

Module 5

Parallel Processing

Jestin James M
Assistant Professor, Dept of Computer Science
Little Flower College, Guruvayoor

Topics to Cover

- Parallel Processing
- Taxonomy- SISD, MISD, SIMD, MIMD structures
- CISC Vs RISC
- Symmetric Multiprocessors
- Cache coherence
- MESI protocol
- Clusters
- Non Uniform Memory Access
- Pipelining
- Instruction Pipelining.
- Hazards,
- Reservation Tables
- Collision,
- Latency,
- Dynamic pipeline,
- Vector processing
- Vector processors



Parallel Processing

- parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- Increasing the computational speed of a computer system
- while an instruction is being executed in the ALU, the next instruction can be read from memory.



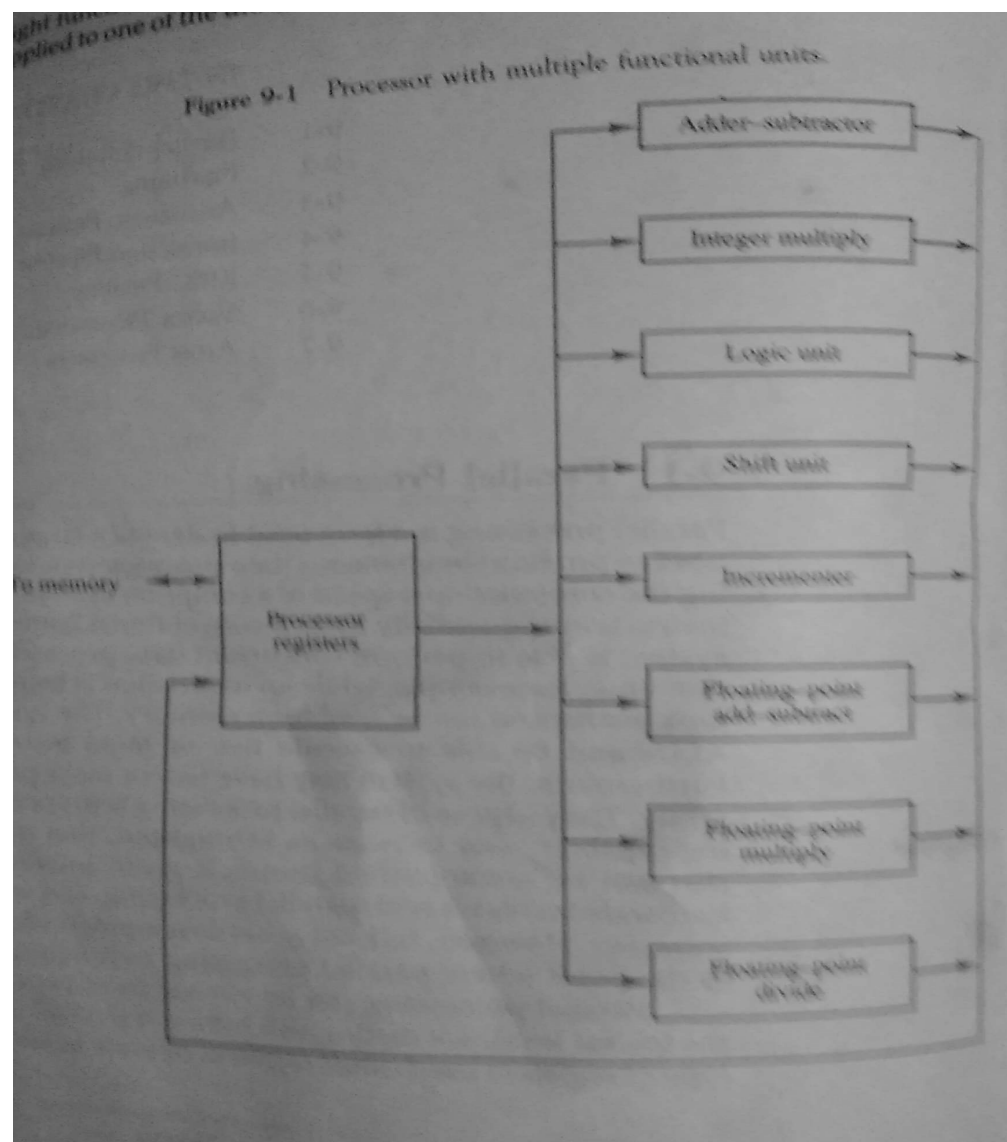
Parallel Processing

- The purpose of parallel processing is to speed up the computer processing capability and increase its throughput
- The amount of hardware increases with parallel processing. and with it, the cost of the system increase
- Parallel processing can be viewed from various levels of complexity



