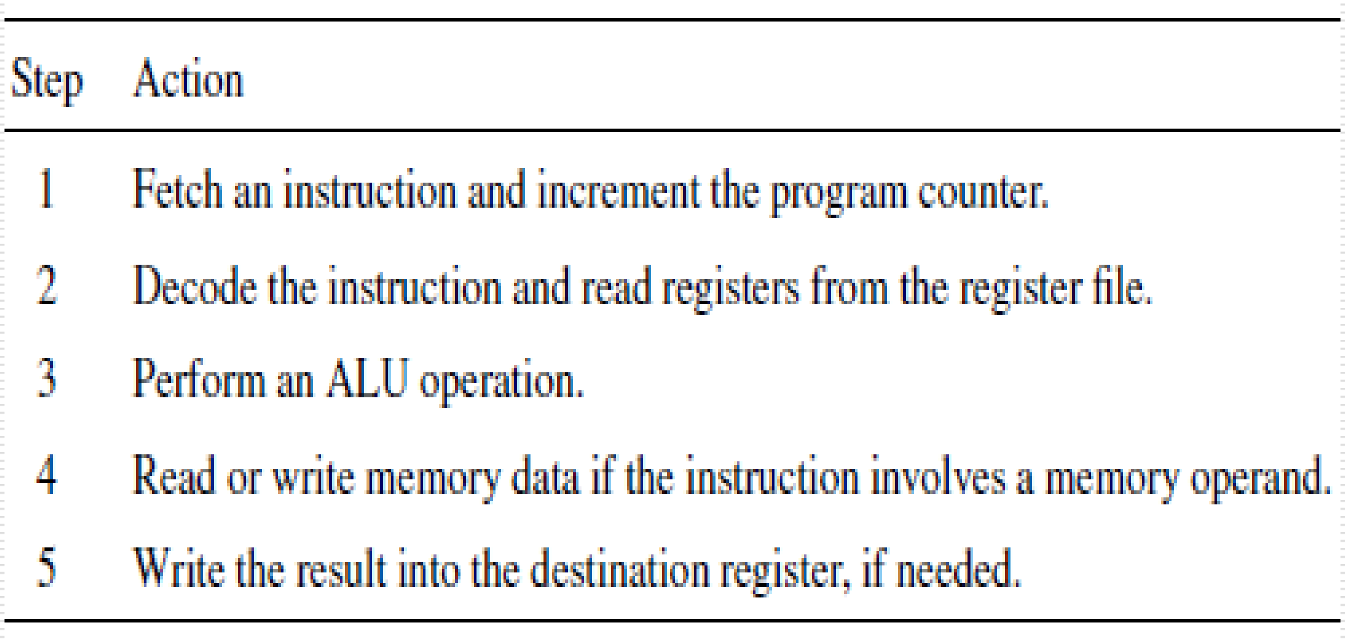
Processing moves from one stage to the next in each clock cycle.
Such a multi-stage system is known as a pipeline.
High-performance processors have a pipelined organization.
Pipelining enables the execution of successive instructions to be overlapped.



A hardware structure with multiple stages.

Five-step sequence of actions to fetch and execute an instruction.

A five-step sequence of actions to fetch and execute an instruction.



Step	Action
1	Fetch an instruction and increment the program counter.
2	Decode the instruction and read registers from the register file.
3	Perform an ALU operation.
4	Read or write memory data if the instruction involves a memory operand.
5	Write the result into the destination register, if needed.

Instruction Execution

Load Instructions

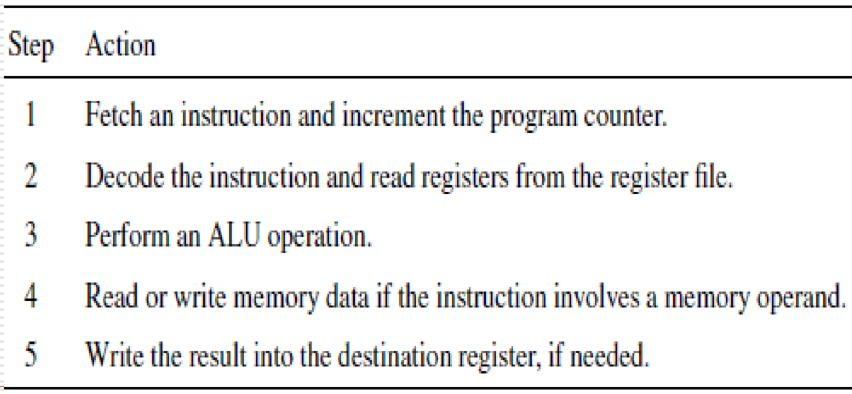
Consider the instruction

Load R5, X(R7)

which uses the Index addressing mode to load a word of data from memory location X + [R7] into register R5.

Fetch and Execution of this instruction involves the following actions:

1. Fetch the instruction and increment the program counter.
2. Decode the instruction and read the contents of register R7 in the register file.
3. Compute the effective address. X + [R7]
4. Read the memory source operand. read the contents of memory location pointed by R7 + X
5. Load the operand into the destination register, R5.



Step	Action
1	Fetch an instruction and increment the program counter.
2	Decode the instruction and read registers from the register file.
3	Perform an ALU operation.
4	Read or write memory data if the instruction involves a memory operand.
5	Write the result into the destination register, if needed.

Instruction Execution (Contd…)

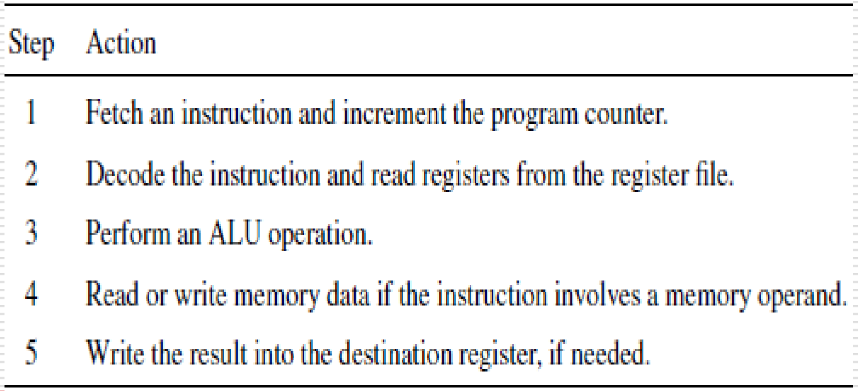
Arithmetic and Logic Instructions:

A typical instruction of this type is

Add R3, R4, R5 R3 == [R4] + [R5]

Fetch and Execution of this instruction involves the following actions:

1. Fetch the instruction and increment the program counter.
2. Decode the instruction and read registers R4 and R5.
3. Compute the sum [R4] + [R5].
4. No action.
5. Load the result into the destination register, R3.



Step	Action
1	Fetch an instruction and increment the program counter.
2	Decode the instruction and read registers from the register file.
3	Perform an ALU operation.
4	Read or write memory data if the instruction involves a memory operand.
5	Write the result into the destination register, if needed.

Instruction Execution (Contd…)

Store Instruction

Store R6, X(R8)

stores the contents of register R6 into memory location X + [R8].

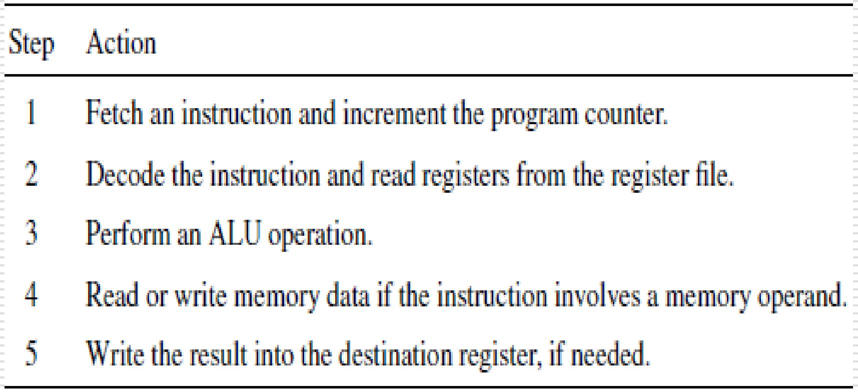
Fetch and Execution of this instruction involves the following actions:

1. Fetch the instruction and increment the program counter. 2. Decode the instruction and read registers R6 and R8.

3. Compute the effective address X + [R8].

4. Store the contents of register R6 into memory location X + [R8].

5. No action



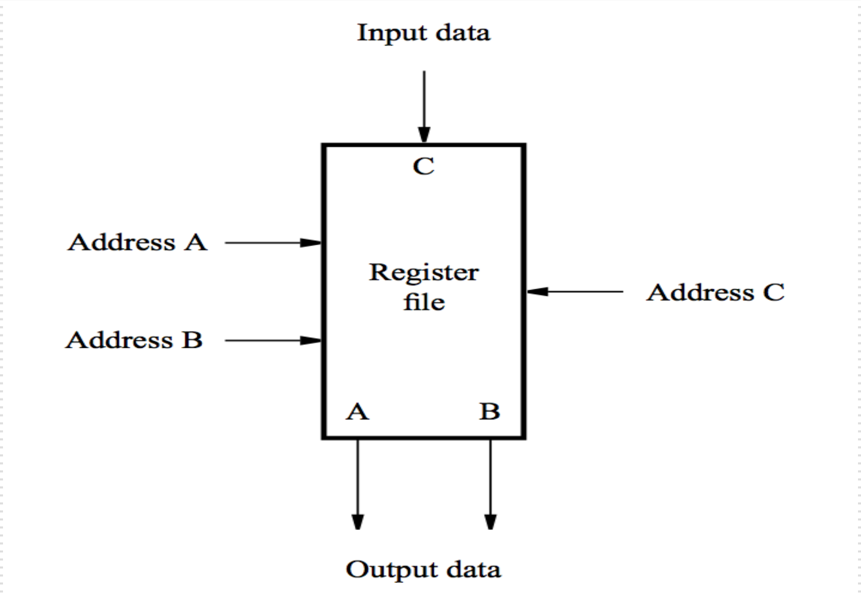
Step	Action
1	Fetch an instruction and increment the program counter.
2	Decode the instruction and read registers from the register file.
3	Perform an ALU operation.
4	Read or write memory data if the instruction involves a memory operand.
5	Write the result into the destination register, if needed.

Hardware Components

The discussion in previous slides indicates that all instructions of a RISC-style processor can be executed using the five-step sequence. Hence, the processor hardware may be organized in five stages, such that each stage performs the actions needed in one of the steps.

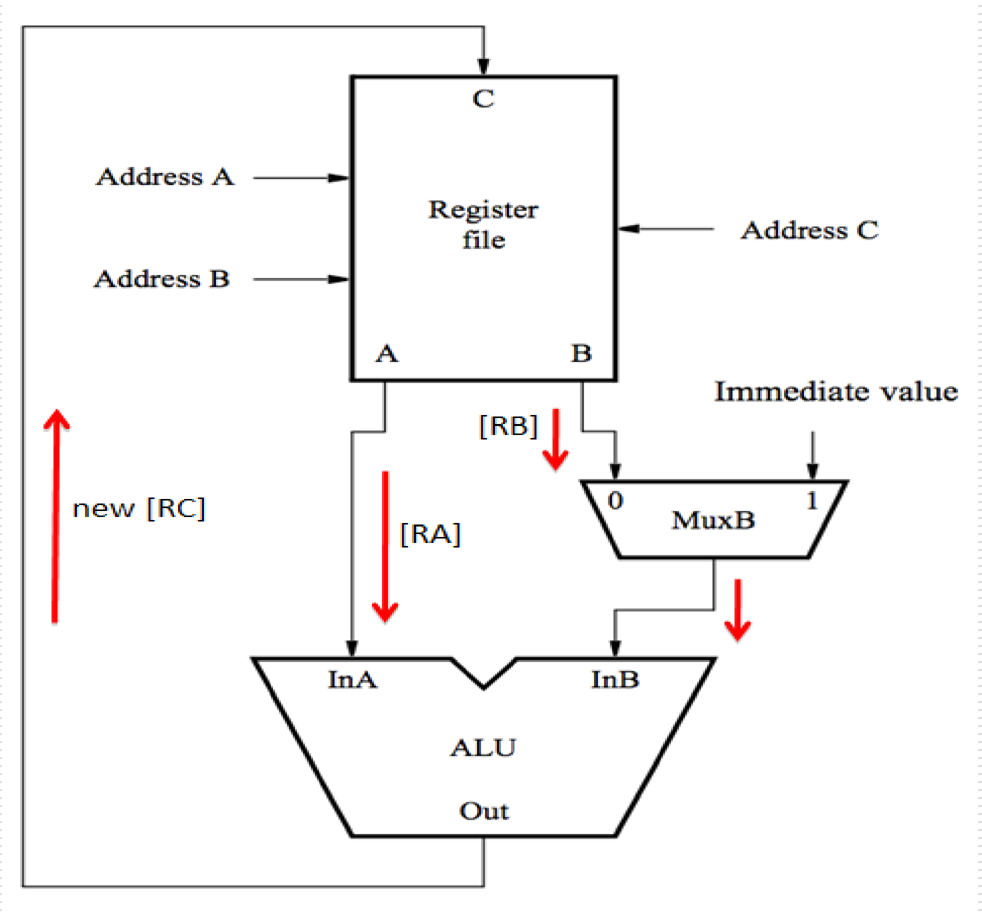
Register File

- A 2-port register file is needed to read the two source registers at the same time.
- It may be implemented using a 2-port memory.



Hardware Components: **ALU** (Contd…)

- Two source operands are from registers.

