# What is *Computer Architecture*

Computer Architecture =

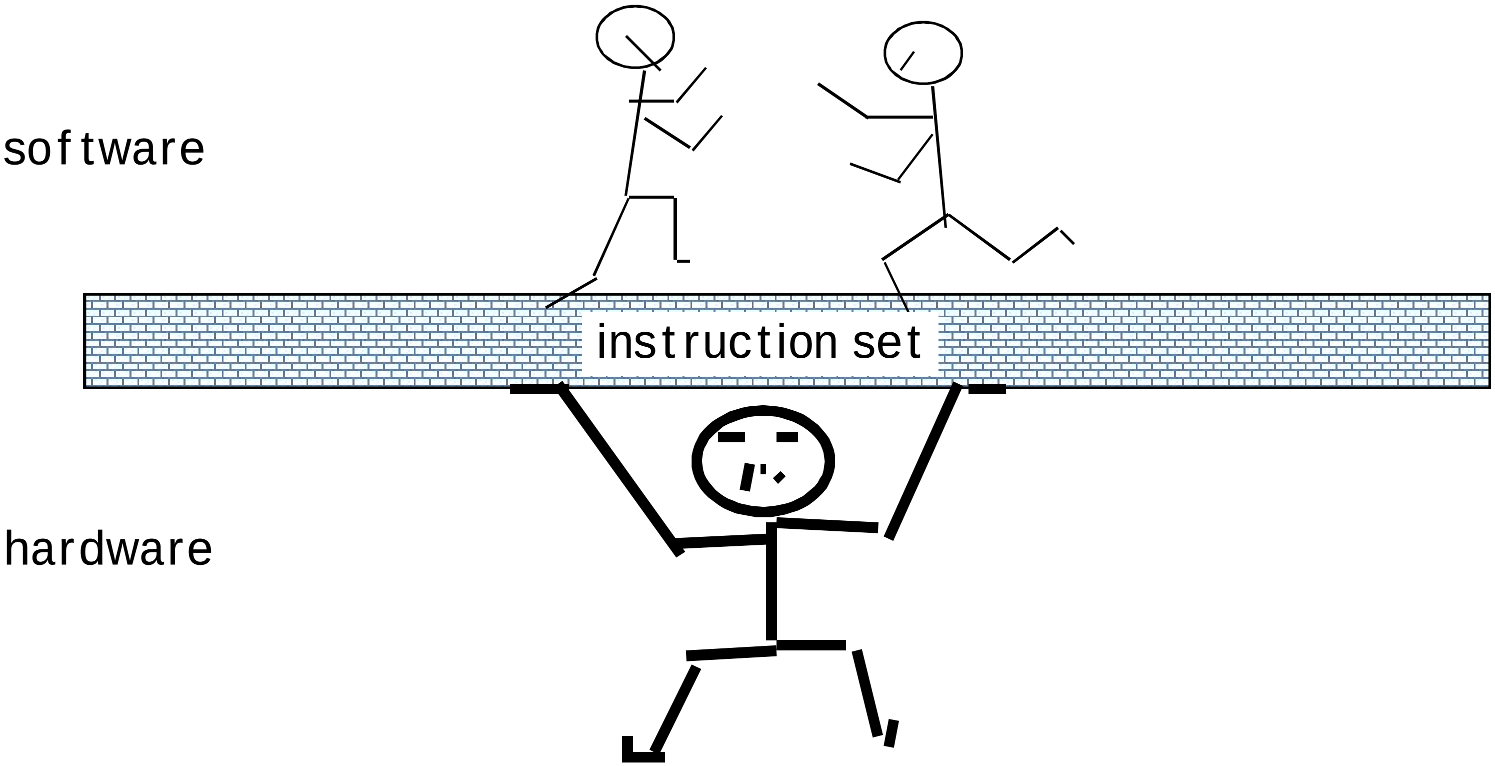
Instruction Set Architecture +

Organization +

Hardware + ...

# The actual programmer visible instruction set

- INSTRUCTION SET



# Instruction-Set Processor Design

- Architecture (ISA) programmer/compiler view
  - “functional appearance to its immediate user/system programmer”
  - **Opcodes, addressing modes, architected registers, IEEE floating point**
- Implementation (µarchitecture) processor designer/view
  - “logical structure or organization that performs the architecture”
  - **Pipelining, functional units, caches, physical registers**
- Realization (chip) chip/system designer view
  - “physical structure that embodies the implementation”
  - **Gates, cells, transistors, wires**

# Hardware

- Machine specifics:
  - Feature size (10 microns in 1971 to 0.18 microns in 2001)
    - Minimum size of a transistor or a wire in either the x or y dimension
  - Logic designs
  - Packaging technology
  - Clock rate
  - Supply voltage

  …

# Applications and Requirements

- Scientific/numerical: weather prediction, molecular modeling
  - Need: large memory, floating-point arithmetic
- Commercial: inventory, payroll, web serving, e-commerce
  - Need: integer arithmetic, high I/O
- Embedded: automobile engines, microwave, PDAs
  - Need: low power, low cost, interrupt driven
- Home computing: multimedia, games, entertainment
  - Need: high data bandwidth, graphics

# UNIT-1

# PARALLEL COMPUTER MODELS

# History of Computers

- **First Generation: Vacuum Tubes**
- ENIAC
  - Electronic Numerical Integrator And Computer

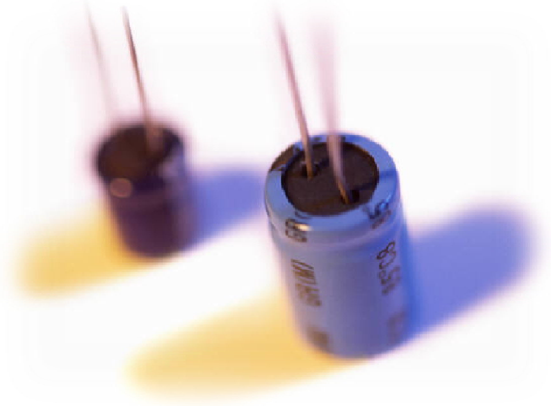
- Designed and constructed at the University of Pennsylvania
  - Started in 1943 – completed in 1946
  - By John Mauchly and John Eckert
- World's first general purpose electronic digital computer
  - Army's Ballistics Research Laboratory (BRL) needed a way to supply trajectory tables for new weapons accurately and within a reasonable time frame
  - Was not finished in time to be used in the war effort

– Its first task was to perform a series of calculations that were used to help determine the feasibility of the hydrogen bomb

– Continued to operate under BRL management until 1955 when it was disassembled

# History of Computers

- Second Generation:  Transistors



  - Smaller
  - Cheaper
  - Dissipates less heat than a vacuum tube
  - Is a *solid state device* made from silicon
  - Was invented at Bell Labs in 1947
  - It was not until the late 1950’s that fully transistorized computers were commercially available

# History of Computers

- Third Generation: Integrated Circuits
- 1958 – the invention of the integrated circuit
- *Discrete component*
  - Single, self-contained transistor
  - Manufactured separately, packaged in their own containers, and soldered or wired together onto masonite-like circuit boards
  - Manufacturing process was expensive and cumbersome

– The two most important members of the third generation were the IBM System/360 and the DEC PDP-8

# Later Generations

- Semiconductor Memory Microprocessors
- LSI Large Scale Integration
- VLSI Very Large Scale Integration
- ULSI Ultra Large Scale Integration

# Multiprocessor

- A hierarchical bus system consists of a hierarchy of buses connecting various systems and sub-systems/components in a computer. Each bus is made up of a number of signal, control, and power lines. Different buses like local buses, backplane buses and I/O buses are used to perform different interconnection functions.
- Switched networks give dynamic interconnections among the inputs and outputs. Small or medium size systems mostly use crossbar networks. Multistage networks can be expanded to the larger systems, if the increased latency problem can be solved.
- Multistage networks or multistage interconnection networks are a class of high-speed computer networks which is mainly composed of processing elements on one end of the network and memory elements on the other end, connected by switching elements.

# Multi computers

- Multi computers are message-passing machines which apply packet switching method to exchange data. Here, each processor has a private memory, but no global address space as a processor can access only its own local memory. So, communication is not transparent.