Parallel Processing

- we distinguish between parallel and serial operations by the type of registers used.
- shows one possible way of separating the execution unit into eight functional units operating in parallel





PARALLELISM

- Instruction level parallelism
- Processor level- parallelism

Instruction -Level Parallelism

- A problem is broken into a discrete series of instructions.
- The concept of pipelining is used so that you can overlap the execution of instructions which are independent of one another.
- Overlap among instructions is called Instruction-Level Parallelism(ILP)
- Instruction is evaluated in parallel
- Vector and array architecture are based on instruction –level parallelism



Two types Instruction level

- Pipelining:
- Superscalar



Pipelining

- Execution of an instruction is divided into multiple stages.
- In one single cycle multiple instructions are being executed at different stages.
- Overall throughput is increased
- but individual instruction takes longer time as compared to non-pipeline system

Superscalar

- Deep pipelining allows short cycle(fast clock rate)causing memory bottle neck.
- Hardware can issue multiple instructions simultaneously.



Processor-level Parallelism

- Multiple processor that share the workload.
- It is achieved by breaking up a problem into series of instructions and distributing the data among various functional unit.
- A single computer with multiple processors
- Or an more than one computer connected by network cluster.



Multi processors

- Broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any time
- Solved in less time with multiple computer resources than with a single computer resource



To provide proper functioning

- 1.multiprocessor
- Inter process communication
- Synchronization

Parallel processors Categorized as

- ❑ Pipeline computers: A pipeline computer performs overlapped computation by exploiting the parallelism
- ❑ Array processors: Deals with repetitive operations. These are SIMD architecture
- ❑ Distributed Architecture : autonomous sub systems which are capable of handling complete systems
- ❑ Multiprocessors : The communication between the processors takes place either through shared memory area or through inter process messages



Parallel processors Categorized as

- Dataflow Architectures: Functionally distributed architectures in which the operations are triggered with the arrival of data at these processors. MIMD
- VLSI computing Structure:



Uniprocessor architecture

- Single CPU
- Serial communication
- Fetch instruction
- Decode
- Execute

Uniprocessor

- Various ways to implement parallel processing in uniprocessor Architecture
 1. Multiplicity of functional unit: Multiple ALUs in one CPU

All independent with each other CDC6600

IBM 360/91 :Two parallel execution unit one for fixed point arithmetic and other for floating point arithmetic.



Uniprocessor

2. Parallelism and pipelining within the CPU:
Parallel adders
3. Overlapped CPU & I/o Operations: DMA, IOP
4. Use of Hierarchical Memory system
5. Balancing of subsystems Bandwidth: Number operations performed at unit time
6. Multiprogramming
7. Time Sharing

Architectural Classification Schemes

- Flynn's Classification(1966): it is based on multiplicity of instruction streams and data streams in a computer
- Feng's Classification(1972): Based on serial Vs Parallel processing
- Shores Classification(1973): Based on organization of constituent elements in the computer
- Handlers Classification(1977): it is determined by degree of parallelism and pipelining in various subsystems

Flynn's Classical Taxonomy

- Two independent dimensions
- “Instruction stream” and “Data stream”
- An instruction stream is a sequence of instructions read from memory
- Data stream: The operations performed on the data in the processor constitutes a data stream