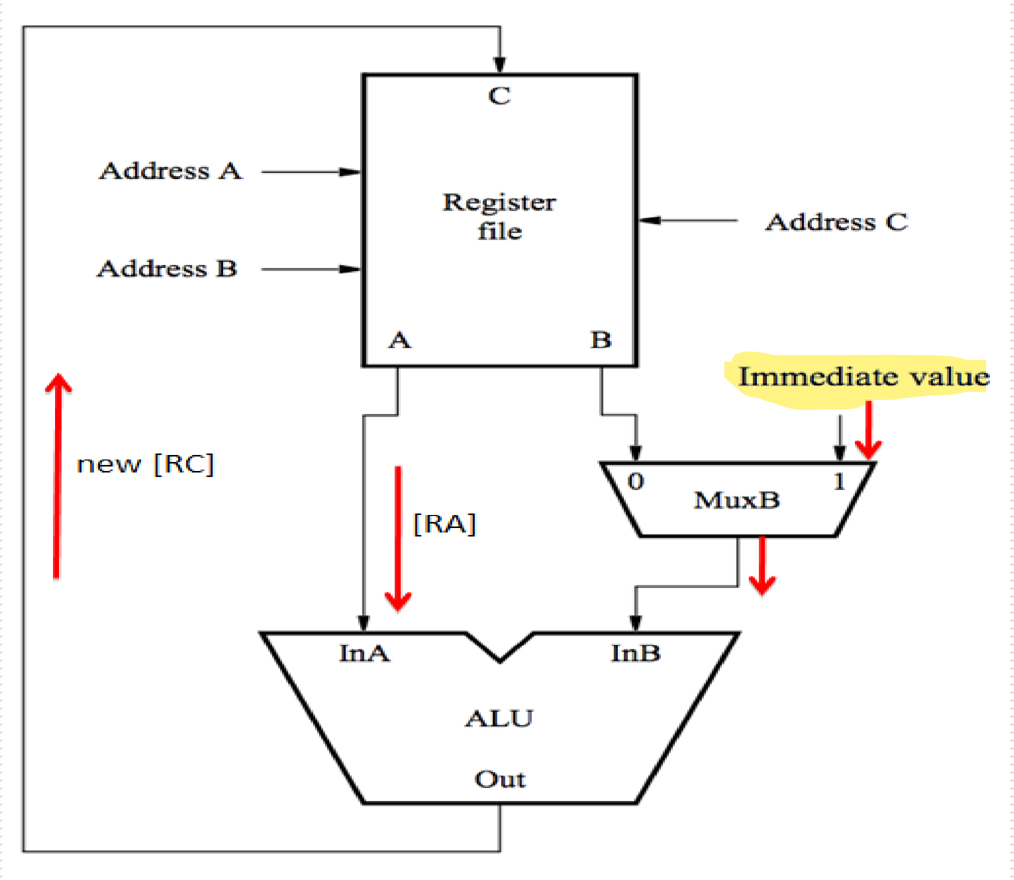


Conceptual view of the hardware needed for computation.

- Both source operands and the destination location are in the register file.
- [RA] and [RB] denote values of registers that are identified by addresses A and B
- new [RC] denotes the result that is stored to the register identified by address C

Hardware Components: **ALU** (Contd…)

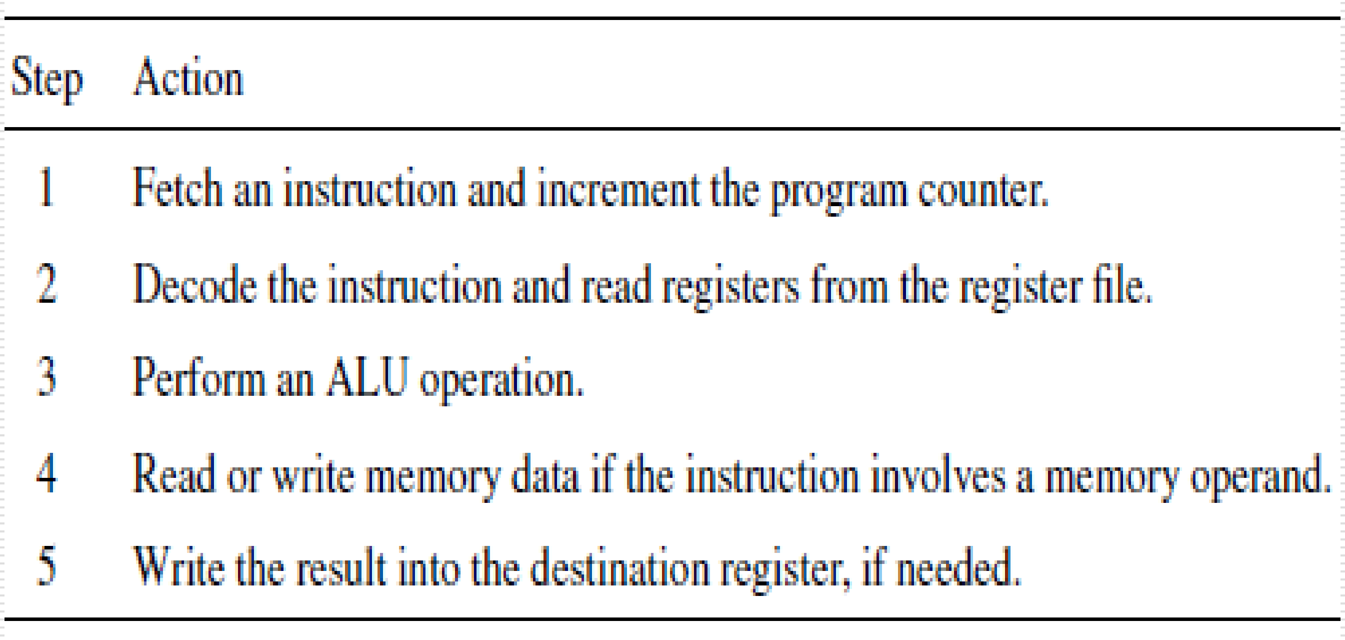
- One of the source operands is the immediate value in the IR.



Conceptual view of the hardware needed for computation.

Five-step sequence of actions to fetch and execute an instruction.

A five-step sequence of actions to fetch and execute an instruction.

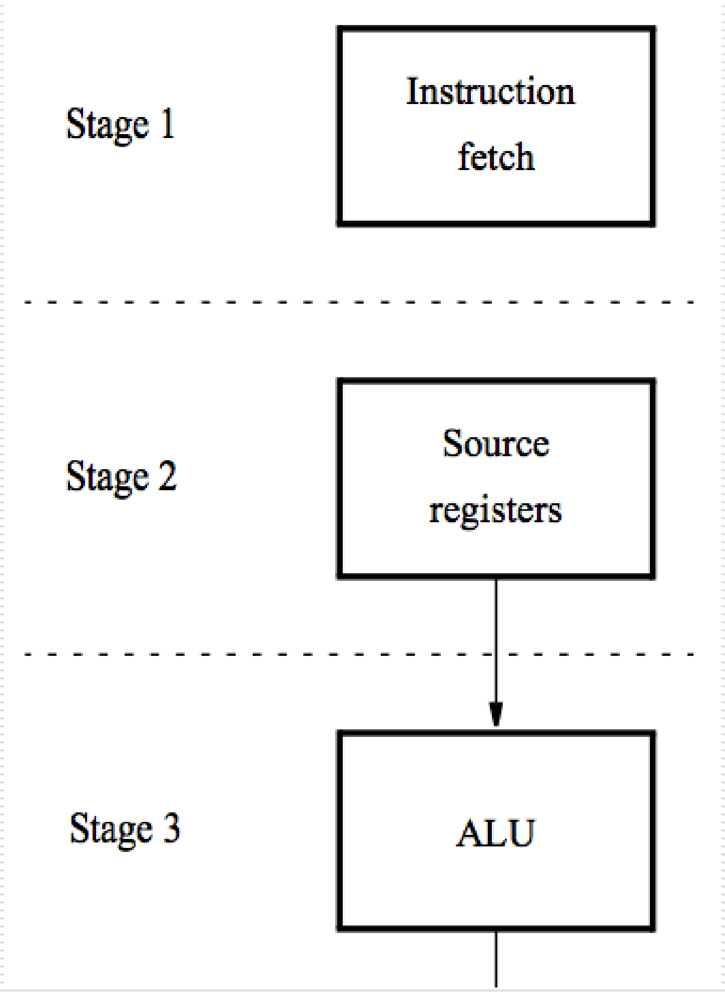


Step	Action
1	Fetch an instruction and increment the program counter.
2	Decode the instruction and read registers from the register file.
3	Perform an ALU operation.
4	Read or write memory data if the instruction involves a memory operand.
5	Write the result into the destination register, if needed.

Hardware Components: **DataPath**

A five-stage organization

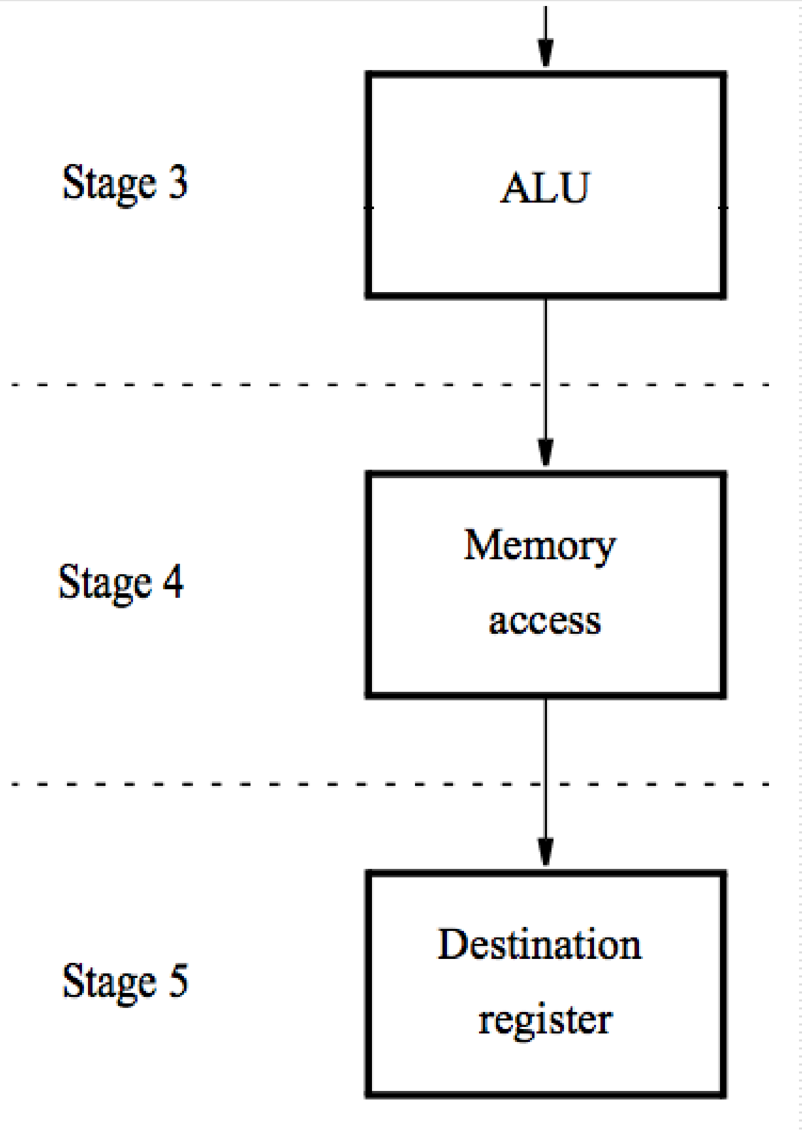
- Instruction processing moves from stage to stage in every clock cycle, starting with fetch.
- The instruction is decoded and the source registers are read in stage 2.
- Computation takes place in the ALU in stage 3.



Hardware Components: **DataPath**

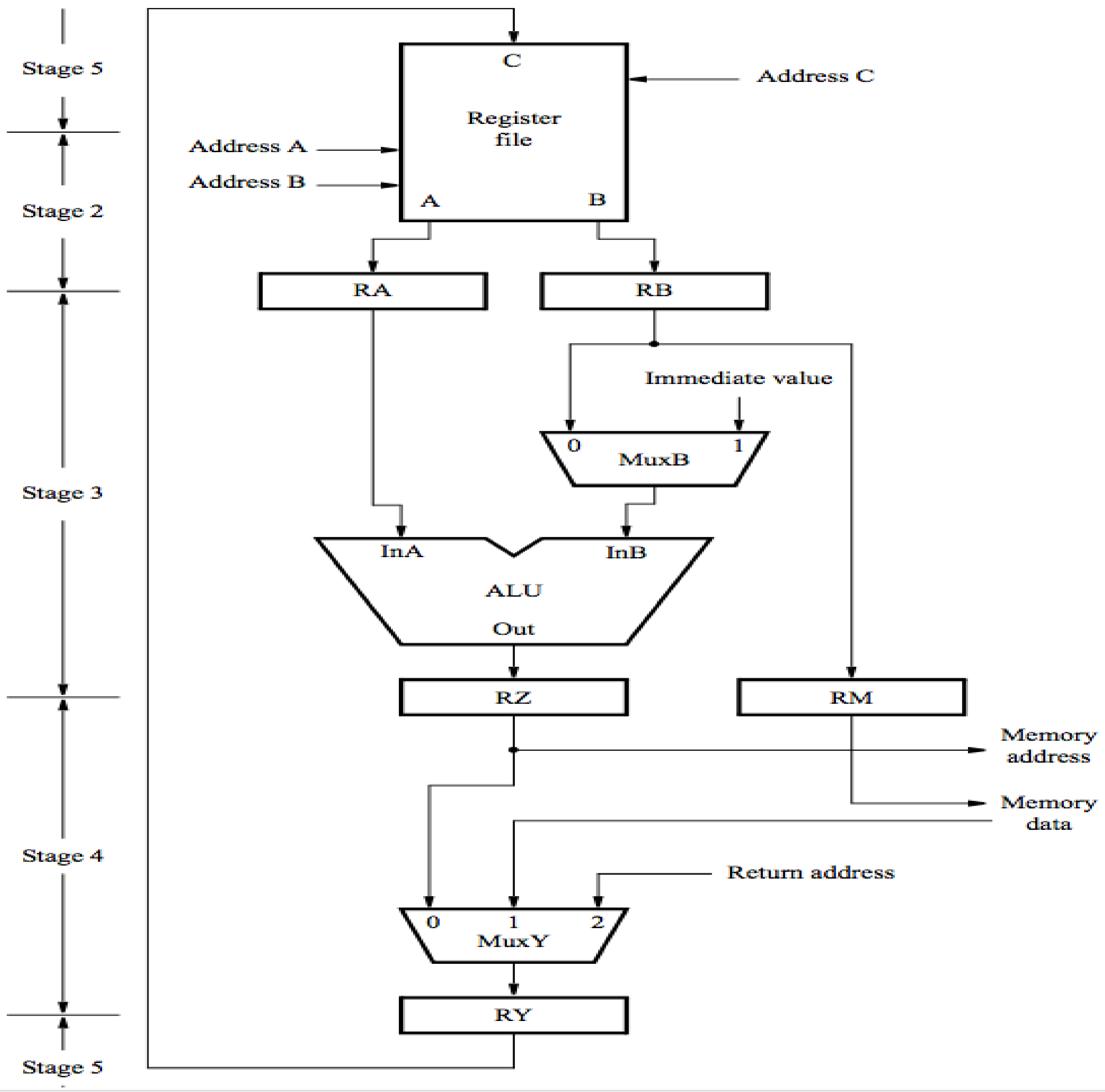
A five-stage organization

- If a memory operation is involved, it takes place in stage 4.
- The result of the instruction is stored in the destination register in stage 5.