  - Virtual Shared Memory (VSM)
  - Shared Virtual Memory (SVM)

- Message-Routing Schemes

  In multicomputer with store and forward routing scheme, packets are the smallest unit of information transmission. In wormhole–routed networks, packets are further divided into flits. Packet length is determined by the routing scheme and network implementation, whereas the flit length is affected by the network size.

# Vector processor and SIMD

- A vector processor or array processor is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one dimensional arrays of data called vectors, compared to scalar processors, whose instructions operate on single data items. Vector processors can greatly improve performance on certain workloads, notably numerical simulation and similar tasks.
- Vector processing techniques have since been added to almost all modern CPU designs, although they are typically referred to as SIMD(differing in that a single instruction always drives a single operation across a vector register, as opposed to the more flexible latency hiding approach in true vector processors). In these implementations, the vector unit runs beside the main scalar CPU, providing a separate set of vector registers, and is fed data from vector instruction aware programs.

# Parallelism

- A *parallel computer* is a set of processors that are able to work cooperatively to solve a computational problem. This definition is broad enough to include parallel supercomputers that have hundreds or thousands of processors, networks of workstations, multiple-processor workstations, and embedded systems. Parallel computers are interesting because they offer the potential to concentrate computational resources.
- Types
  - Instruction level Parallelism

    Pipelining

    superscalar
  - Processor level Parallelism

    Array computer

    Multiprocessor

# Partitioning and scheduling

- Partitioning and scheduling are multiprocessor-dependent issues. Partitioning is necessary to ensure that the granularity of the parallel program is coarse enough for the target multiprocessor, without losing too much parallelism. Scheduling is necessary to achieve a good processor utilisation and to optimise inter-processor communication in the target multiprocessor.
- The task's sequential execution time (also called the task's size).
- The task's total overhead, which includes scheduling overhead and communication overhead for the task's inputs and outputs.
- The task's precedence constraints, which specify the parallelism in the partitioned program.

# Program Flow Mechanisms

- Conventional machines used control flow mechanism in which order of program execution explicitly stated in user programs.
- Dataflow machines which instructions can be executed by determining operand availability.
- Reduction machines trigger an instruction’s execution based on the demand for its results.
- Control flow machines used shared memory for instructions and data. Since variables are updated by many instructions, there may be side effects on other instructions. These side effects frequently prevent parallel processing. Single processor systems are inherently sequential.

# Amdahl’s Law

- Deals with the potential speedup of a program using multiple processors compared to a single processor
- Illustrates the problems facing industry in the development of multi-core machines
  - Software must be adapted to a highly parallel execution environment to exploit the power of parallel processing
- Can be generalized to evaluate and design technical improvement in a computer system

# Gustafson's law

- Gustafson's law addresses the shortcomings of Amdahl's law, which is based on the assumption of a fixed Problem size, that is of an execution workload that does not change with respect to the improvement of the resources. Gustafson's law instead proposes that programmers tend to set the size of problems to fully exploit the computing power that becomes available as the resources improve.

- A task executed by a system whose resources are improved compared to an initial similar system can be split into two parts:

  -a part that does not benefit from the improvement of the resources of the system;

  -a part that benefits from the improvement of the resources of the system.

# UNIT-2

## MEMORY SYSTEMS AND BUSES

# Memory hierarchy

- In computer architecture the memory hierarchy is a concept used to discuss performance issues in computer architectural design, algorithm predictions, and lower level programming constructs involving locality of reference. The memory hierarchy in computer storage separates each of its levels based on response time.
- The memory hierarchy in most computers is as follows:

  * Processor registers – fastest possible access (usually 1 CPU cycle), only hundreds of bytes in size

  * Level 1 (L1) cache – often accessed in just a few cycles, usually tens of kilobytes

  * Level 2 (L2) cache – higher latency than L1 by 2× to 10×, often 512KB or more

  * Level 3 (L3) cache – (optional) higher latency than L2, often multiple MB's

  * Main memory (DRAM) – may take hundreds of cycles, but can be multiple gigabytes

  * Disk storage – hundreds of thousands of cycles latency, but very large

# Cache memory

- The central processing unit (CPU) is the brain of the computer. All of the instructions have to run through the CPU for the various parts of a computer to work together. CPU chips have been getting smaller and faster as chip technology has advanced. One of the slower aspects of computer processing is the interaction between the CPU chip and the main memory in the form of random-access memory (RAM). Installing more memory is not always a solution - the bottleneck is often the time it takes to access the memory.
- So, what have chip designers come up with? A small form of memory located directly on the chip itself. This is the **CPU cache**. It is much smaller, but can be accessed much faster than the main memory. The CPU cache stores the most frequently used pieces of information so they can be retrieved more quickly. This information is a duplicate of information stored elsewhere, but it is more readily available.
- Cache memory, also called CPU memory, is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

# How the CPU Cache Works

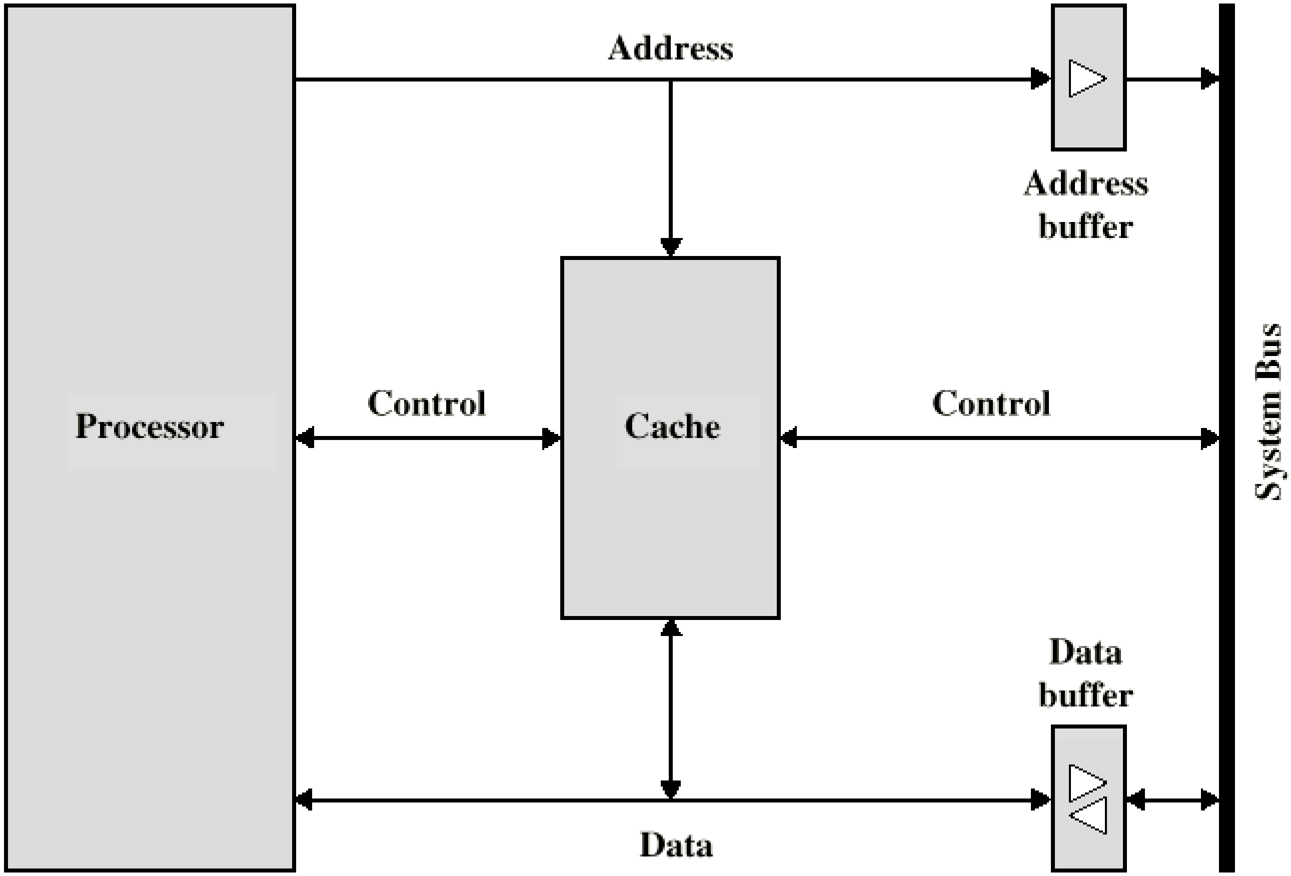
- To carry out a particular instruction, the CPU needs a specific piece of information. The CPU will first check to see if this information is available in the CPU cache. If the information is found, this is called a **cache hit**. If the information is not found, this is called a **cache miss**, and the CPU goes on looking for the information elsewhere. In the case of a cache miss, the piece of information will be found in the main memory, but it will simply take longer.