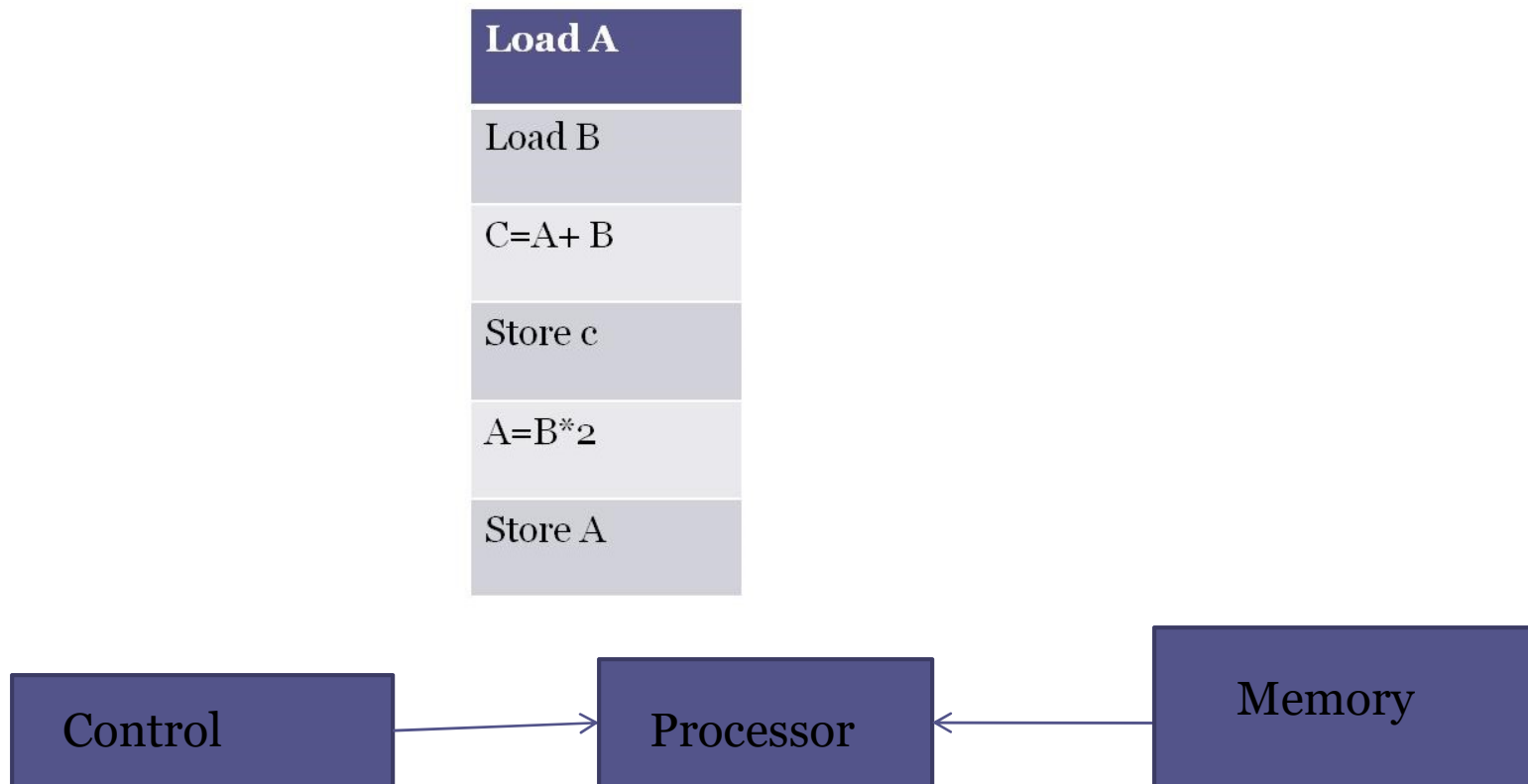
4 Flynn's Categories

1. SISD- Single Instruction stream, Single data Stream
2. SIMD-Single instruction stream multiple data stream
3. MISD-Multiple Instruction stream Single Data Stream
4. MIMD-Multiple instruction stream ,multiple data stream

SISD

- Single computer containing a control unit, a processor unit, and a memory unit
- System may or may not have internal parallel processing.
- Parallel processing may be achieved by multiple functional unit or by pipeline processing

SISD





SSID

- These are also called scalar processors
- One instruction stream on one clock cycle
- Deterministic execution
- Instruction are executed sequentially

SIMD

- Many processing units under a common control unit
- All processors receive the same instruction from control unit but operate on different items of data

Prev instr
Load A(1)
Load B(1)
$C(1)=A(1) * B(1)$
Store c(1)
Next instr

Prev instr
Load A(2)
Load B(2)
$C(2)=A(2) * B(2)$
Store c(2)
Next instr

Prev instr
Load A(n)
Load B(n)
$C(n)=A(n) * B(n)$
Store c(n)
Next instr

MISD

- Single data stream is fed into multiple processing units
- Each processing unit operates on the data independently via independent instruction stream
- Ex: Multiple cryptography algorithms attempting to crack a single coded message

MIMD

- A computer system capable of processing several programs at the same time
- It can be categorized as loosely coupled or tightly coupled depending on sharing of data and control
- Execution of instruction scheme can be synchronous and asynchronous

CISC(Complex Instruction set Computer)

- A computer with a large number of instructions is classified as a complex instruction set Computer, abbreviated CISC.