Hardware Components: **DataPath**

Datapath in a processor



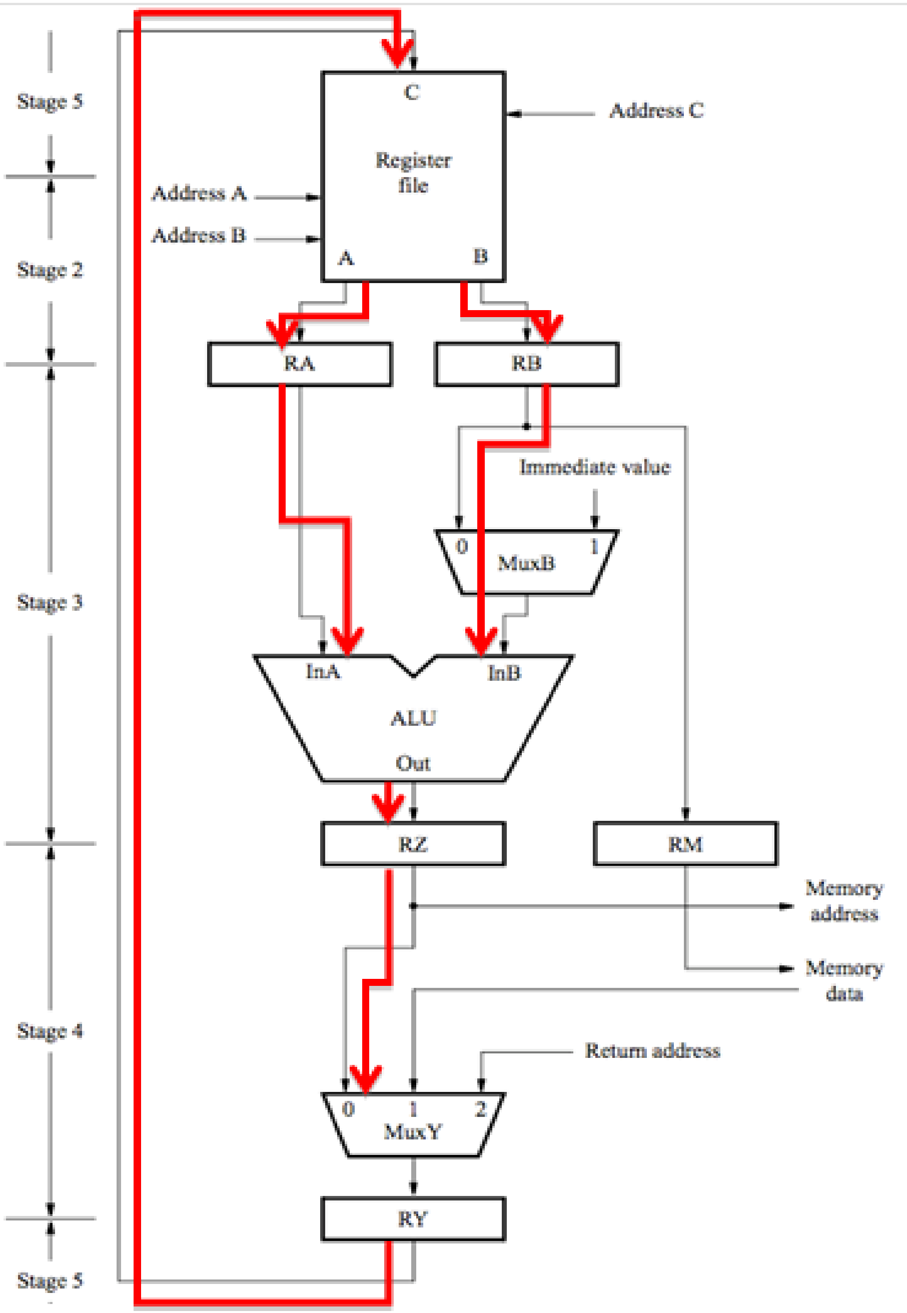
Hardware Components: **DataPath**

Sequence of actions needed
to fetch and execute the instruction:
Add R3, R4, R5.

Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction, RA ← [R4], RB ← [R5]
3	RZ ← [RA] + [RB]
4	RY ← [RZ]
5	R3 ← [RY]

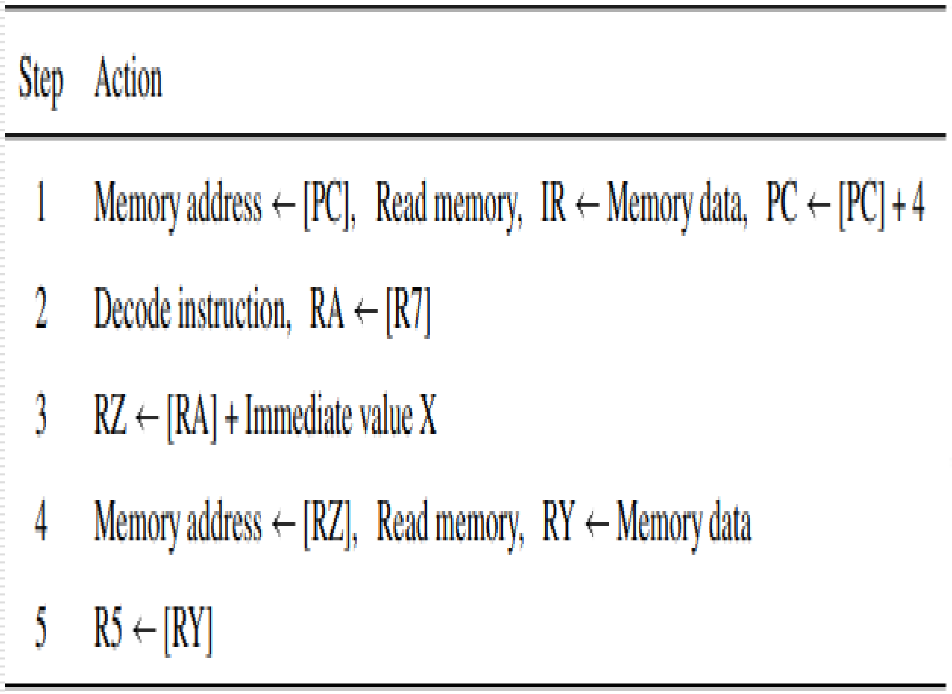
STEP 1:
memory address should be loaded by the PC
send READ signal
the instruction pointed by the PC is loaded into the IR
PC is incremented by 4

Datapath in a processor



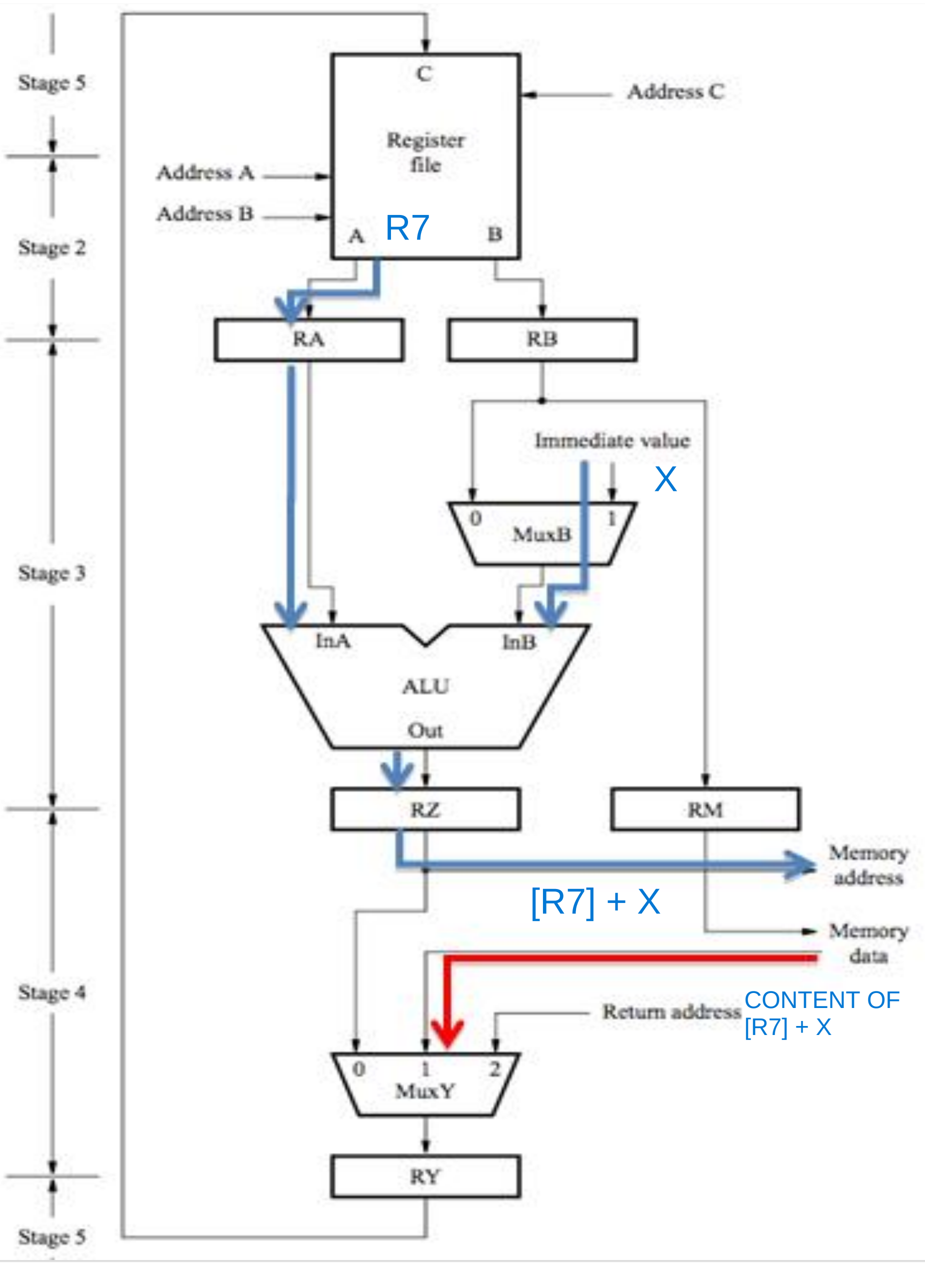
Hardware Components: **DataPath**

Sequence of actions needed to fetch and execute the instruction: **Load R5, X(R7)**.



Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction, RA ← [R7]
3	RZ ← [RA] + Immediate value X
4	Memory address ← [RZ], Read memory, RY ← Memory data
5	R5 ← [RY]

Datapath in a processor

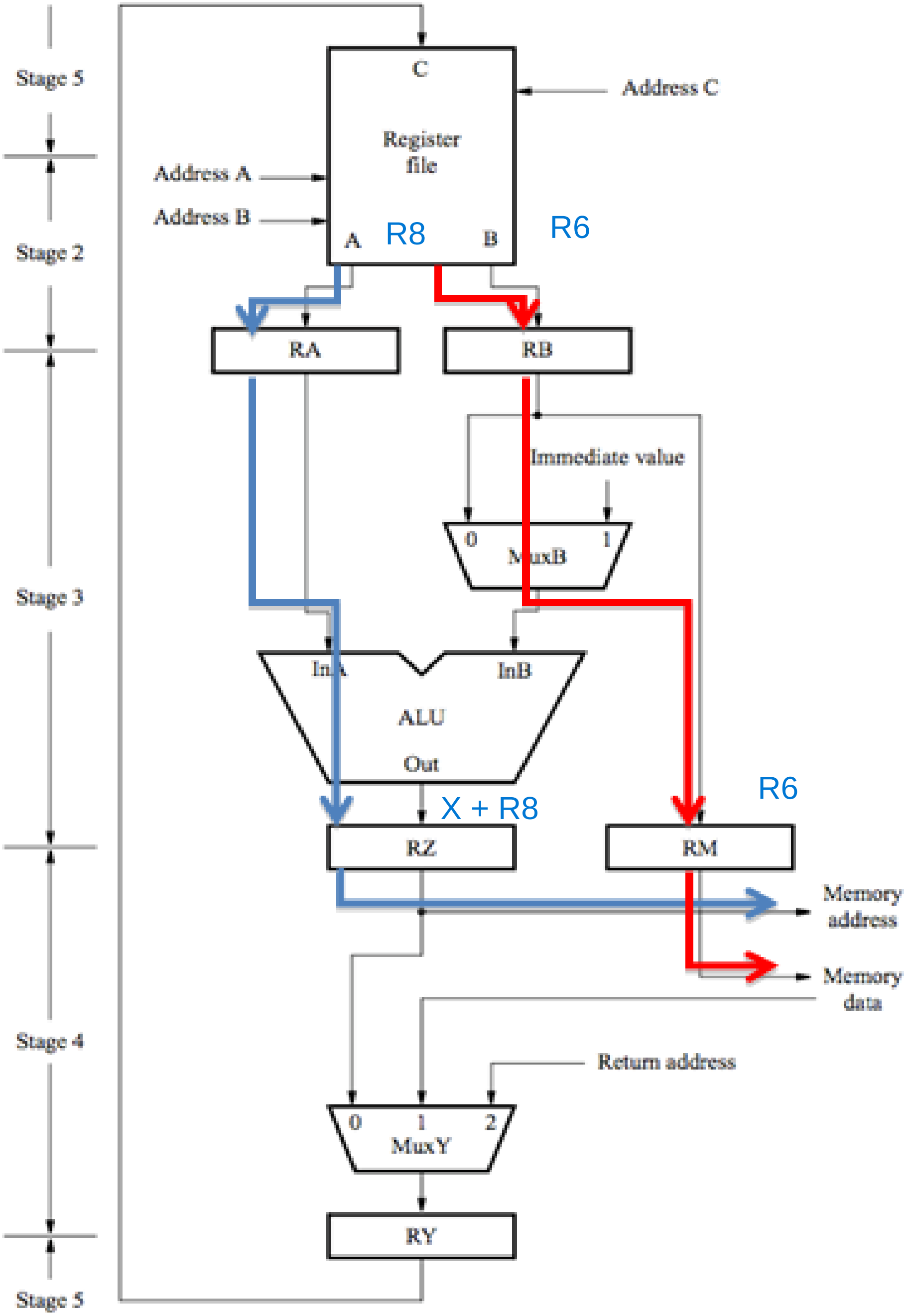


Hardware Components: **DataPath**

Sequence of actions needed to fetch and execute the instruction: **Store R6, X(R8)**.