- A lot of research has gone into how to optimize the design of cache memory. The result has been somewhat counter-intuitive: smaller is faster. What this means is that a relatively small CPU cache improves speed, but as the cache gets really large, it no longer helps as much, and the CPU might as well look for the information in the main memory.

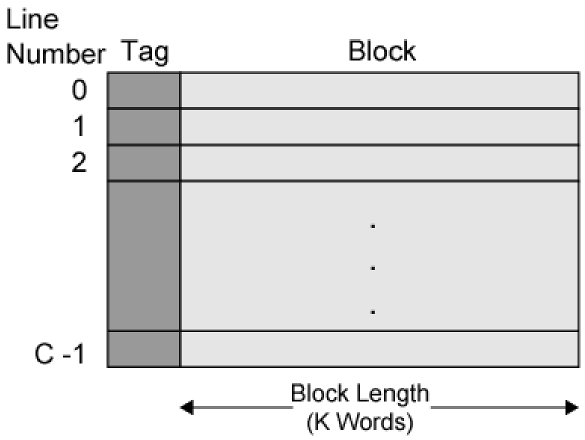
# shared-memory

- In computer programming, shared memory is a method by which program processes can exchange data more quickly than by reading and writing using the regular operating system services. For example, a client process may have data to pass to a server_process that the server process is to modify and return to the client. Ordinarily, this would require the client writing to an output file (using the buffers of the operating system) and the server then reading that file as input from the buffers to its own work space.
- Using a designated area of shared memory, the data can be made directly accessible to both processes without having to use the system services. To put the data in shared memory, the client gets access to shared memory after checking a semaphorevalue, writes the data, and then resets the semaphore to signal to the server (which periodically checks shared memory for possible input) that data is waiting. In turn, the server process writes data back to the shared memory area, using the semaphore to indicate that data is ready to be read.

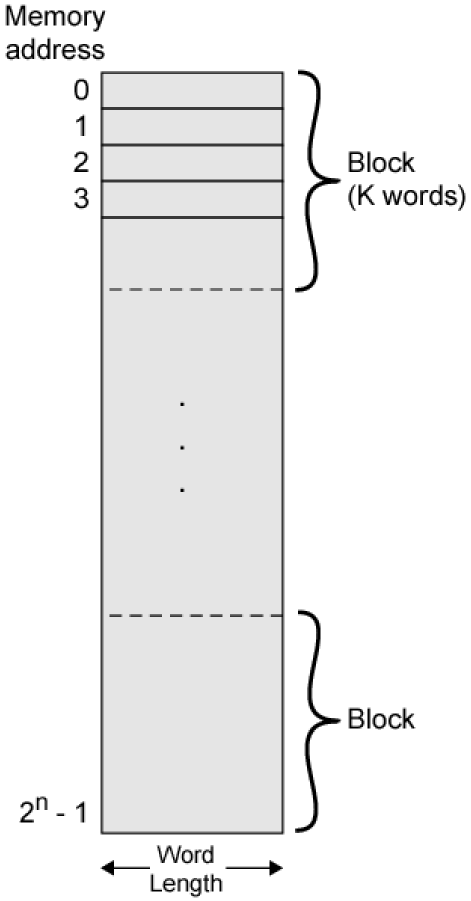
# cache memory organization



# Cache/Main Memory Structure



(a) Cache

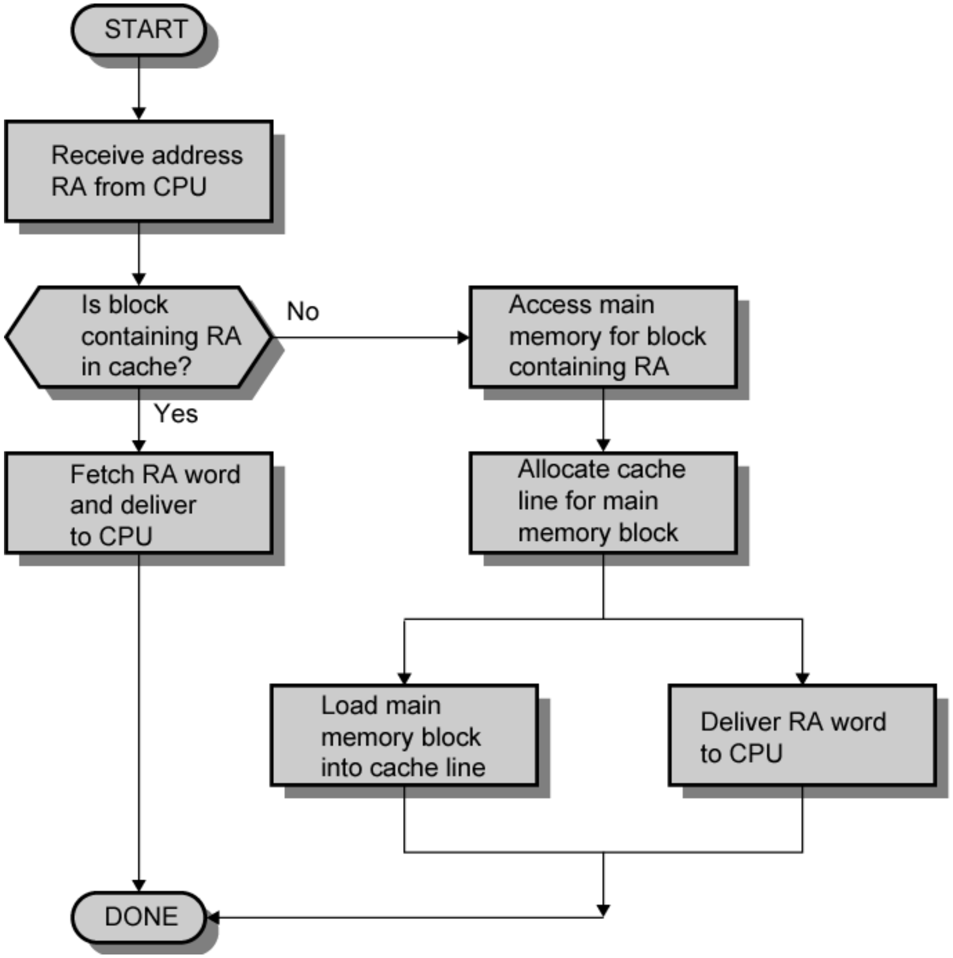


(b) Main memory

# Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

# Cache Read Operation - Flowchart


START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

# Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

# CACHE MEMORY

Principle of locality helped to speed up main memory access by introducing small fast memories known as CACHE MEMORIES that hold blocks of the most recently referenced instructions and data items.

Cache is a small fast storage device that holds the  operands and instructions most likely to be used by the CPU.

# Mapping techniques

There are three commonly used methods to translate main memory addresses to cache memory addresses.

- Associative Mapped Cache
- Direct-Mapped Cache
- Set-Associative Mapped Cache

# Set-Associative Mapping

- This is a trade-off between associative and direct mappings where each address is mapped to a certain set of cache locations.

- The cache is broken into sets where each set contains "N" cache lines, let's say 4. Then, each memory address is assigned a set, and can be cached in any one of those 4 locations within the set that it is assigned to. In other words, within each set the cache is associative, and thus the name.

# ASSOCIATIVITY

- Associativity : N-way set associative cache memory means that information stored at some address in operating memory could be placed (cached) in N locations (lines) of this cache memory.