It is designed to develop high level language Compiler

It is easy to program

Makes efficient use of memory

CISC characteristics

1. A large number of instructions-typically from 100 to 250 instructions
2. Some instructions that perform specialized tasks and are used infrequently
3. A large variety of addressing modes-typically from 5 to 20 different modes
4. Variable-length instruction formats
5. Instructions that manipulate operands in memory

RISC(Reduced Instruction Set Computer)

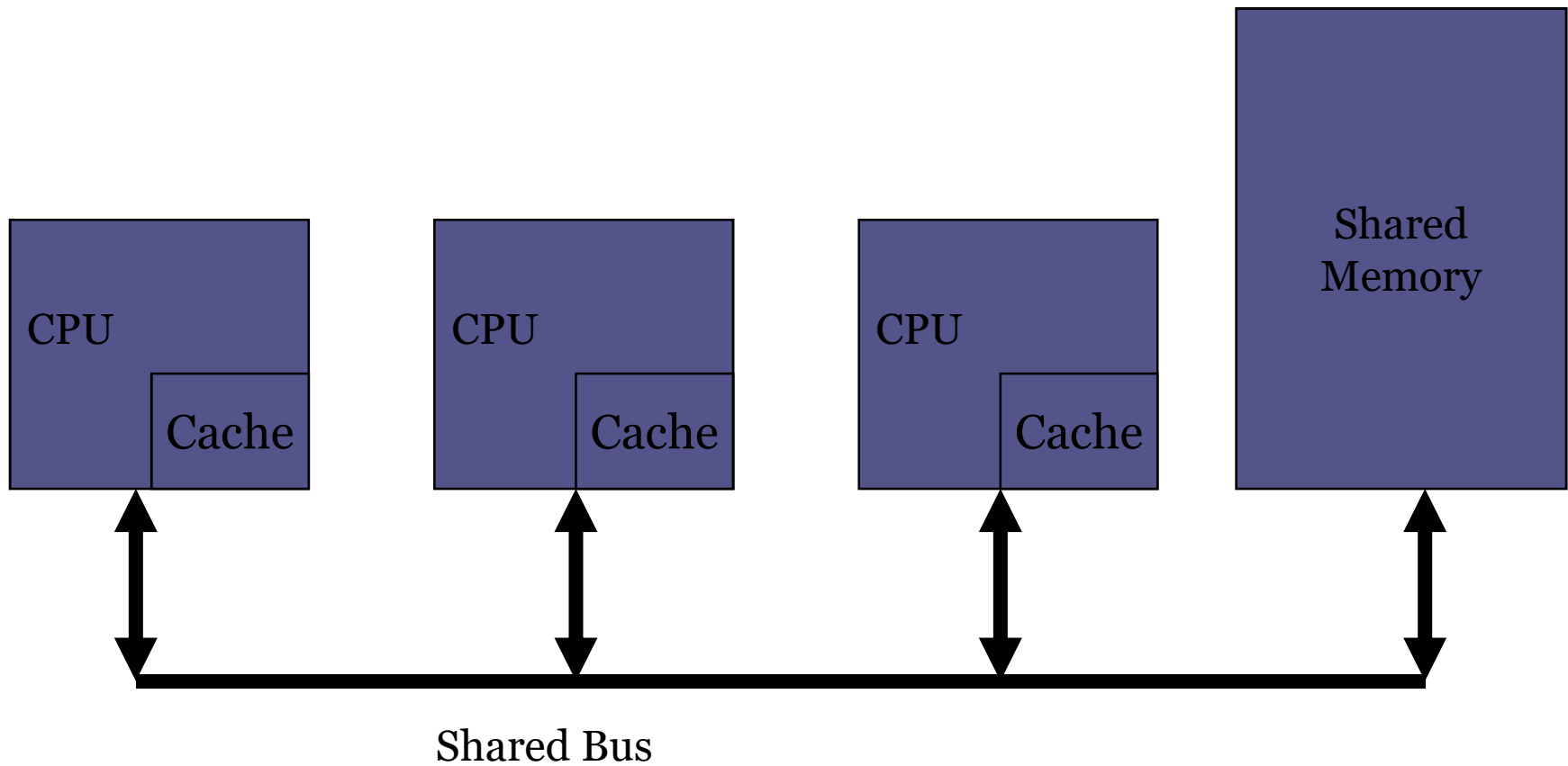
- 1. Relatively few instructions
- 2. Relatively few addressing modes
- 3. Memory access limited to load and store instructions
- 4. All operations done within the registers of the CPU
- 5. Fixed-length, easily decoded instruction format
- 6. Single-cycle instruction execution
- 7. Hardwired rather than micro programmed control