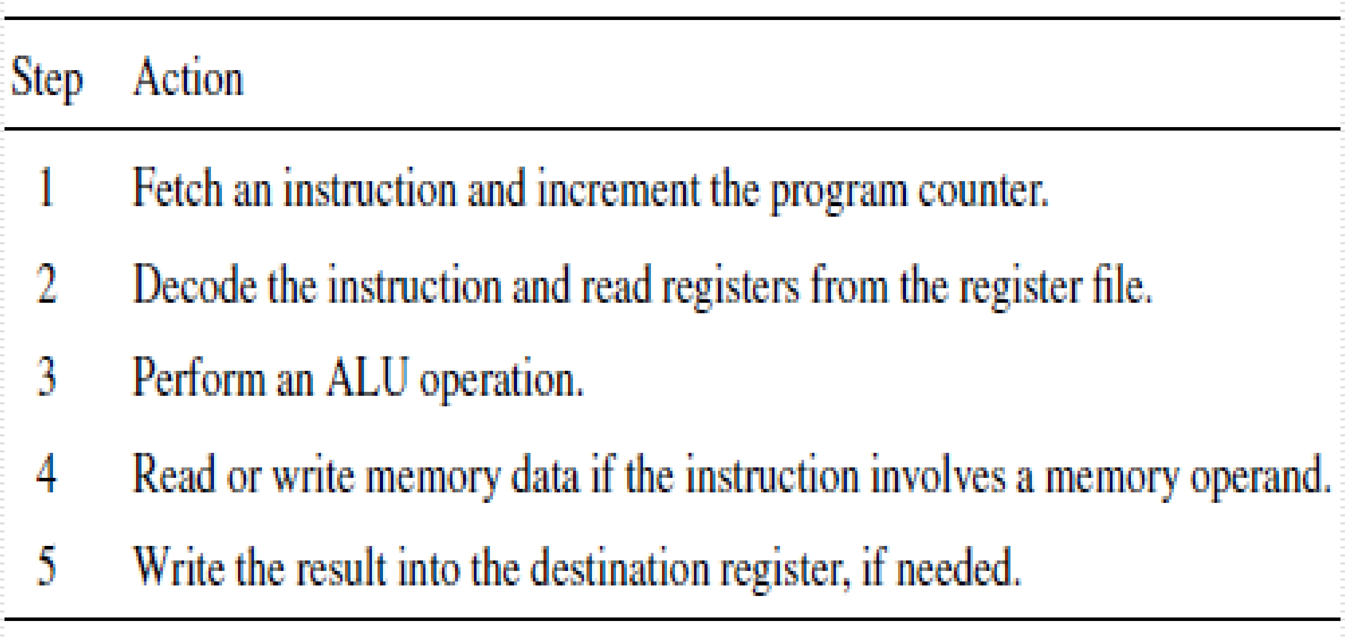
Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction, RA ← [R8], RB ← [R6]
3	RZ ← [RA] + Immediate value X, RM ← [RB]
4	Memory address ← [RZ], Memory data ← [RM], Write memory
5	No action

Datapath in a processor



Five-step sequence of actions to fetch and execute an instruction.

A five-step sequence of actions to fetch and execute an instruction.



Step	Action
1	Fetch an instruction and increment the program counter.
2	Decode the instruction and read registers from the register file.
3	Perform an ALU operation.
4	Read or write memory data if the instruction involves a memory operand.
5	Write the result into the destination register, if needed.

Hardware Components: **DataPath**

Sequence of actions needed to fetch and execute the instruction: **Unconditional Branch Instruction**

IR :: JMP NEXT_OFFSET

Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction
3	PC ← [PC] + Branch offset
4	No action
5	No action

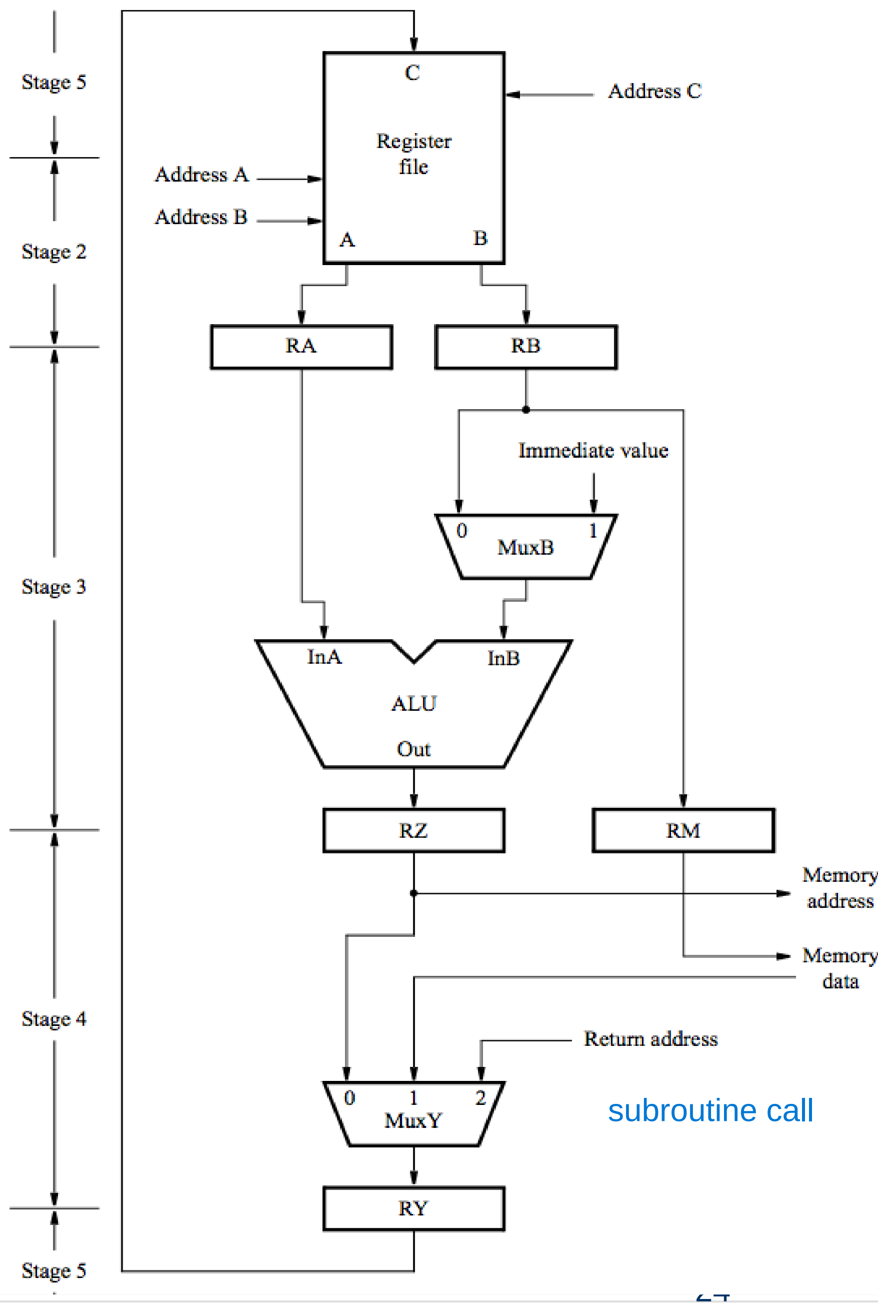
2004

PC == 2000

PC = 2008

2000 : JMP NEXT_OFFSET
2004 :
JMP_OFFSET: 2008

Datapath in a processor



subroutine call

Hardware Components: **DataPath**

Datapath in a processor

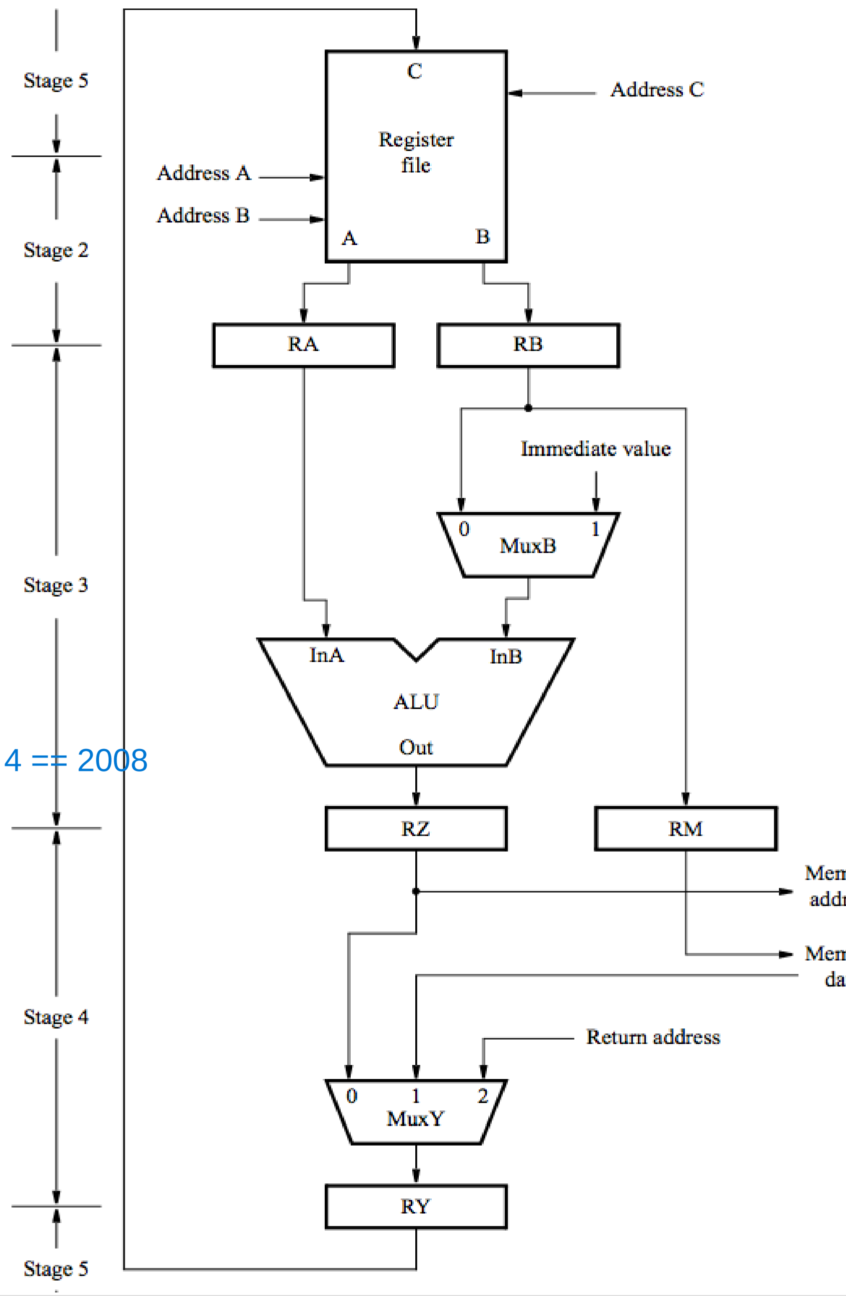
Sequence of actions needed to fetch and execute the instruction: **Conditional Branch Instruction**

Branch_if_[R5]=[R6] LOOP 2000

2004

Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction, RA ← [R5], RB ← [R6]
3	Compare [RA] to [RB], If [RA] = [RB], then PC ← [PC] + Branch offset
4	No action
5	No action

2004+ 4 == 2008



2000 : JMP NEXT_OFFSET
2004 :
JMP_OFFSET: 2008

Hardware Components: **DataPath**

Datapath in a processor

Sequence of actions needed to fetch and execute the instruction:

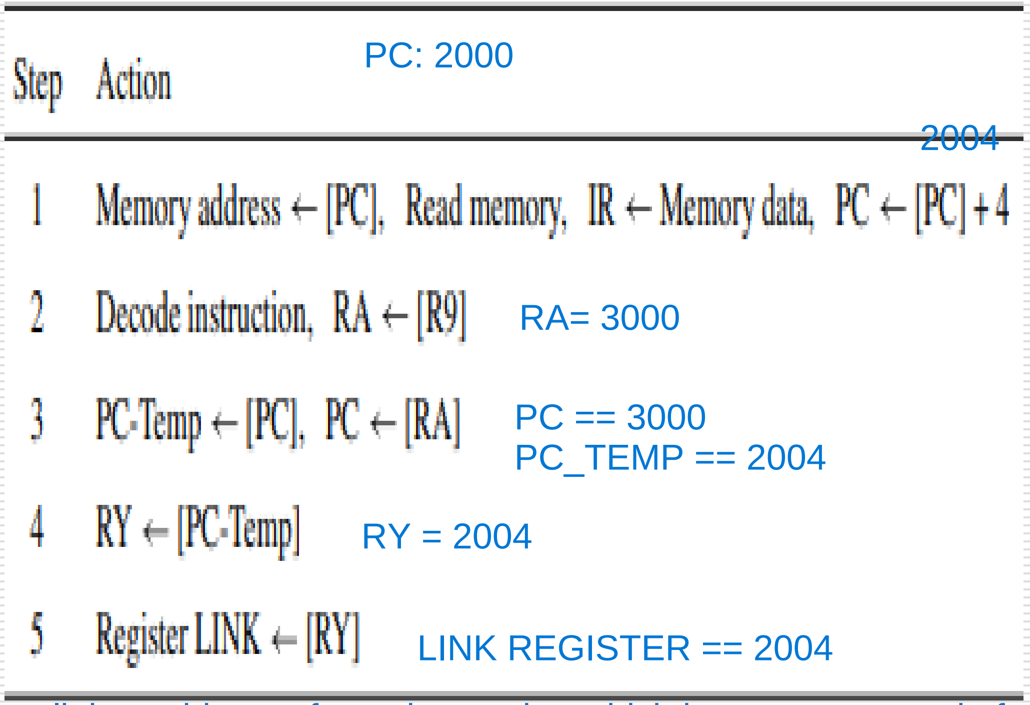
Subroutine Call Instructions

Call_Register R9

which calls a subroutine whose address is in register R9

2000: CALL R9

R9 SUBROUTINE 3000:



PC: 2000

2004

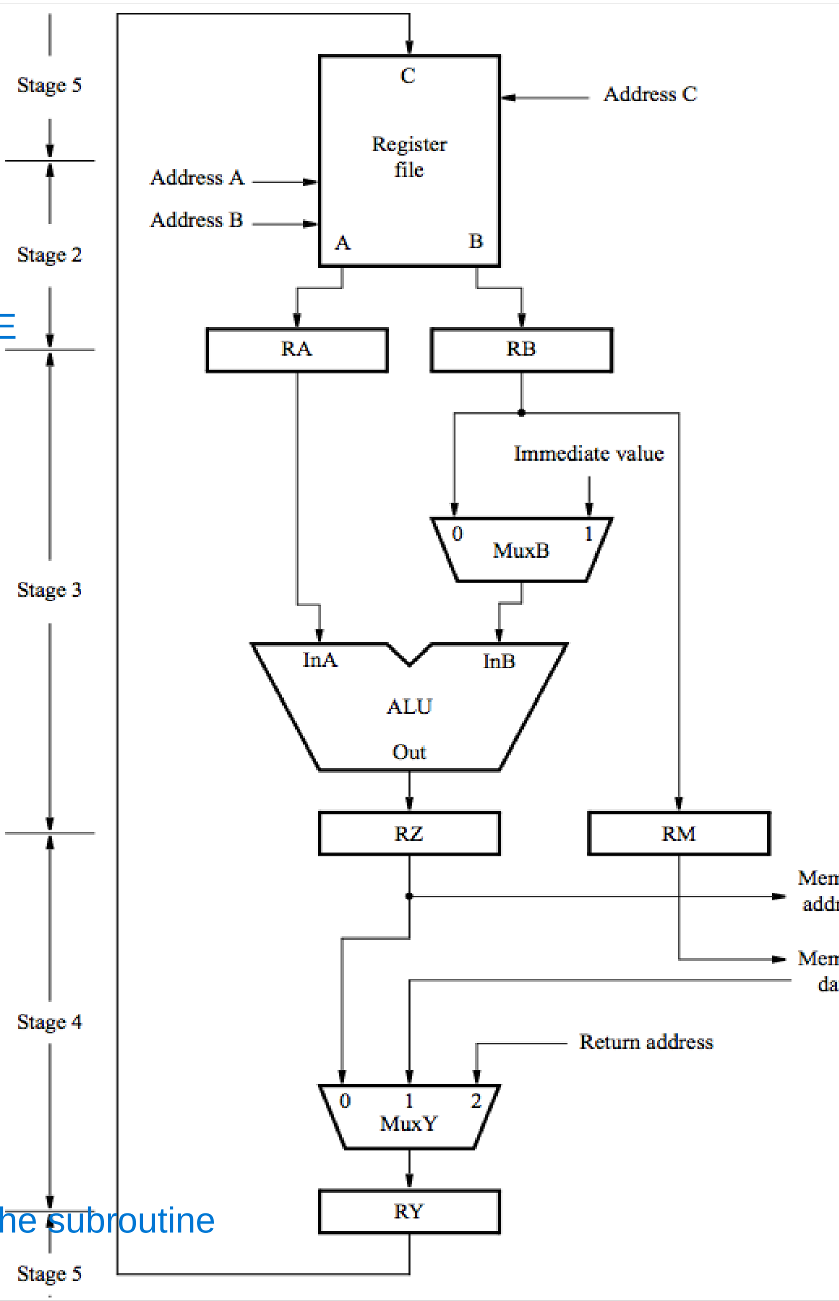
Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction, RA ← [R9]
3	PC-Temp ← [PC], PC ← [RA]
4	RY ← [PC-Temp]
5	Register LINK ← [RY]

RA= 3000

PC == 3000
PC_TEMP == 2004

RY = 2004

LINK REGISTER == 2004



link -- address of next instruction which is to get executed after the subroutine

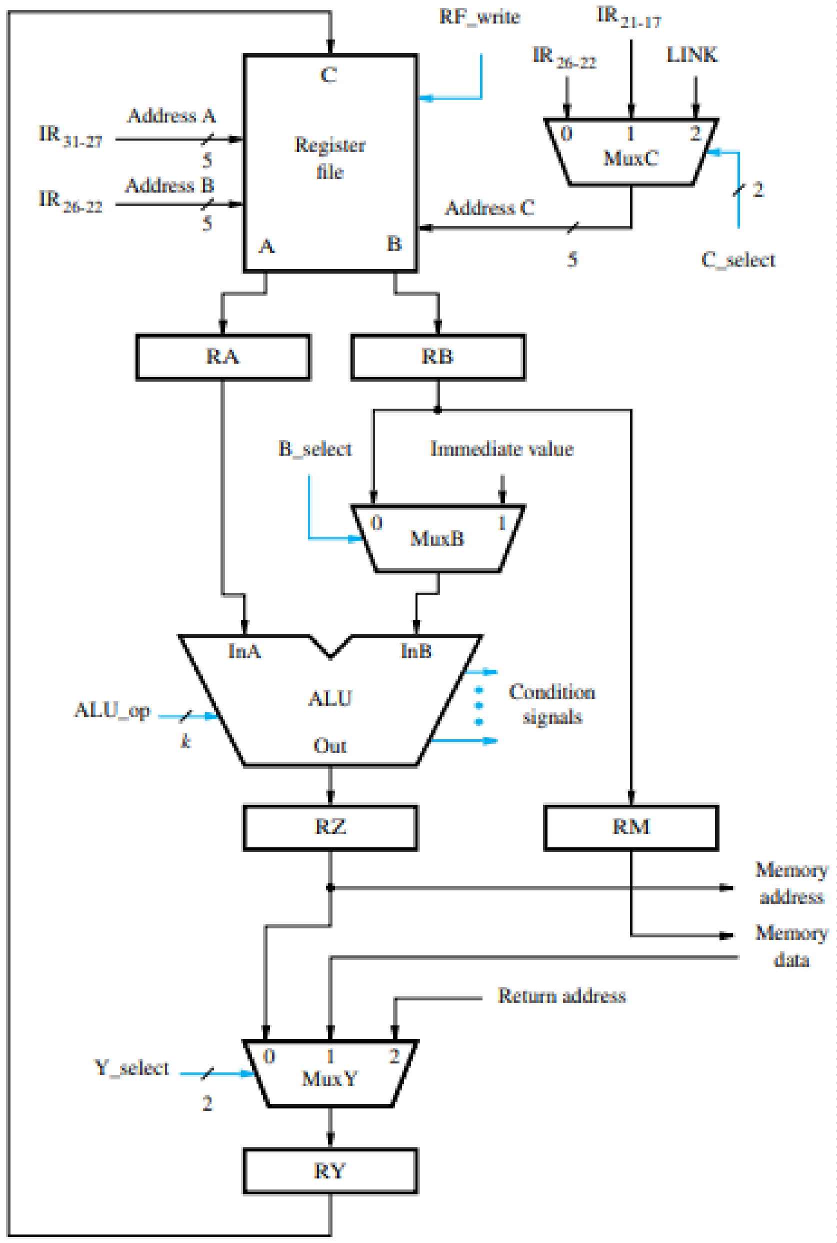
Control Signals

Select multiplexer inputs to guide the flow of data.

Set the function performed by the ALU.

Determine when data are written into the PC, the IR, the register file, and the memory.

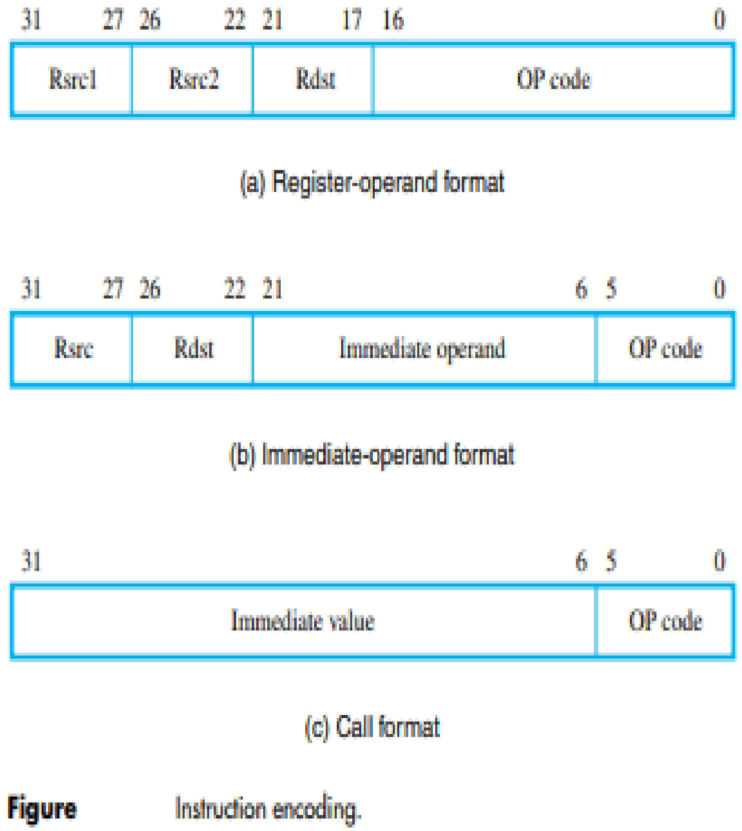
Inter-stage registers are always enabled because their contents are only relevant in the cycles for which the stages connected to the register outputs are active.



Control signals for the datapath

Rough Slide: to Explain Control Signals

Register File Control Signals



31 – 27	26 – 22	21 – 17	16 – 0
Rsrc1	Rsrc2	Rdst	OP code

(a) Register-operand format

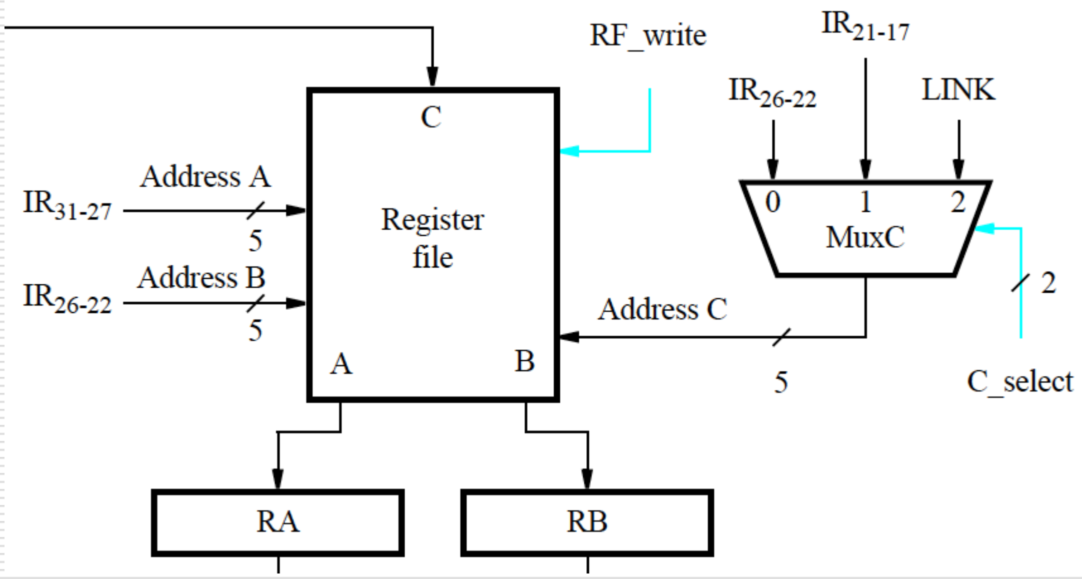
31 – 27	26 – 22	21 – 6	5 – 0
Rsrc	Rdst	Immediate operand	OP code

(b) Immediate-operand format

31 – 6	5 – 0
Immediate value	OP code

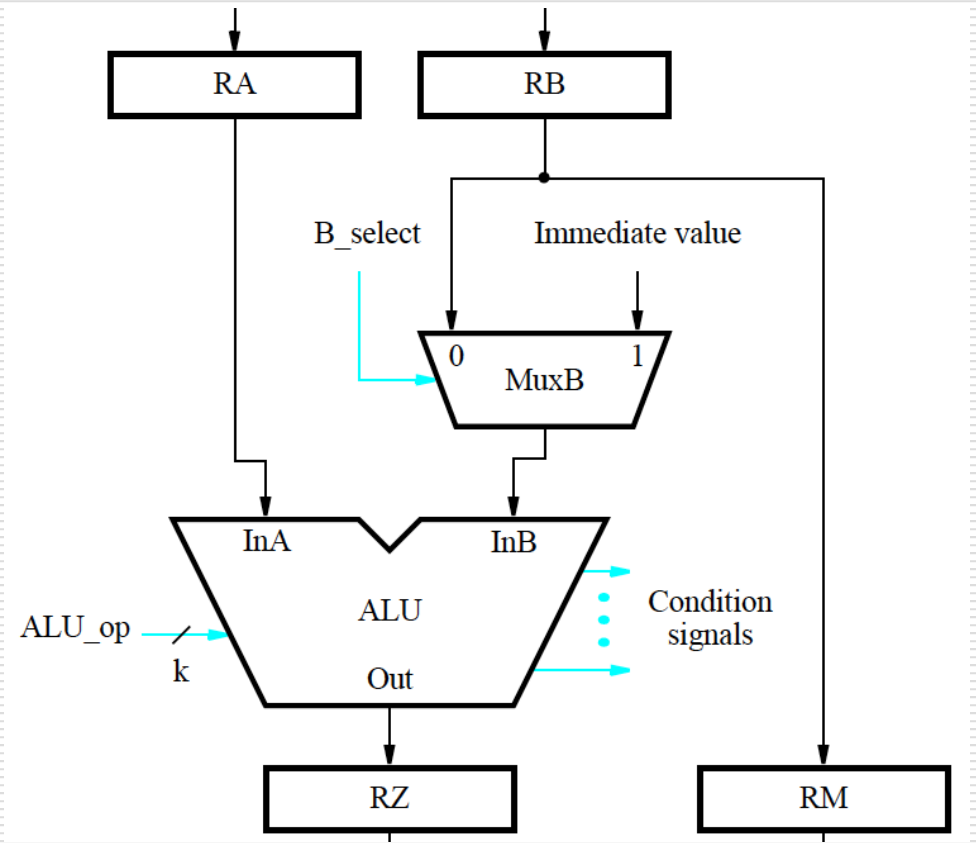
(c) Call format

Figure Instruction encoding.