- The basic principle of logical segmentation says that there is only one line within any particular segment to be capable of caching information located at some memory address.

# Direct Mapping

- To avoid the search through all CM blocks needed by associative mapping, this method only allows

  # blocks in cache memory

  # blocks in main memory

blocks to be mapped to each Cache Memory block.

# Characteristics of shared memory systems

- Any processor can *directly* reference any memory location.
- Communication occurs implicitly as result of loads and stores.
- Location of data in memory is transparent to the programmer.
- Inherently provided on wide range of platforms (standard processors today have specific extra hardware for share memory systems)
- Memory may be physically distributed among processors.

# Memory Interleaving

- Memory interleaving is the technique used to increase the throughput. The core idea is to split the memory system into independent banks, which can answer read or write requests independents in parallel.

# Interleaved Memory organization

- Various organization of the physical memory are included in this section. In order to close up the speed gap between Cache and main memory. And interleaving technique is represented allow pipelined access of the parallel memory modules.
- The memory design goal (interleaving goal) is to broaden the effective memory bandwidth so that more memory words can be accessed per unit time.
- The ultimate purpose is to match the memory bandwidth with the bus bandwidth and with the processor bandwidth.

# Interleaving models

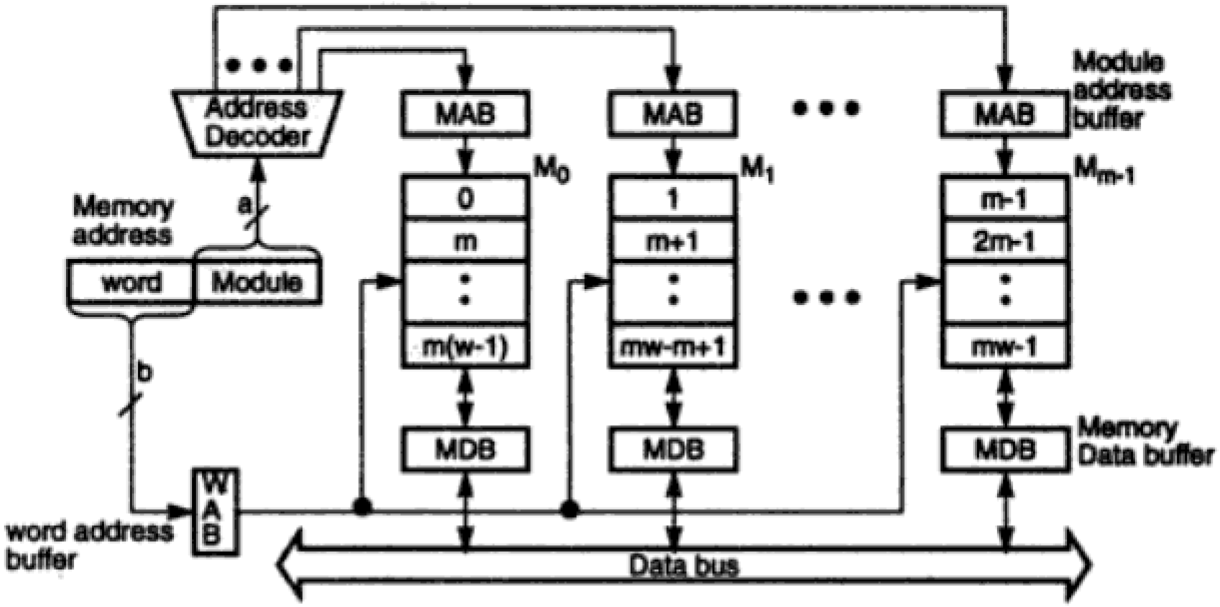
Low order interleaving

Low order interleaving spreads contiguous memory location across the modules horizontally. This implies that the low order bits of the memory address are used to indentify the memory module. High order bits are the word addresses (displacement) within each module

High order interleaving

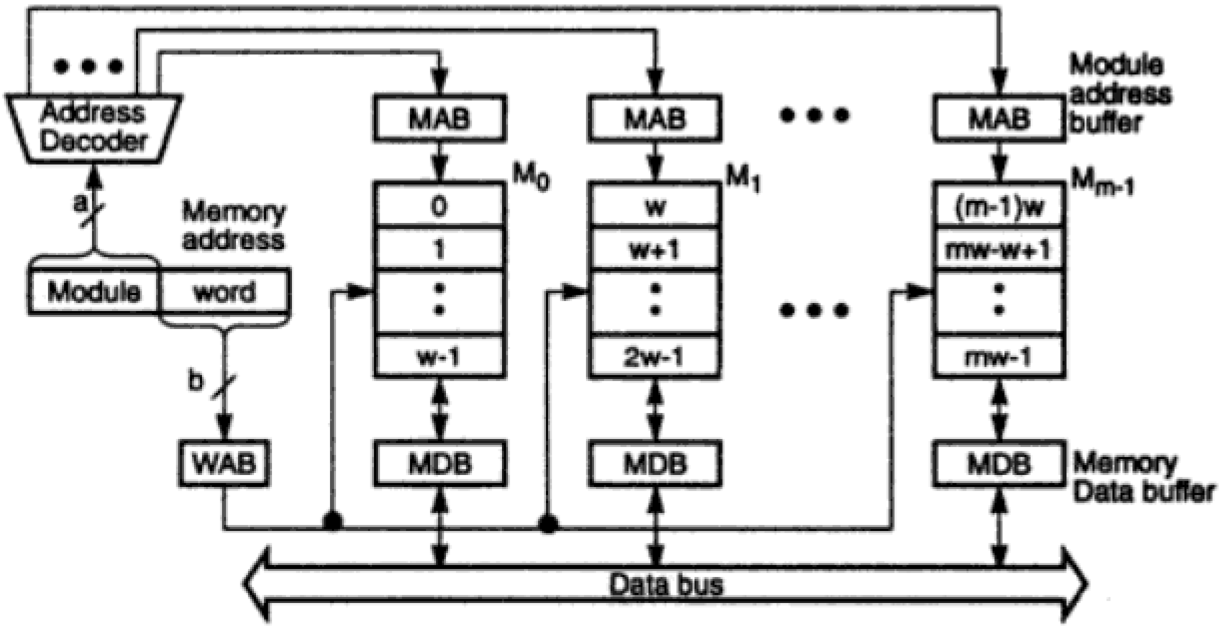
High order interleaving uses the high order bits as the module address and the low order bits as the word address within each module.

# Low order  interleaving



(a) Low-order $m$-way interleaving (the C-access memory scheme)

# High order  interleaving



(b) High-order $m$-way interleaving

**15 Two interleaved memory organizations with $m = 2^a$ modules a**

# Bus Arbitration

- The arbitration procedure comes into picture whenever there are more than one processors requesting the services of bus.
- Because only one unit may at a time be able to transmit successfully over the bus, there is some selection mechanism is required to maintain such transfers. This mechanism is called as *Bus Arbitration.*
- Bus arbitration decides which component will use the bus among various competing requests.
- A selection mechanism must be based on fairness or priority basis.
- Various methods are available that can be roughly classified as either centralized or distributed.
- There are three arbitration schemes-
  - Daisy – chaining
  - Polling
  - Independent requesting

# UNIT-3

## ADVANCED PROCESSORS

# Instruction Set Architecture

- Program Operations
  - Load a register with a given number
  - Copy data from register to another
  - Carry out arithmetic & logical operations on a data word or a pair of data words
  - Test a bit or word & jump, or not, depending on result of the test
  - Jump to a point in the program
  - Jump to a subroutine
  - Carry out a special control operation

# Instruction Classification

- Instruction Types
  - Data transfers
  - Arithmetic, logic
  - Rotates, shifts
  - Bit set, bit reset, & bit test
  - General-purpose, CPU control
  - Jumps
  - Calls, returns

# Data Transfer Instructions

- Used to transfer data from one location to another
  - Register to Register, Register to Memory, Memory to Register
- MOV
  - **`MOVLW 3         ; W ← 03h`**
  - **`MOVWF R1        ; R1 ← 03h`**
  - MOV is same as LDA (Load) in textbook