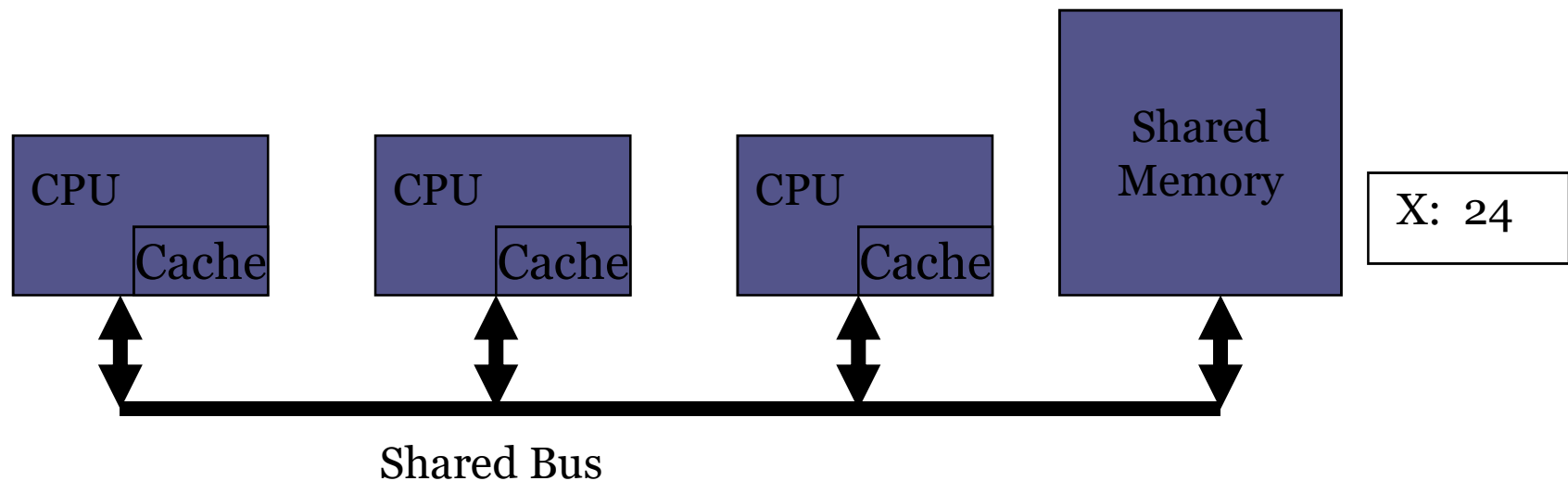
Cache Coherence

- Any system that allows multiple processors to access multiple copies of data
- If there exist two copies of data

Example



Example Continue



- Processor 1 reads X: obtains 24 from memory and caches it
- Processor 2 reads X: obtains 24 from memory and caches it