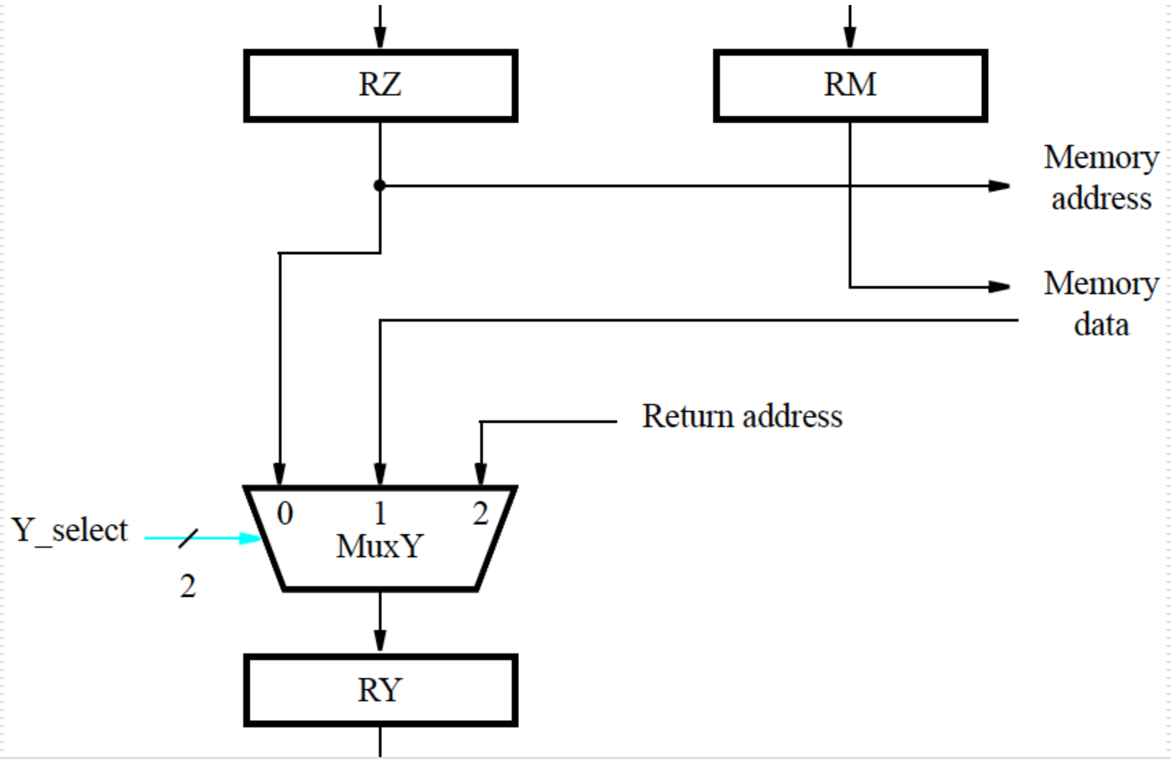
Rough Slide: to Explain Control Signals

ALU Control Signals



Rough Slide: to Explain Control Signals

Result Selection Signals

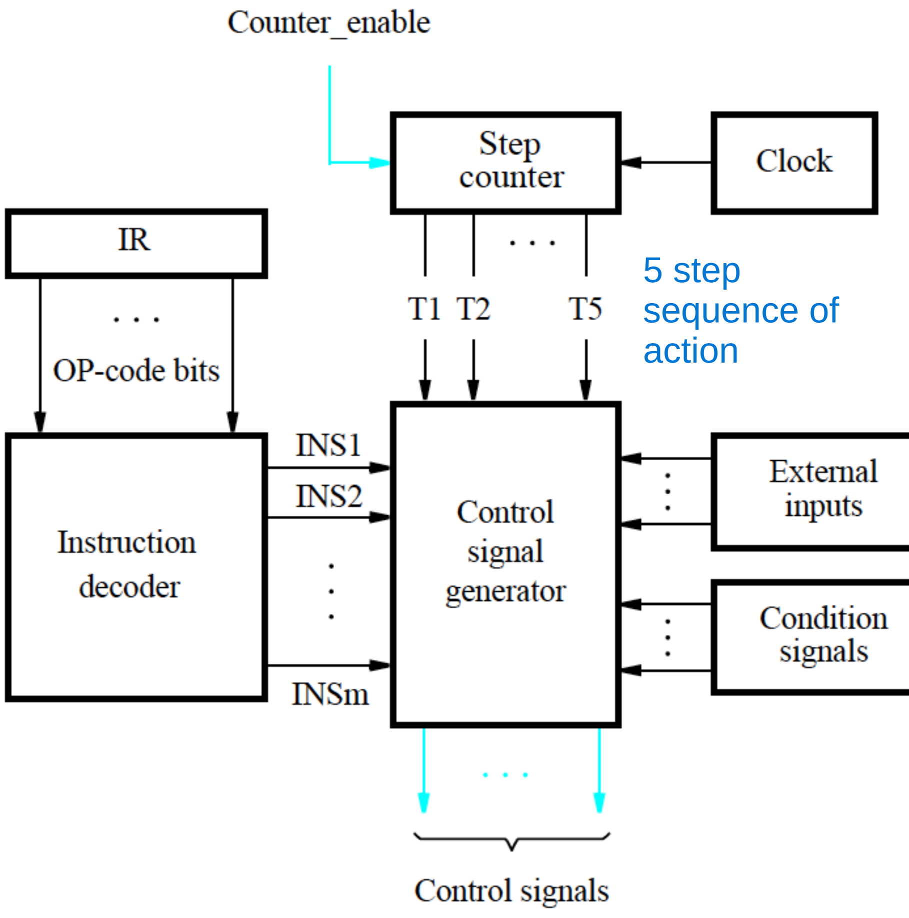


Control signal generation:

- Actions to fetch & execute instructions have been described.
- The necessary control signals have also been described.
- Circuitry must be implemented to generate control signals so actions take place in correct sequence and at correct time.
- There are two basic approaches:
  Hardwired control and Microprogramming

Hardwired Control

Hardwired control involves implementing circuitry that considers step counter, IR, ALU result, and external inputs.
Step counter keeps track of execution progress,
one clock cycle for each of the five steps described earlier (unless a memory access takes longer than one cycle).



Generation of the control signals.

Parallel Computer Architecture

Parallel Computer Architecture: Processor Architecture and Technology Trends, Flynn’s Taxonomy of Parallel Architectures, **Memory Organization of Parallel Computers:** Computers with Distributed Memory Organization, Computers with Shared Memory Organization,

Thread-Level Parallelism: Simultaneous Multithreading, Multicore Processors

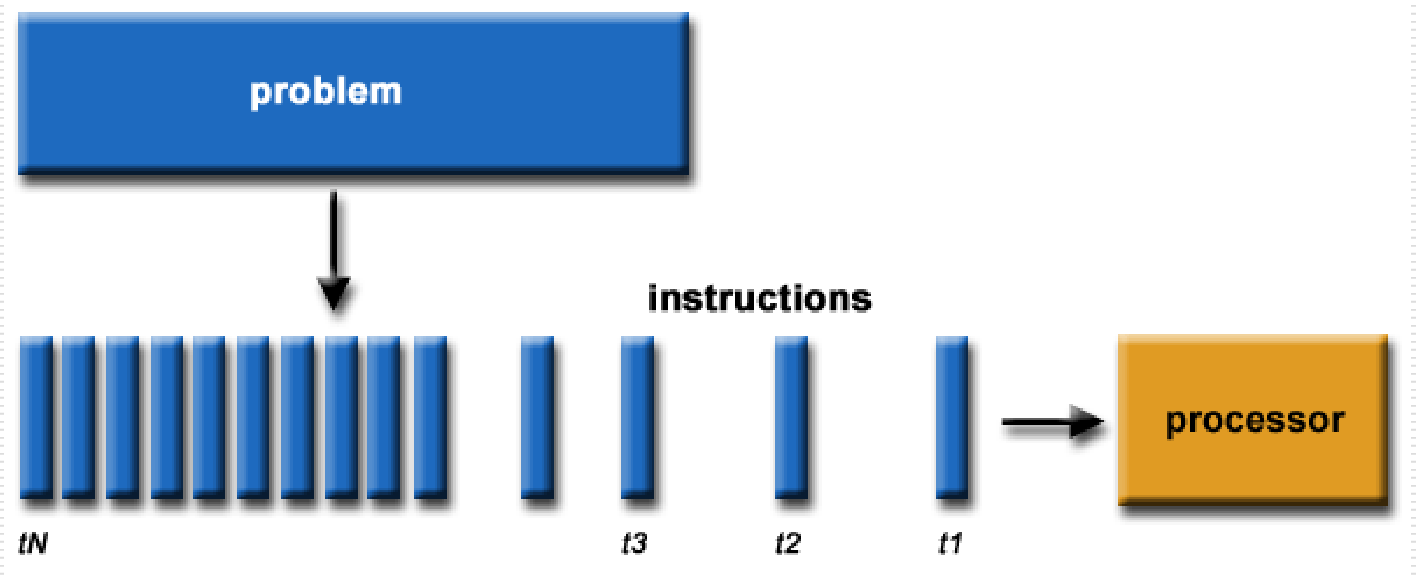
What is Parallel Computing ?

Serial vs Parallel Computing

Serial Computing

Traditionally, software has been written for ***serial*** computation: A problem is broken into a discrete series of instructions

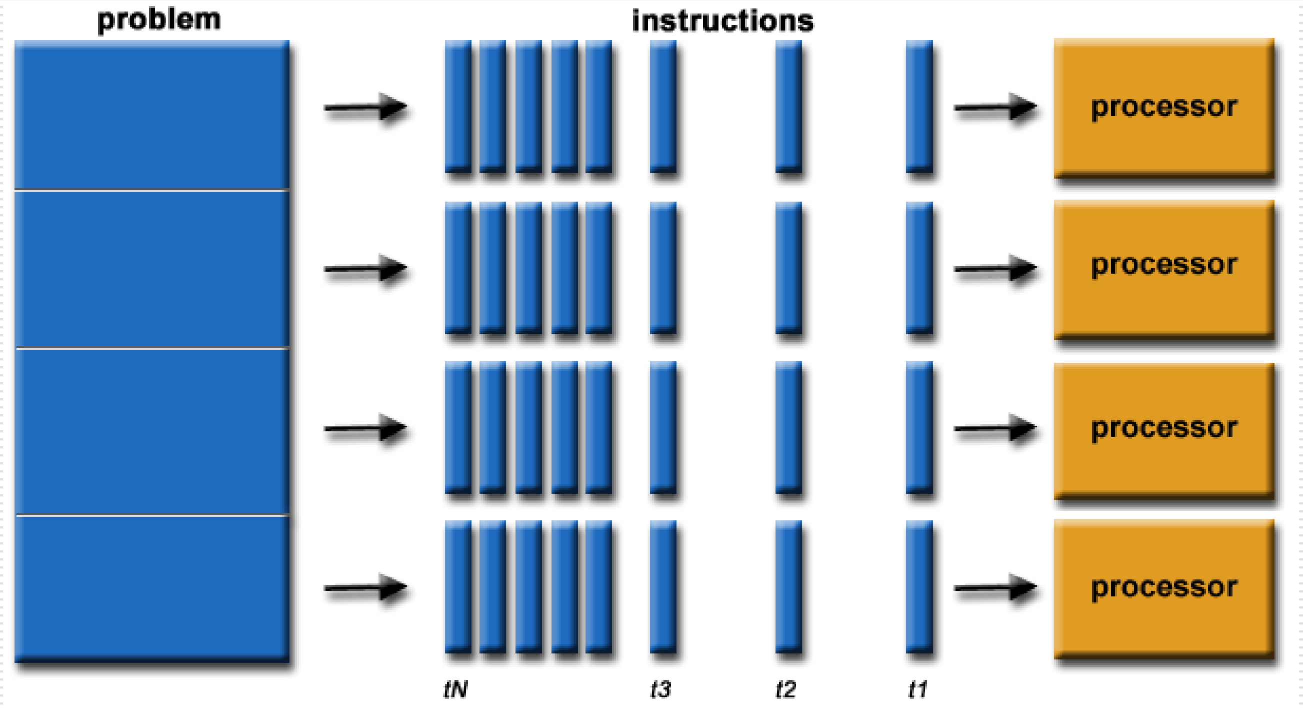
- Instructions are executed sequentially one after another
- Executed on a single processor
- Only one instruction may execute at any moment in time



Parallel Computing

In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem: A problem is broken into discrete parts that can be solved concurrently

- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed



Internal Parallelism Levels

Question:

List and Explain different levels of Internal Parallelism ?

Internal Parallelism Levels

1. Bit level parallelism
2. Parallelism by pipelining
3. Parallelism by multiple functional units
4. Parallelism at process or thread level

Internal Parallelism Levels

1. **Bit level parallelism:** 1970 to ~1985
   - 4 bits, 8 bit, 16 bit, 32, 64 bit microprocessors
2. **Parallelism by pipelining:** ~1985 through today

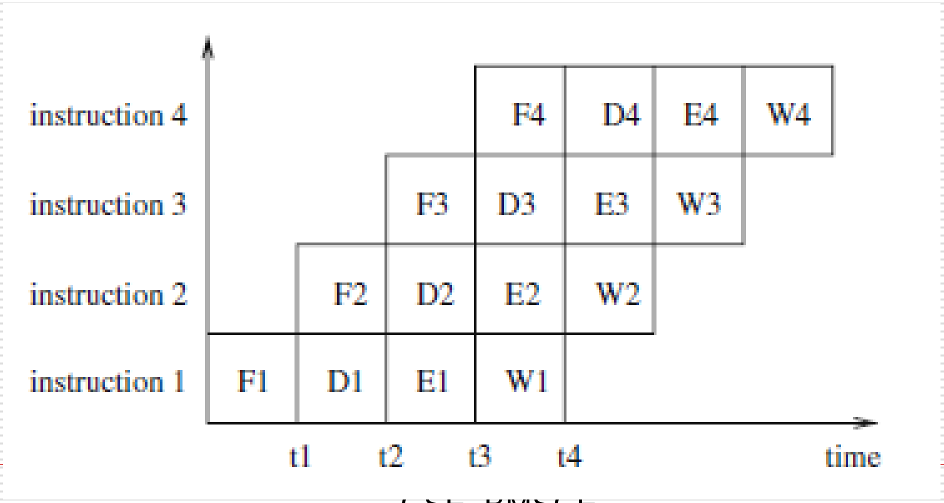
- Instruction level parallelism (ILP): The idea of pipelining at instruction level is an overlapping of the execution of multiple instructions.
- The execution of each instruction is partitioned into several steps which are performed by dedicated hardware units (pipeline stages) one after another. A typical partitioning could result in the following steps:

(a) *fetch*: fetch the next instruction to be executed from memory;

(b) *decode*: decode the instruction fetched in step (a);

(c) *execute*: load the operands specified and execute the instruction;

(d) *write-back*: write the result into the target register/memory.



Internal Parallelism Levels (Contd…)

3. Parallelism by multiple functional units:

- Many processors are *multiple-issue processors*.
- They use multiple, independent functional units like ALUs (*arithmetic logical units*), FPUs (*floating-point units*), load/store units, or branch units.
- These units can work in parallel, i.e., different independent instructions can be executed in parallel by different functional units. Thus, the average execution rate of instructions can be increased.

4. Parallelism at process or thread level:

- The three techniques described so far assume a *single sequential* control flow which is provided by the compiler and which determines the execution order if there are dependencies between instructions.
- An alternative approach is to use add multiple, independent processor cores onto a single processor chip.
- This approach has been used for typical desktop processors since 2005. The resulting processor chips are called **multicore processors**. Example: Dual Core, Quadcore processor

Flynn’s Taxonomy of Parallel Computers

Question:

List and Explain different classifications of Parallel Computer according to Flynn’s Taxonomy ?

Flynn’s Taxonomy of Parallel Computers

Flynn’s Taxonomy: Classification according to important characteristics of a parallel computer.