# Transfer Instructions (Jump Instructions)

- Can be used to jump here & there within program
- Can be used to control loops
  - GOTO – **`GOTO loop ;go to loop label`**
  - CALL – **`CALL delay ;call delay subroutine`**

# CISC versus RISC

- The two conceptions of architectures are in concurrent relation from the 1975: .
- CISC (Complex Instruction Set Computer)
- RISC (Reduced Instruction Set Computer)

# CISC with common cache memory

- CISC machine with microprogramming control
  - Control unit
  - CPU
  - ROM microinstr.
  - Cache
  - Main memory

# CISC with overlapping

- CISC processors became use overlapping of execution and reading following instruction
- In the 95% of time it is executed over 25% instructions of all number of kinds
- Processors were equipped by too much complex instruction decoding controller, which consumes too much area of chip, it was solved as Finite State Machine in hardware (complications with testing)

# Basic properties of RISC processors

- Small number of relatively simple instructions
- Every instruction is executed in only one instruction cycle
- Controller is realised by logic net of gates
- Operations use only registers of notepad memory
- High number of programme usable registers
- Introduced changes in hardware need necessary of optimisation of programme by optimising compilation
- It is addressed to 192 registers for reading of operands and for loading of result of operations
- The memory is used only by Load/Save (Load/Store) instructions

# RISC – controller with simple logic

- Simple
- Logic Controller CPU
- I - Cache
- D - Cache
- Main Memory

# The state of the arts of RISC

- The RISC machines bring the standard solution as L/S architecture (Load/Store) z The RISC are able to issue several instructions in one instruction cycle z This is called superscalar technique

# Superscalar static architecture

- The computational power enhancement is attained by concatenating of computational units (the instruction execution is more parallel)
- The parallel reading of instruction is used: the architecture is called superscalar static architecture
- Hardware solution: very long instruction word (VLIW), which is partially decoded

# Superscalar dynamic architecture

- The power enhancement is ensured by planning of several instructions in every step of execution
- Instructions of branching causes that some results of computation are lost, because the reading must precede instruction execution
- This mechanism is used namely during reading, and writing data into memory

# VLIW

- In VLIW and superscaler both the method pipelining and replication are employed to achieve higher performace.
- In both of them it involves specifying multiple independent operations per instruction.
- However the two architectures differ in a way they specify such instructions.
- This kind of complexity of specifying instructions in superscaler computer is at Hardware level
- While as it as software (Compiler) level in VLIW.

# VLIW vs Super Scalar

- Super Scalar architectures, in contrast, use dynamic scheduling that transform all ILP complexity to the hardware

- This leads to greater hardware complexity that is not seen in VLIW hardware

- VLIW chips don’t need most of the complex circuitry that Super Scalar chips must use to coordinate parallel execution at runtime

# VLIW principles

1. The compiler analyzes dependence of all instructions among sequential code, tries to extract as much parallelism as possible.
2. Based on the analysis, the compiler re-codes the piece of sequential code in VLIW instruction words.
3. Finally, the work left with VLIW hardware is only fetch the VLIWs from cache, decode them, and then dispatch the independent primitive instructions to corresponding function units and execute.

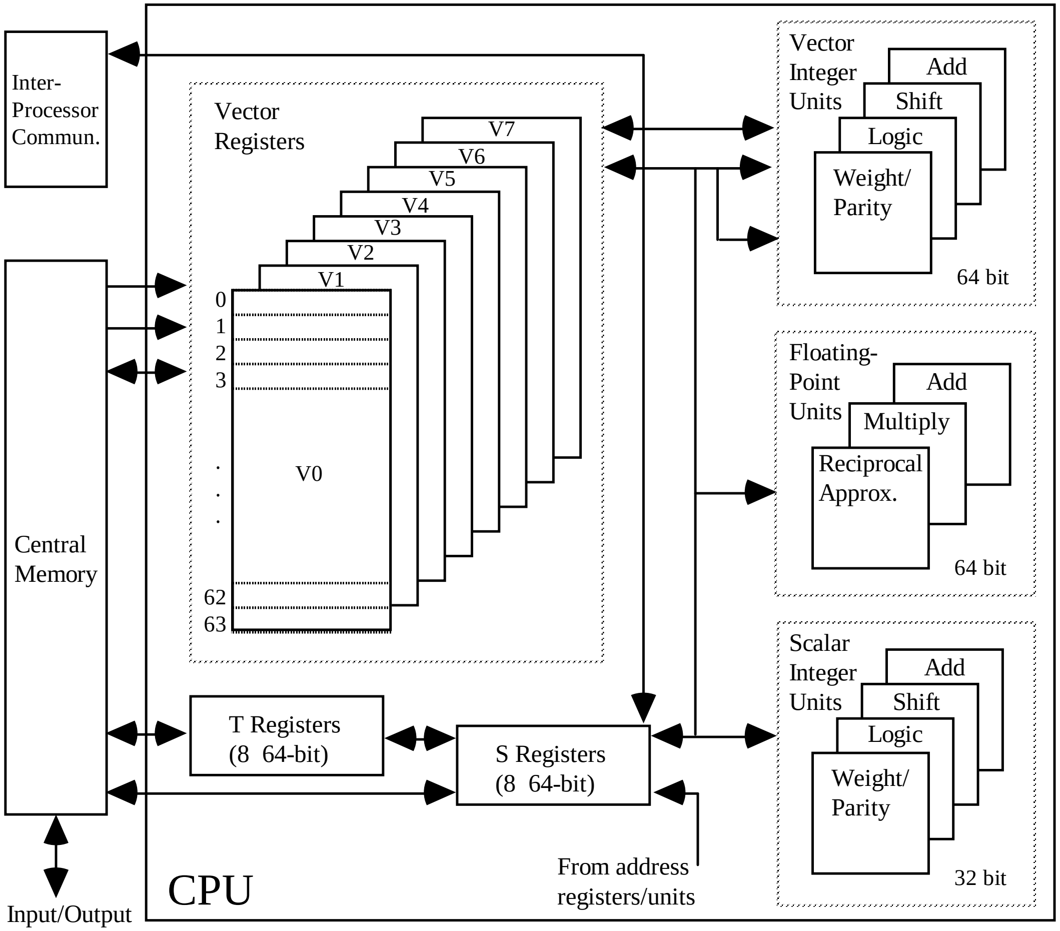
# SIMD - Single Instruction Multiple Data

- Originally thought to be the ultimate massively parallel machine!
- Some machines built
  - Illiac IV
  - Thinking Machines CM2
  - MasPar
  - Vector processors *(special category!)*
- SIMD performance depends on
  - Mapping problem ⇨ processor architecture
  - Image processing
    - Maps naturally to 2D processor array
    - Calculations on individual pixels trivial
    - ⌛ Combining data is the problem!
  - Some matrix operations also