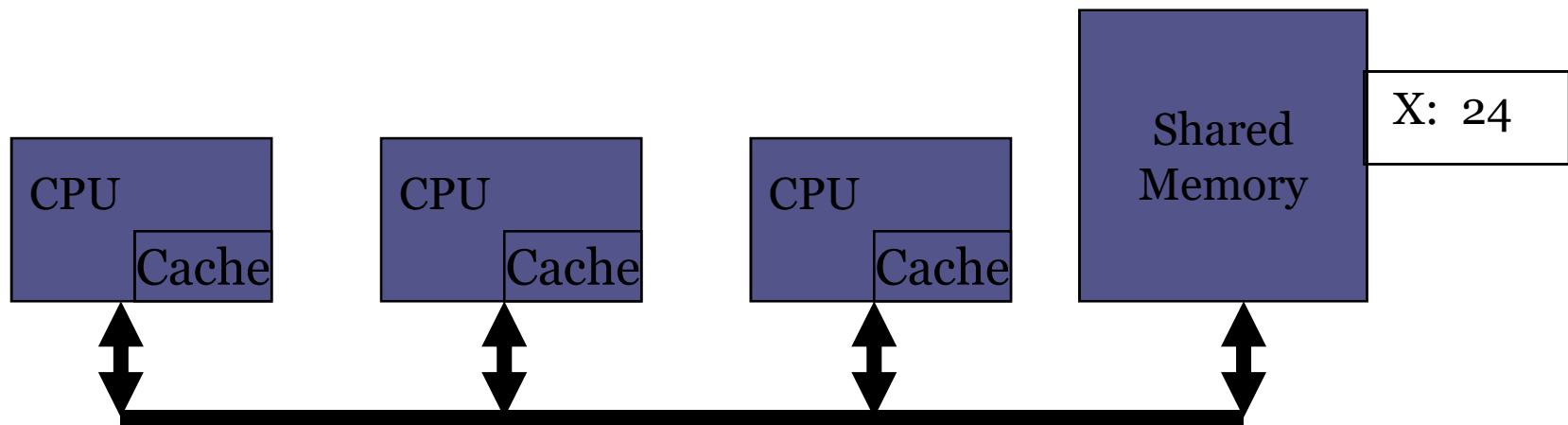


Example continue

Write through

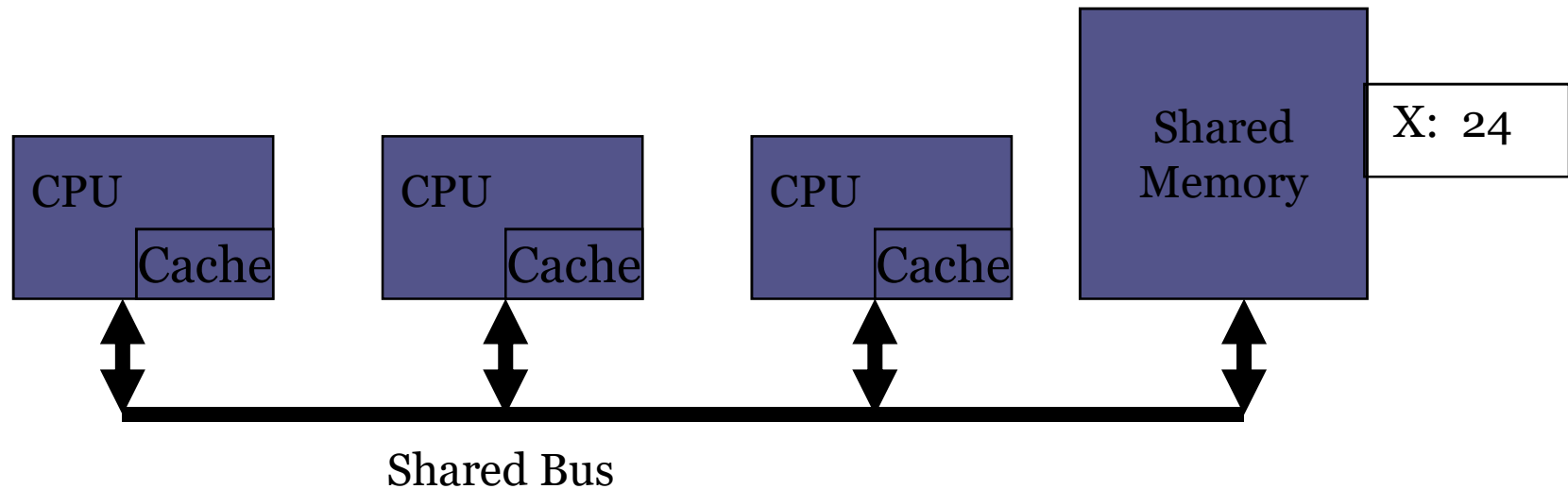
Write back policy

Write through both cache and memory is written
in parallel



- Processor 1 writes 32 to X: its locally cached copy is updated
- Processor 3 reads X: what value should it get?
- processor 2 think it is 24
- Memory and Processor 1 thinks it is 32