Four categories are distinguished based on:

- How Many Instruction Streams
- How Many Data Streams

1. **Single-Instruction, Single-Data (SISD)**
2. **Multiple-Instruction, Single-Data (MISD)**
3. **Single-Instruction, Multiple-Data (SIMD)**
4. **Multiple-Instruction, Multiple-Data (MIMD)**

Flynn's Taxonomy of Parallel Computers

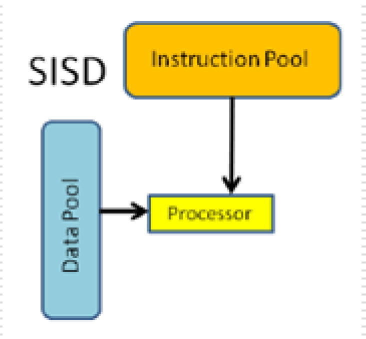
Flynn's Taxonomy: Classification according to important characteristics of a parallel computer.

Four categories are distinguished based on:

- How Many Instruction Streams
- How Many Data Streams

1. **Single-Instruction, Single-Data (SISD)**
2. **Multiple-Instruction, Single-Data (MISD)**
3. **Single-Instruction, Multiple-Data (SIMD)**
4. **Multiple-Instruction, Multiple-Data (MIMD)**

	Instruction Stream	**Data Stream**
SSID	1	1
MISD		
SIMD		
MIMD		



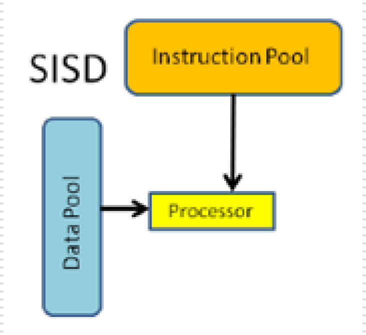
Flynn’s Taxonomy of Parallel Computers

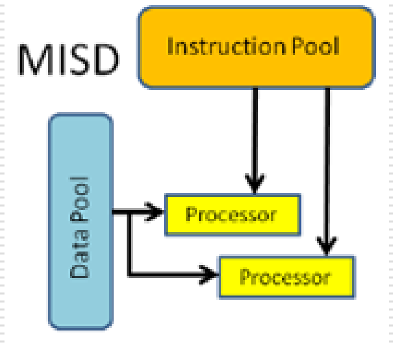
Flynn’s Taxonomy: Classification according to important characteristics of a parallel computer.
Four categories are distinguished based on:

- How Many Instruction Streams
- How Many Data Streams

1. **Single-Instruction, Single-Data (SISD)**
2. **Multiple-Instruction, Single-Data (MISD)**
3. **Single-Instruction, Multiple-Data (SIMD)**
4. **Multiple-Instruction, Multiple-Data (MIMD)**

	Instruction Stream	**Data Stream**
SSID	1	1
MISD	>1	1
SIMD		
MIMD		





Flynn’s Taxonomy of Parallel Computers

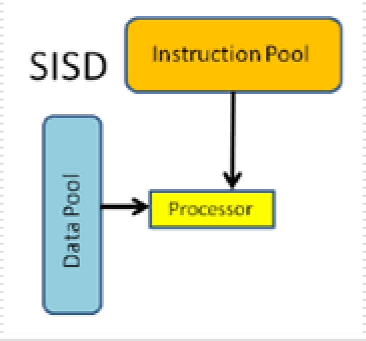
Flynn’s Taxonomy: Classification according to important characteristics of a parallel computer.

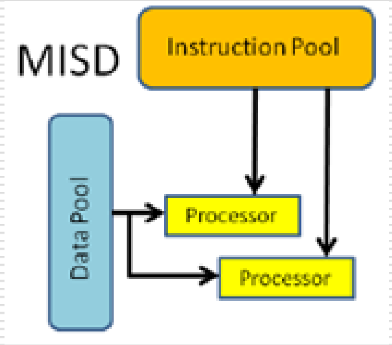
Four categories are distinguished based on:

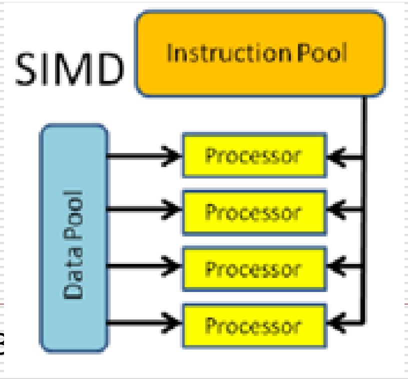
- How Many Instruction Streams
- How Many Data Streams

1. **Single-Instruction, Single-Data (SISD)**
2. **Multiple-Instruction, Single-Data (MISD)**
3. **Single-Instruction, Multiple-Data (SIMD)**
4. **Multiple-Instruction, Multiple-Data (MIMD)**

	Instruction Stream	**Data Stream**
SSID	1	1
MISD	>1	1
SIMD	1	>1
MIMD		







Flynn’s Taxonomy of Parallel Computers

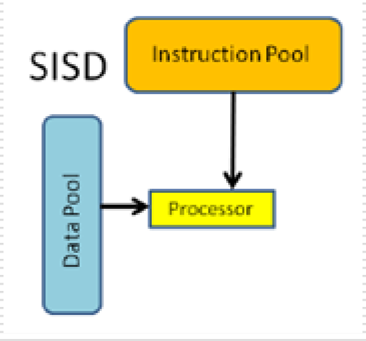
Flynn’s Taxonomy: Classification according to important characteristics of a parallel computer.

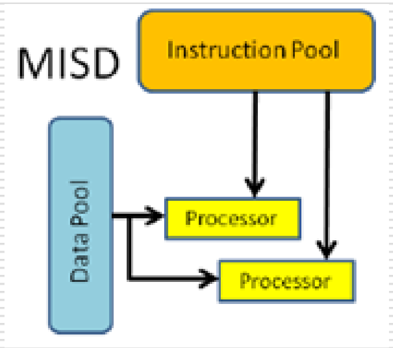
Four categories are distinguished based on:

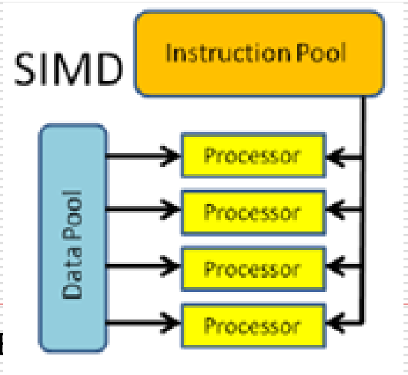
- How Many Instruction Streams
- How Many Data Streams

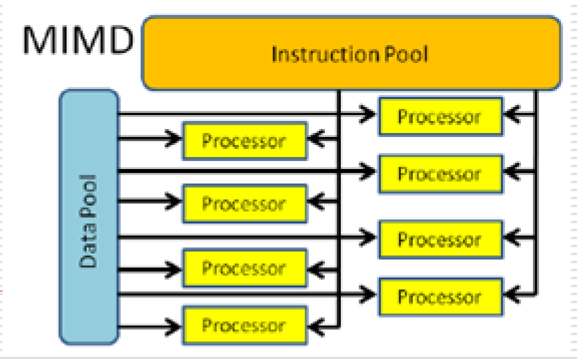
1. **Single-Instruction, Single-Data (SISD)**
2. **Multiple-Instruction, Single-Data (MISD)**
3. **Single-Instruction, Multiple-Data (SIMD)**
4. **Multiple-Instruction, Multiple-Data (MIMD)**

	Instruction Stream	**Data Stream**
SSID	1	1
MISD	>1	1
SIMD	1	>1
MIMD	>1	>1









Flynn’s Taxonomy of Parallel Computers (Contd…)

1. **Single-Instruction, Single-Data (SISD)**: There is one processing element which has access to a single program and data storage. In each step, the processing element loads an instruction and the corresponding data and executes the instruction. The result is stored back in the data storage. Thus, SISD is the conventional sequential computer according to the *von Neumann model*.

2. **Multiple-Instruction, Single-Data (MISD)**: There are multiple processing elements each of which has a private program memory, but there is only one common access to a single global data memory. In each step, each processing element obtains the *same* data element from the data memory and loads an instruction from its private program memory. These possibly different instructions are then executed in parallel by the processing elements using the previously obtained (identical) data element as operand. This execution model is very restrictive and no commercial parallel computer of this type has ever been built.

Flynn’s Taxonomy of Parallel Computers (Contd…)

3. **Single-Instruction, Multiple-Data (SIMD)**: There are multiple processing elements each of which has a private access to a (shared or distributed) data memory. But there is only one program memory from which a special control processor fetches and dispatches instructions. In each step, each processing element obtains from the control processor the *same* instruction and loads a separate data element through its private data access on which the instruction is performed. Thus, the instruction

is synchronously applied in parallel by all processing elements to different data elements.

For applications with a significant degree of data parallelism, the SIMD approach can be very efficient. Examples are multimedia applications or computer graphics algorithms to generate realistic three-dimensional views of computer-generated environments.

4. **Multiple-Instruction, Multiple-Data (MIMD)**: There are multiple processing elements each of which has a separate instruction and data access to a (shared or distributed) program and data memory. In each step, each processing element loads a separate instruction and a separate data element, applies the instruction to the data element, and stores a possible result back into the data storage. The processing elements work asynchronously with each other. Multicore processors or cluster systems are examples for the MIMD model.

Memory Organization of Parallel Computers

A Further classification of MIMD computers can be done according to their memory organization:

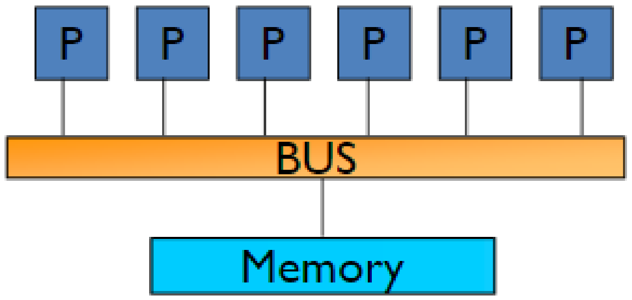
1. Computers with Distributed Memory Organization

2. Computers with Shared Memory Organization

Shared vs. Distributed Memory

The simplest and most useful way to classify modern parallel computers is by their memory model:

Shared memory and Distributed memory



Shared memory - single address space. All processors have access to a pool of shared memory.

Distributed memory - each processor has it's own local memory. Must do message passing to exchange data between processors.

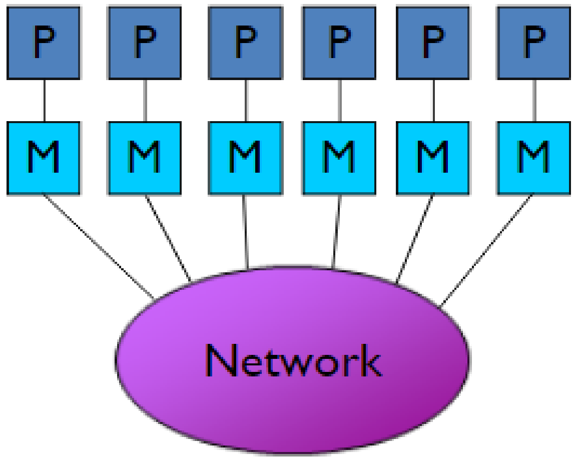
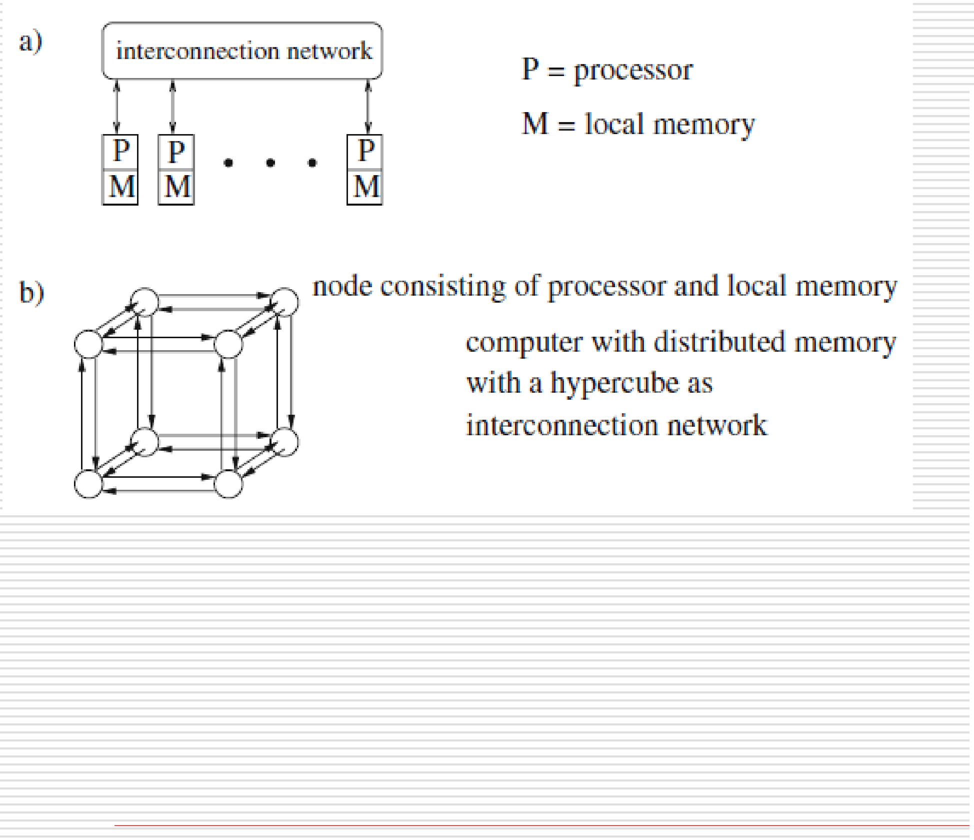


Illustration of computers with distributed memory

Illustration of computers with distributed memory: (**a**) abstract structure, (**b**) computer with distributed memory and hypercube as interconnection structure, (**c**) DMA (direct memory access), (**d**) processor–memory node with router, and (**e**) interconnection network in the form of a mesh to connect the routers of the different processor–memory nodes


a)
interconnection network
P
M
P = processor
M = local memory
b)
node consisting of processor and local memory
computer with distributed memory with a hypercube as interconnection network
c)
interconnection network
DMA
M
P
DMA (direct memory access) with DMA connections to the network
d)
P
M
Router
external input channels
external output channels
e)
N
R
R = Router
N = node consisting of processor and local memory