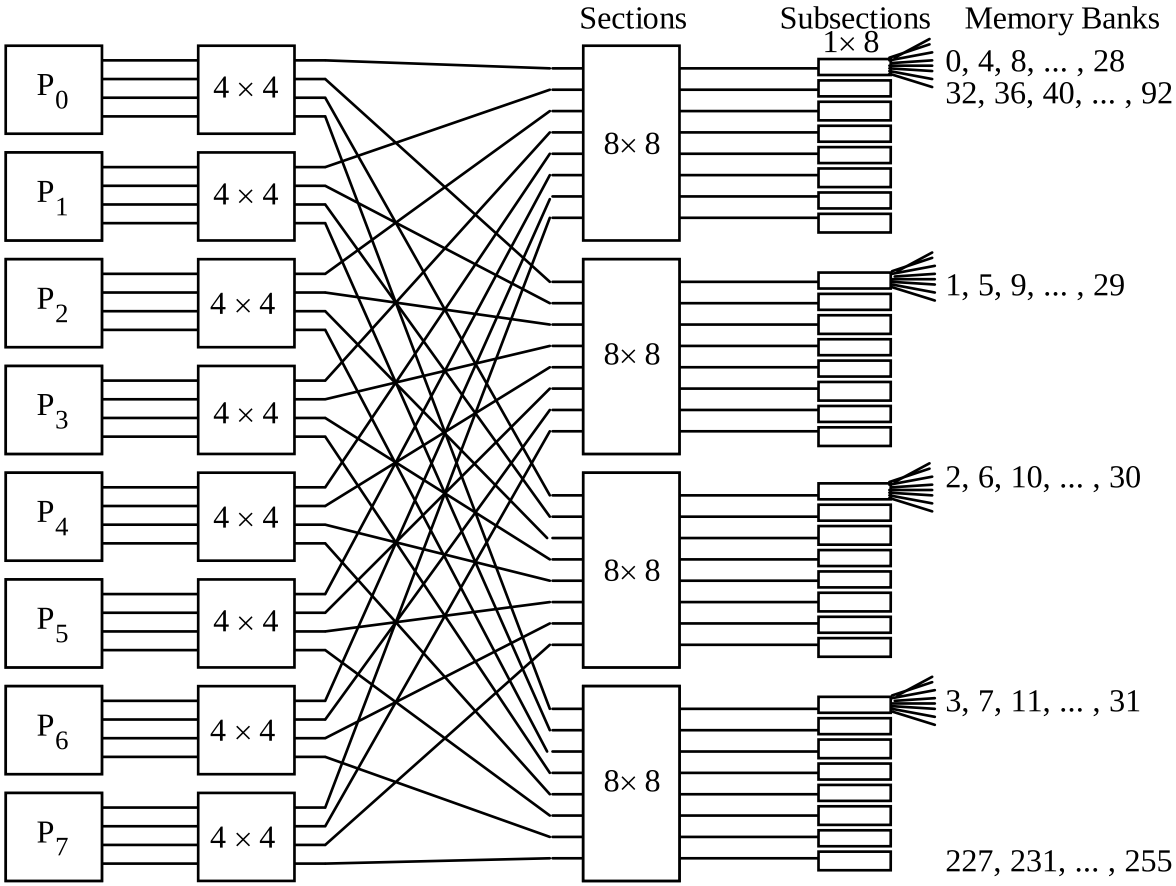
# Vector Processor

- Also called an Array Processor.
- Runs multiple mathematical operations on multiple data elements simultaneously.
- Common in supercomputers of the 1970’s 80’s and 90’s.
- Today most CPU designs contains at least some vector processing instructions, typically referred to as SIMD.
- Typically operate on a few vectors elements per clock cycle in a pipeline v. SIMD which will operate on all at once.

# Cray Y-MP


Inter-
Processor
Commun.
Central
Memory
Input/Output
CPU
Vector
Registers
V7
V6
V5
V4
V3
V2
V1
V0
0
1
2
3
.
.
.
62
63
T Registers
(8 64-bit)
S Registers
(8 64-bit)
From address
registers/units
Vector
Integer
Units
Add
Shift
Logic
Weight/
Parity
64 bit
Floating-
Point
Units
Add
Multiply
Reciprocal
Approx.
64 bit
Scalar
Integer
Units
Add
Shift
Logic
Weight/
Parity
32 bit

# Cray Y-MP’s Interconnection Network



# Unit-4

## MULTI PROCESSOR AND MULTI COMPUTERS

# Interconnection Structures

- The components that form a multiprocessor system are CPUs, IOPs connected to input—output devices, and a memory unit that may be partitioned into a number of separate modules.
- The interconnection between the components can have different physical configurations, depending on the number of transfer paths that are available between the processors and memory in a shared memory system or among the processing elements in a loosely coupled system.
- There are several physical forms available for establishing an interconnection network, Some of these schemes are presented in this section:
- 1. Time-shared common bus
- 2, Multiport memory
- 3. Crossbar switch
- 4. Multistage switching network
- 5. Hypercube system

# Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU. This is shown in Fig. 13- 3 for four CPUs and four memory modules (MMs). Each processor bus is connected to each memory module. A processor bus consists of the address, data, and control lines required to communicate with memory.
- The memory module is said to have four ports and each port accommodates one of the buses. The module must have internal control logic to determine which port will have access to memory at any given time.
- Memory access conflicts are resolved by assigning fixed priorities to each memory port.
- Thus CPU 1 will have priority over CPU 2, CPU 2 will have priority over CPU 3, and CPU 4 will have the lowest priority.

# Crossbar Switch

- The crossbar switch organization consists of a number of crosspoints that are placed at intersections between processor buses and memory module paths.
- Figure 13-4 shows a crossbar switch interconnection between four CPUs and four memory modules.
- The small square in each crosspoint is a switch that determines the path from a processor to a memory module.
- Each switch point has control logic to set up the transfer path between a processor and memory.
- It examines the address that is placed in the bus to determine whether its particular module is being addressed.

# Pipelining

- Consider the following decomposition for processing the instructions
  - Fetch instruction – Read into a buffer
  - Decode instruction – Determine opcode, operands
  - Calculate operands (i.e. EAs) – Indirect, Register indirect, etc.
  - Fetch operands – Fetch operands from memory
  - Execute instructions - Execute
  - Write result – Store result if applicable
- Overlap these operations to make a 6 stage pipeline
- The textbook uses a 5 stage pipeline (Fetch/Decode/Operand Fetch/Execute/Write Back)

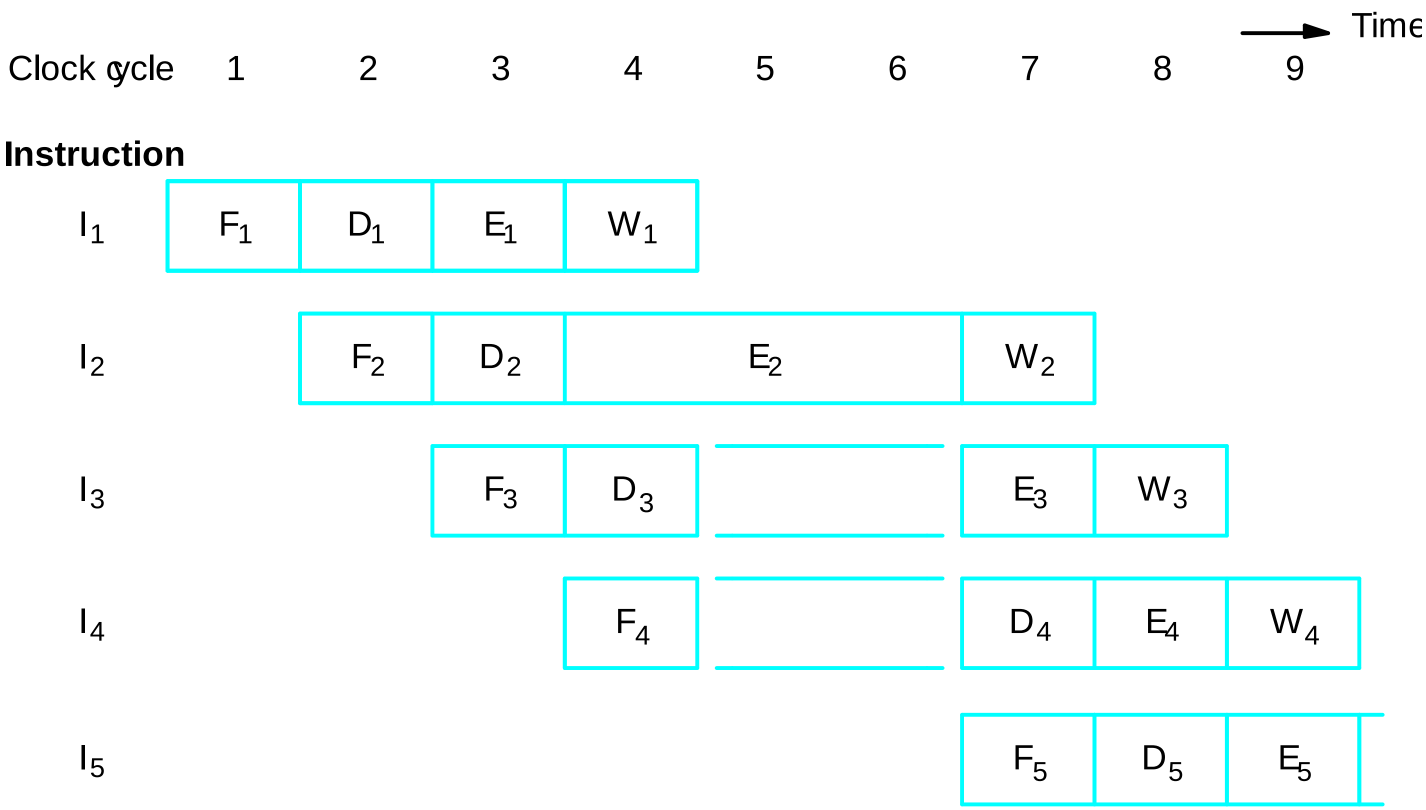
# Pipelined Processors

- It is technique of decomposing a sequential process into sub operation, with each sub operation completed in dedicated segment.
- Pipeline is commonly known as an assembly line operation.
- It is similar like assembly line of car manufacturing.
- First station in an assembly line set up a chasis, next station is installing the engine, another group of workers fitting the body.