Write back policy problem



- Processor 1 reads X: obtains 24 from memory and caches it
- Processor 2 reads X: obtains 24 from memory and caches it
- Processor 1 writes 32 to X: its locally cached copy is updated
- Processor 3 reads X: what value should it get?
 - Memory and processor 2 think it is 24
 - Processor 1 thinks it is 32

Cache coherence

- If one processor modifies a shared cached value then the other processor must get the latest value
 1. disallow private cacheshared cache means not close to CPU. Increases memory access time



Cache coherence software solutions

2. Allow only non-shared and read only data to be stored in the cache(Cacheable)

Compiler will tag data as cacheable and non-cacheable

But it degrades due to software overhead and also data to reside in the cache



Cache coherence software solutions

3. Use the centralised global table such that writeable data exists in at least one cache

This table is maintained by compiler

Each block in the table is identified RO(Read Only) or RW(Read and write)

All cache will have RO

Only common cache will have RW block

Hardware approaches

1. Snooping Protocol : Cache controller listens passively for bus activity .
 - This is non scalable and is not appropriate for machine with tens of processors or more
2. Directory based: point-to-point message to handle coherence.

A memory structure called a directory and it maintains information about data sharing

MESI

Any cache line can be in one of 4 states (2 bits)

- **Modified** - cache line has been modified, is different from main memory - is the only cached copy. (multiprocessor 'dirty')
- **Exclusive** - cache line is the same as main memory and is the only cached copy
- **Shared** - Same as main memory but copies may exist in other caches.
- **Invalid** - Line data is not valid (as in simple cache)



Pipelining

- Pipeline processing is an implementation technique where arithmetic sub operations
- or the phases of a computer instruction cycle overlap in execution



Pipelining

- Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments

Figure 9-2
Example of Pipelining.

- $R1 \leftarrow A_i, R2 \leftarrow B_i$
Input A_i and B_i
- $R3 \leftarrow R1 * R2, R4 \leftarrow C_i$
Multiply and input C_i
- $R5 \leftarrow R3 + R4$
Add C_i to product