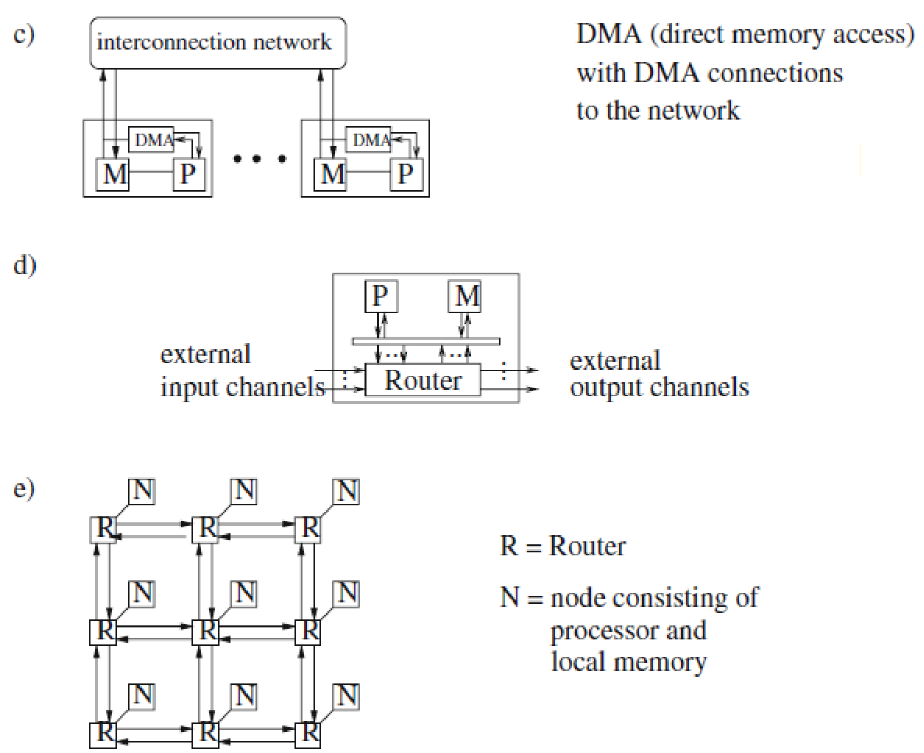


Illustration of a computer with shared memory

Illustration of a computer with shared memory: (**a**) abstract view and (**b**) implementation of the shared memory with memory modules

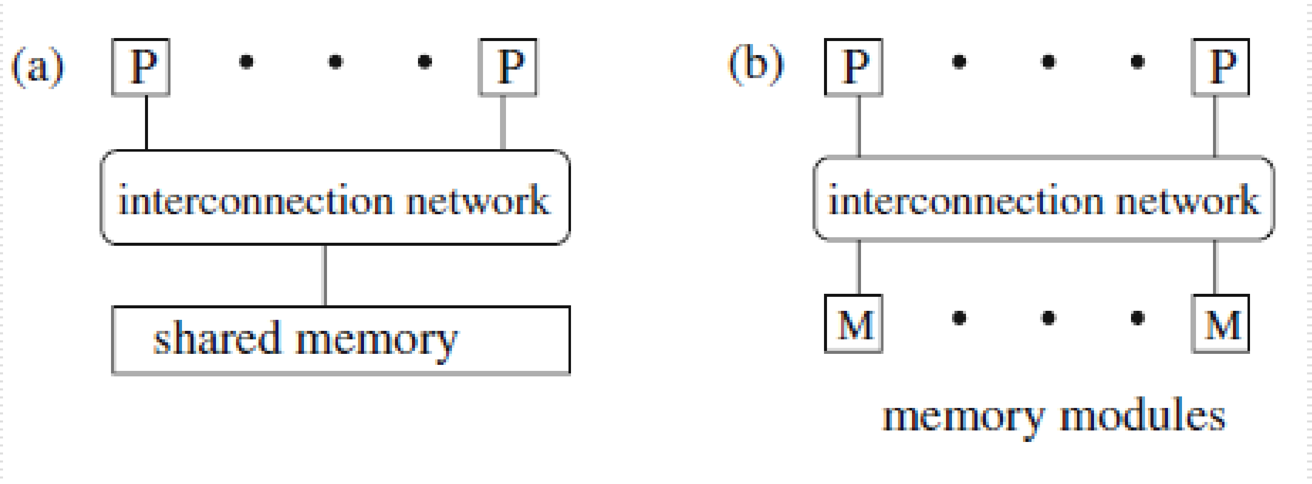
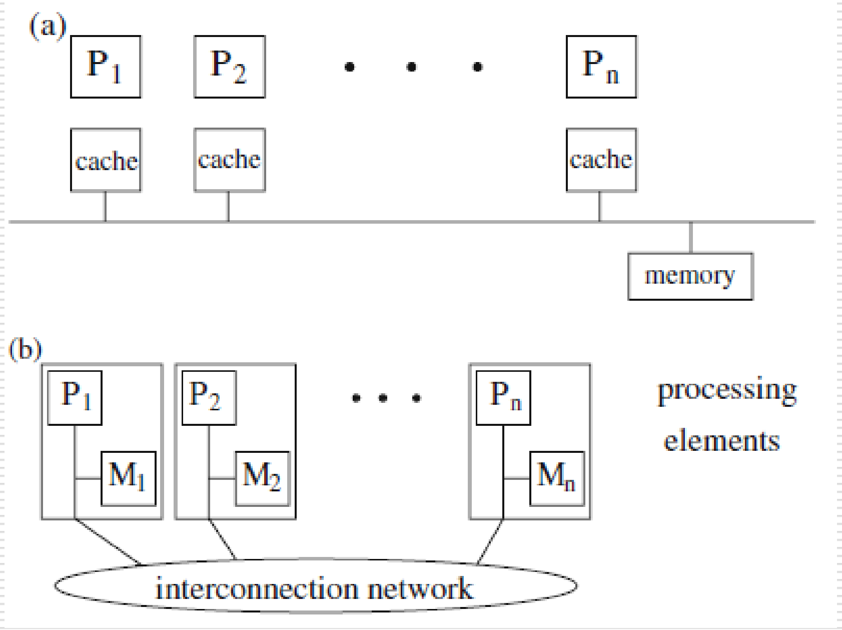
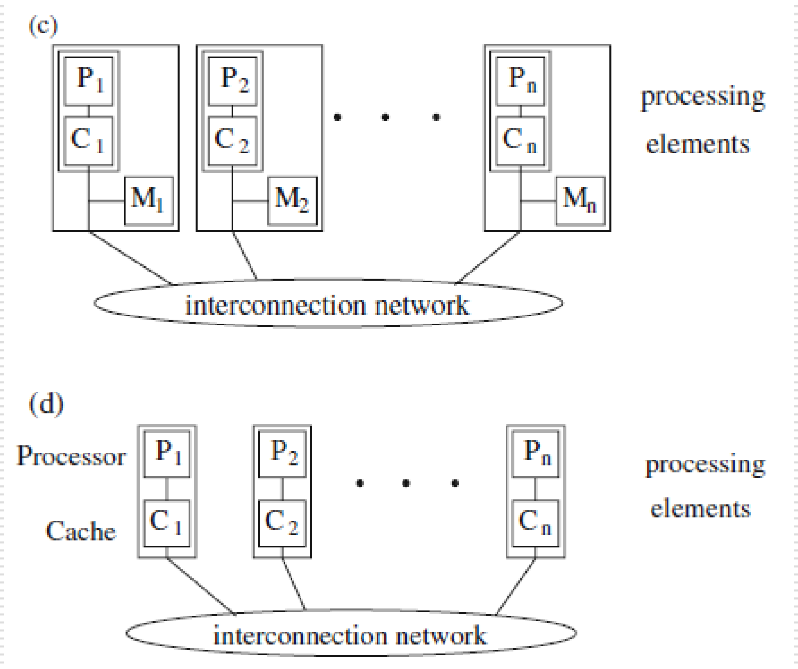


Illustration of the architecture of computers with shared memory:

Illustration of the architecture of computers with shared memory: (**a**) SMP – symmetric multiprocessors, (**b**) NUMA – non-uniform memory access, (**c**) CC-NUMA – cache-coherent NUMA, and (**d**) COMA – cache-only memory access





Thread-Level Parallelism

What is Thread w.r.t Computers ?

Thread: A Process (or Program) with own instructions and data

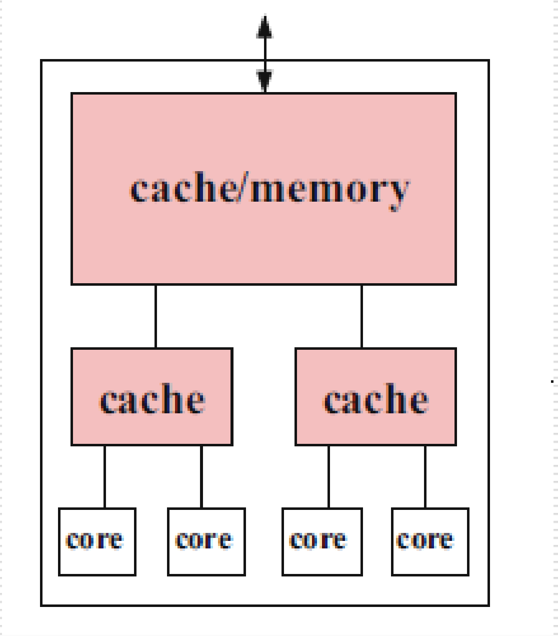
- Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute

Thread-Level Parallelism

Multi-Core Processors: Placement of multiple independent execution cores with all execution resources onto a single processor chip.

Design choices for multicore chips

Hierarchical design

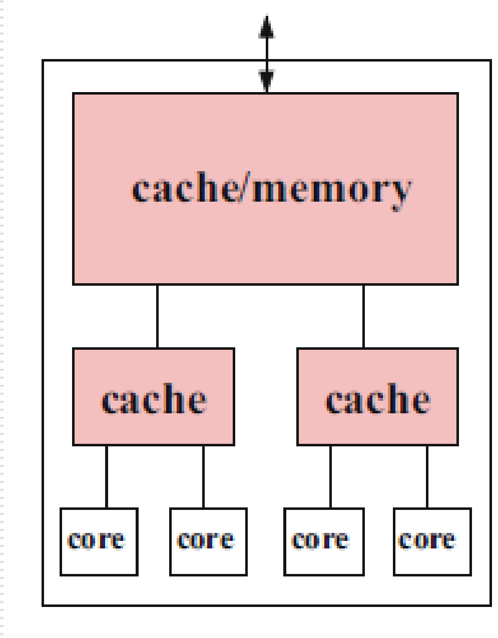


Thread-Level Parallelism

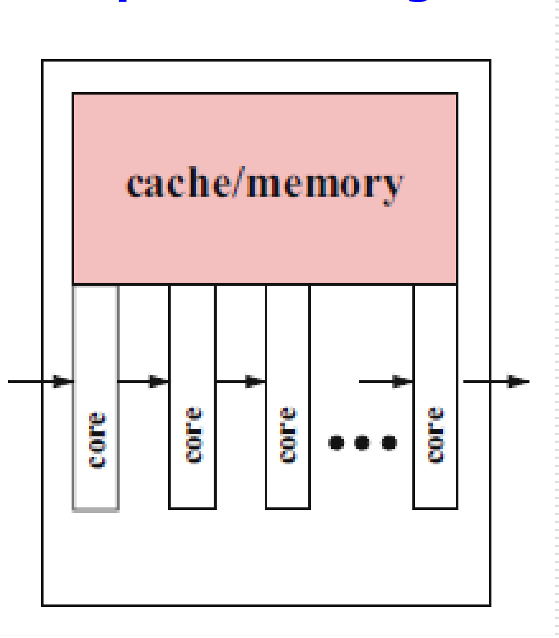
Multi-Core Processors: Placement of multiple independent execution cores with all execution resources onto a single processor chip.

Design choices for multicore chips

Hierarchical design



Pipelined design

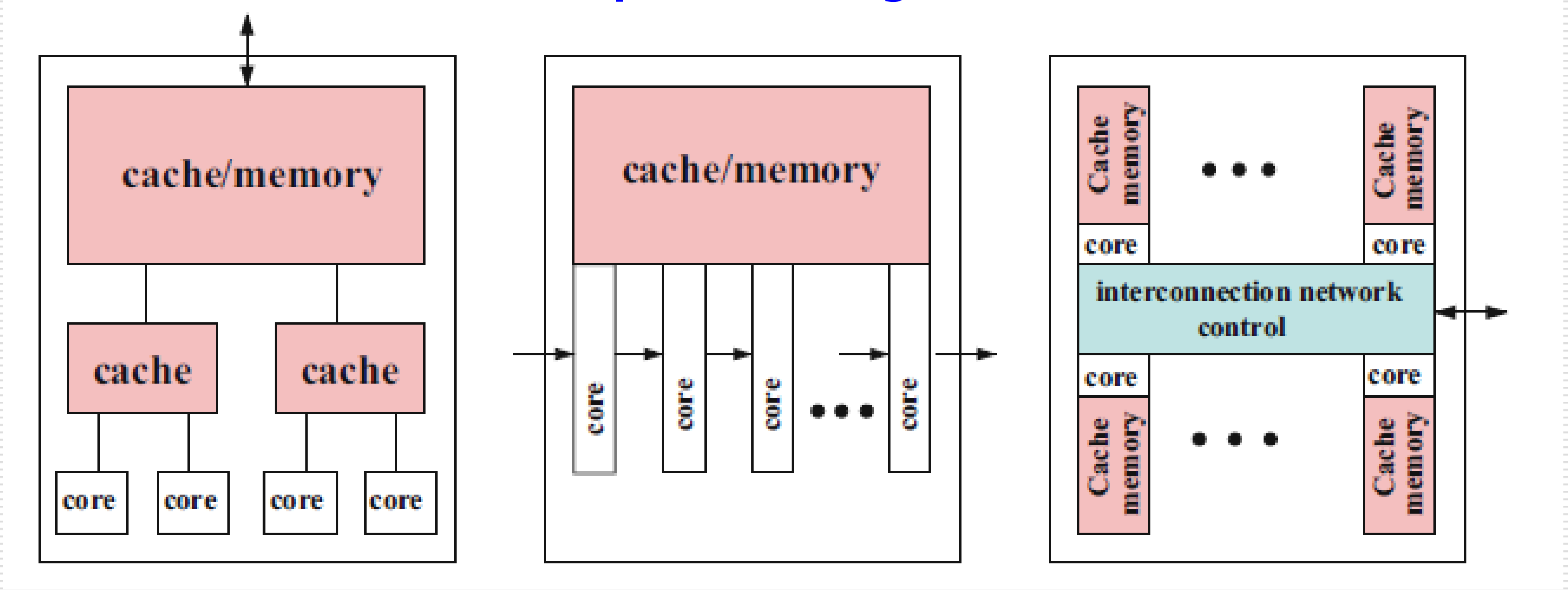


Thread-Level Parallelism

Multi-Core Processors: Placement of multiple independent execution cores with all execution resources onto a single processor chip.

Design choices for multicore chips

Hierarchical design **Pipelined design** **Network-based design**



Thanks for Listening