# Pipeline Performance

- The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages.
- However, this increase would be achieved only if all pipeline stages require the same time to complete, and there is no interruption throughout program execution.
- Unfortunately, this is not true.
- Floating point may involve many clock cycle

# Pipeline Performance



| Clock cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **Instruction** | | | | | | | | | |
| $I_1$ | $F_1$ | $D_1$ | $E_1$ | $W_1$ | | | | | |
| $I_2$ | | $F_2$ | $D_2$ | $E_2$ | | | $W_2$ | | |
| $I_3$ | | | $F_3$ | $D_3$ | | | $E_3$ | $W_3$ | |
| $I_4$ | | | | $F_4$ | | | $D_4$ | $E_4$ | $W_4$ |
| $I_5$ | | | | | | | $F_5$ | $D_5$ | $E_5$ |

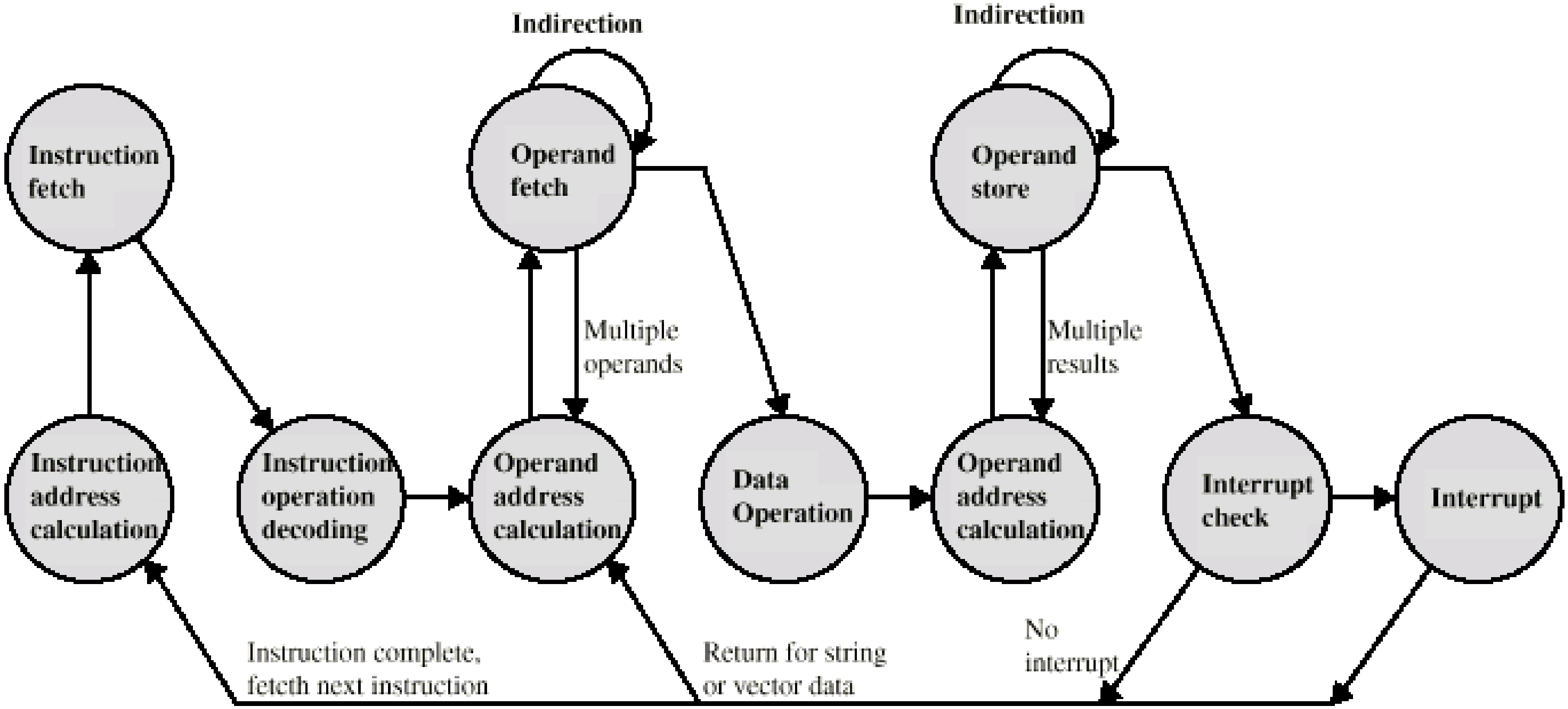
Figure 8.3. Effect of an execution operation taking more than one clock cycle.

# Pipeline Performance

- The previous pipeline is said to have been stalled for two clock cycles.
- Any condition that causes a pipeline to stall is called a hazard.
- Data hazard – any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. So some operation has to be delayed, and the pipeline stalls.
- Instruction (control) hazard – a delay in the availability of an instruction causes the pipeline to stall.[cache miss]
- Structural hazard – the situation when two instructions require the use of a given hardware resource at the same time.
- Again, pipelining does not result in individual instructions being executed faster; rather, it is the throughput that increases.
- Throughput is measured by the rate at which instruction execution is completed.
- Pipeline stall causes degradation in pipeline performance.
- We need to identify all hazards that may cause the pipeline to stall and to find ways to minimize their impact.

# Instruction pipeline design

- Instruction Cycle State Diagram


Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
No interrupt
Instruction complete, fetcth next instruction
Return for string or vector data

# Instruction Pipeline

- An **instruction pipeline** is a technique used in the design of computers and other digital electronic devices to increase the instructions that can be executed in a unit of time.

# Unit-5

## DATA FLOW COMPUTERS AND VLSI COMPUTATIONS

# data flow computer architecture-Dataflow languages

- Main characteristic: The *single-assignment rule*
  - **A variable may appear on the left side of an assignment only once within the area of the program in which it is active**.
- Examples: VAL, Id, LUCID
- A dataflow program is compiled into a dataflow graph which is a directed graph consisting of named nodes, which represent instructions, and arcs, which represent data dependences among instructions.
  - The dataflow graph is similar to a dependence graph used in intermediate representations of compilers.
- During the execution of the program, data propagate along the arcs in data packets, called *tokens*.
- This flow of tokens enables some of the nodes (instructions) and fires them.

# Two important characteristics of dataflow graphs

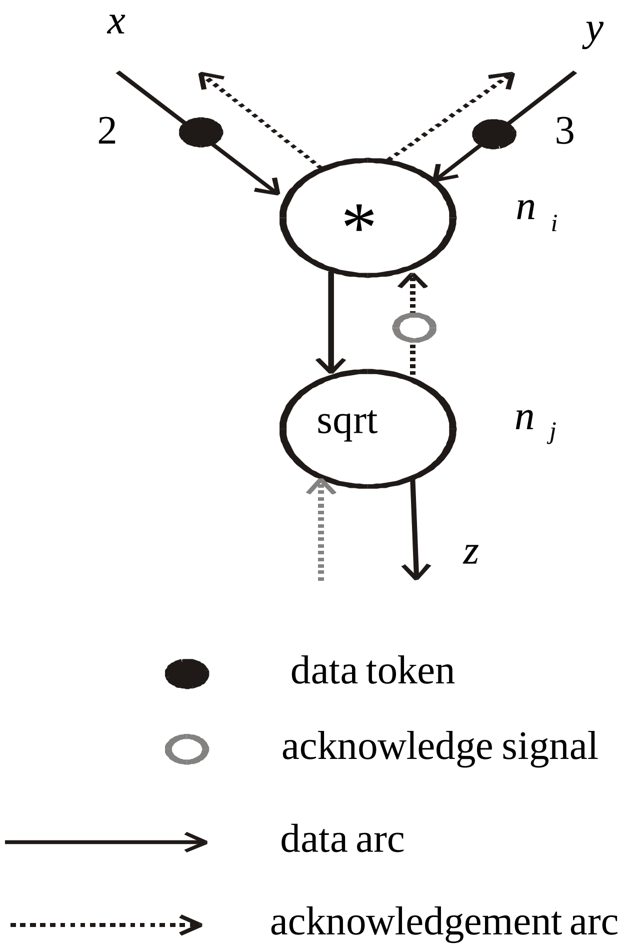
- *Functionality*: The evaluation of a dataflow graph is equivalent to evaluation of the corresponding mathematical function on the same input data.