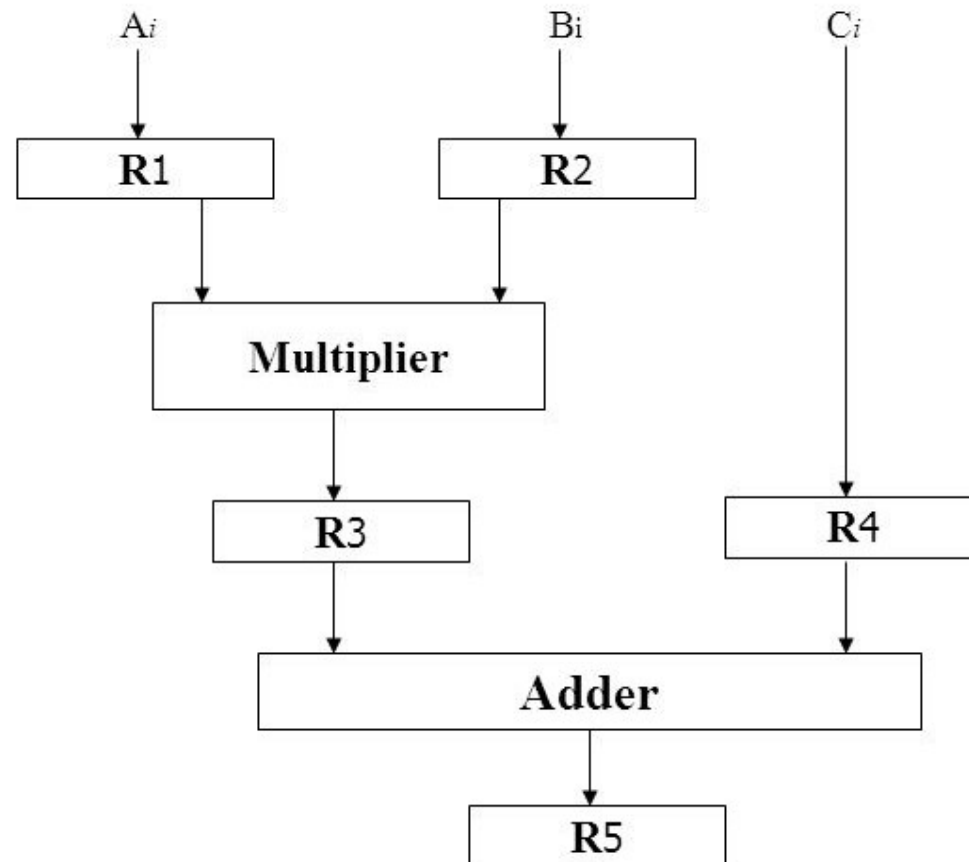


TABLE 9-1 Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$



Different Pipelining

1. Arithmetic pipelining
2. Instruction pipelining

Instruction Pipeline

- Instruction cycle
 1. Fetch the instruction from memory.
 2. Decode the instruction.
 3. Calculate the effective address.
 4. Fetch the operands from memory.
 - 5 . Execute the instruction.
 6. Store the result in the proper place.

Instruction Pipeline

- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segment
- There are certain difficulties
 1. Different segments may take different times to operate on the incoming information
 2. Two or more segments may require memory access at the same time

Instruction Pipeline

- 3. Some segments are skipped for certain operations. Ex: register mode instruction does not need an effective address calculation.
- Memory access conflict resolve -two memory buses for accessing instructions and data in separate modules
- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration

Pipeline Hazards

- Where one instruction cannot immediately follow another
- Types of hazards
 - Structural hazards - attempt to use the same resource by two or more instructions
 - Control hazards - attempt to make branching decisions before branch condition is evaluated
 - Data hazards - attempt to use data before it is ready
- Can always resolve hazards by waiting

Structural Hazards

- Attempt to use the same resource by two or more instructions at the same time
- Example: Single Memory for instructions and data
- Solutions
 - Delay the second access by one clock cycle, OR(Stall)
 - Provide separate memories for instructions & data

Structural Hazards

- Having more than one resources
- Separate data & instruction memory if there is some conflict in memory access.
- Example 5. 7: Refer text

Stall

- low cost, simple
- Increases CPI (cycles per Instruction)
- use for rare case since stalling has performance effect



Structural Hazards

Replicate resource

- good performance
- increases cost (+ maybe interconnect delay)
- useful for cheap or divisible resources
- Consistency problem . Different registers will have different value



Data Hazards

- Data hazards occur when data is used before it is ready


Data Hazards

Execution Order is:

Instr_I
Instr_J

Read After Write (RAW)

Instr_J tries to read operand before Instr_I writes it

 I: add r1, r2, r3
J: sub r4, r1, r3

- Caused by a “**Dependence**” (in compiler nomenclature). This hazard results from an actual need for communication.

Data Hazards

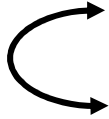
Execution Order is:

Instr_I
Instr_J

Write After Write (WAW)

Instr_J tries to write operand *before* Instr_I writes it

- Leaves wrong result (Instr_I not Instr_J)

 I: sub r1, r4, r3
J: add r1, r2, r3
K: mul r6, r1, r7

- Called an “**output dependence**” by compiler writers
This also results from the reuse of name “r1”.

Data Hazards

Execution Order is:

Instr_I

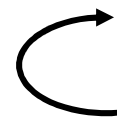
Instr_J

Write After Read (WAR)

Instr_J tries to write operand *before* Instr_I reads it

- Gets wrong operand

```
    I: sub  r4, r1, r3
    J: add  r1, r2, r3
    K: mul  r6, r1, r7
```



- Called an “anti-dependence” by compiler writers. This results from reuse of the name “r1”.



Data Hazards

- Solutions for Data Hazards
 - **Stalling**
 - **Forwarding:**
 - connect new value directly to next stage
 - **Reordering**

Control Hazards

A *control hazard* is when we need to find the destination of a branch, and can't fetch any new instructions until we know that destination.

A branch is either

- **Taken:** $PC \leq PC + 4 + \text{Immediate}$
- **Not Taken:** $PC \leq PC + 4$



Control Hazards - Solutions

- Delayed branches – code rearranged by compiler to place independent instruction after every branch (in delay slot).



Vector Processing

- In many science and engineering applications, the problems can be formulated in terms of vectors and matrices that lend themselves to vector processing



Applications

- Long-range weather forecasting
- Petroleum explorations
- Seismic data analysis
- Medical diagnosis
- Aerodynamics and space flight simulations
- Artificial intelligence and expert systems
- Mapping the human genome
- Image processing



Vector Operations