- *Composability*: Dataflow graphs can be combined to form new graphs.

# Static Dataflow

- A *dataflow graph* is represented as a collection of *activity templates*,
  each containing:
  - the opcode of the represented instruction,
  - operand slots for holding operand values,
  - and destination address fields, referring to the operand slots in sub-sequent activity templates that need to receive the result value.
- Each *token* consists only of a value and a destination address.

# Dataflow graph and Activity template



# Acknowledgement signals

- Notice, that different tokens destined for the same destination cannot be distinguished.
- Static dataflow approach allows at most one token on any one arc.
- *Extending the basic firing rule* as follows:
  - **An enabled node is fired if there is no token on any of its output arcs**.
- Implementation of the restriction by *acknowledge signals* (additional tokens ), traveling along additional arcs from consuming to producing nodes.
- The firing rule can be changed to its original form:
  - **A node is fired at the moment when it becomes enabled**.
- Again: structural hazards are ignored assuming unlimited resources!

# Dynamic Dataflow

- Each loop iteration or subprogram invocation should be able to execute in parallel as a separate instance of a reentrant subgraph.
- The replication is only conceptual.
- Each *token* has a *tag*:
  - address of the instruction for which the particular data value is destined
  - and context information
- Each arc can be viewed as a bag that may contain an arbitrary number of tokens with different tags.
- The *enabling and firing rule* is now:

  **A node is enabled and fired as soon as tokens with identical tags are present on all input arcs**.
- Structural hazards ignored!

# What are Systolic Arrays?

- This is a form of **pipelining**, sometimes in more than one dimension.
- Machines have been constructed based on this principle, notable the **iWARP**, fabricated by Intel.
- *Laying out algorithms in VLSI'*
  - efficient use of hardware
  - not general purpose
  - not suitable for large I/O bound applications
  - control and data flow must be regular
  - The idea is to exploit VLSI efficiently by laying out algorithms (and hence architectures) in 2-D (not all systolic machines are 2-D, but probably most are)

- Simple cells
- Each cell performs one operation
  - (usually)

# Systolic Computing

- **Definition 1.**
- sys·to·le (sîs¹te-lê) noun
- The rhythmic contraction of the heart, especially of the ventricles, by which blood is driven through the aorta and pulmonary artery after each dilation or diastole.
- [Greek sustolê, contraction, from sustellein, to contract. See systaltic.]
- — sys·tol¹ic (sî-stòl¹îk) adjective
- American Heritage Dictionary
- **Definition 2.**
- Data flows from memory in a rhythmic fashion, passing through many processing elements before it returns to memory.
- **Definition 3.**
- **A set of simple processing elements with regular and local connections which takes external inputs and processes them in a predetermined manner in a pipelined fashion**

# Systolic computers have both pipelining and parallelism

- This is good for computation-intensive tasks but not I/O-intensive tasks
    - e.g. signal processing
- Most designs are simple and regular in order to keep the VLSI implementation costs low
    - programs with simple data and control flow are best
- Systolic computers show both pipelining and parallel computation

# What are the functions of a cell in a Systolic System?

- Systolic systems consists of an array of PE(Processing Elements)
  - processors are called cells,
  - each cell is connected to a small number of nearest neighbours in a mesh like topology.

- Each cell performs a sequence of operations on data that flows between them.

- Generally the operations are the same in each cell.

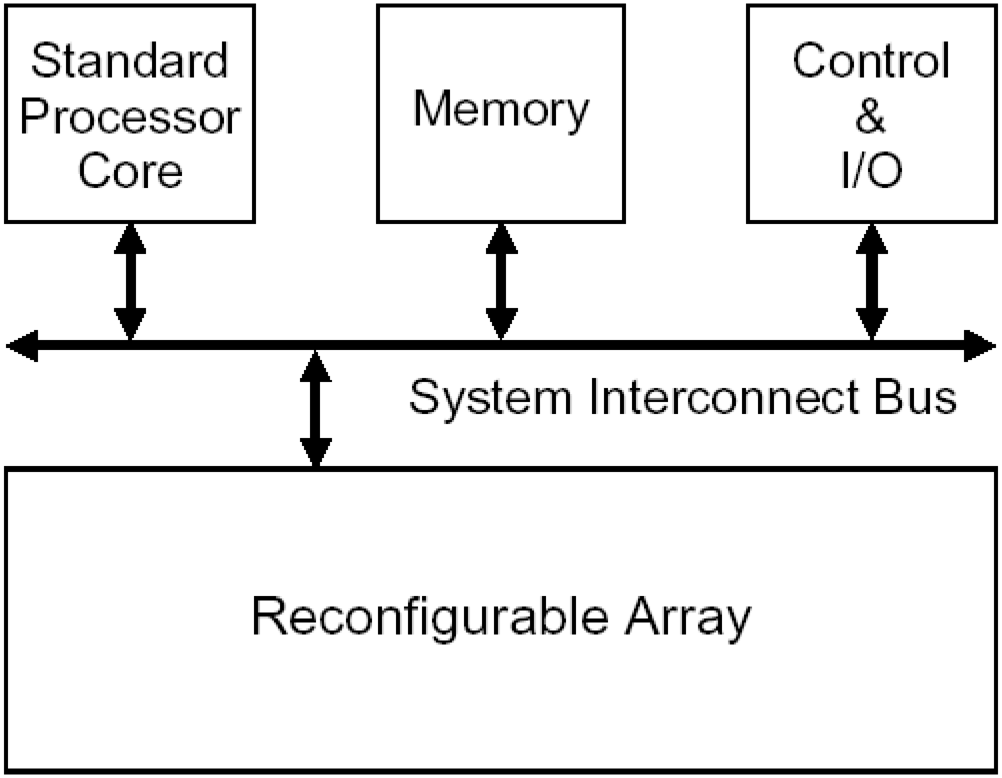
- Each cell performs an operation or small number of operations on a data item and then passes it to its neighbor.

- Systolic arrays compute in “lock-step” with each cell (processor) undertaking alternate compute/communicate phases.

# Reconfigurable processor array

- Reconfigurable Computing (RC) is a computing paradigm in which algorithms are implemented as a temporally and spatially ordered set of very complex tasks. These tasks are executed on a large set of interconnected programmable hardware elementscomputing paradigm - defines the basic RC computing model without reference to implementation.
- very complex tasks – commonly referred to as **configurations** RC tasks require more time than general purpose computing instructions and more area than the typical general purpose execution unit.
- Spatial and temporal partitioning – algorithms are decomposed into tasks in both the space and time domains.
- hardware elements - at their core RC devices consist of a very large set of simple programmable elements collectively called **Reconfigurable Execution Unit (REU)**

# Basic Architecture of today's commercial reconfigurable processor



# vlsi arithmetic models

- Digital Computer Arithmetic belongs to Computer Architecture, however, it is also an aspect of logic design.
- The objective of Computer Arithmetic is to develop appropriate algorithms that are utilizing available hardware in the most efficient way.
- Ultimately, speed, power and chip area are the most often used measures, making a strong link between the algorithms and technology of implementation

# Basic Operations

- Addition
- Multiplication
- Multiply-Add
- Division
- Evaluation of Functions
- Multi-Media