MIT Art Design & Technology University's

MIT School of Engineering, Pune

COMPUTER ORGANIZATION & MICROPROCESSORS INTERFACING

UNIT I : COMPUTER EVOLUTION

Computer Architecture & Computer Organization

- Architecture is those attributes visible to the programmer
  - Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques.
  - e.g. Is there a multiply instruction?
- Organization is how features are implemented
  - Control signals, interfaces, memory technology.
  - e.g. Is there a hardware multiply unit or is it done by repeated addition?

Structure & Function

- Structure is the way in which components relate to each other
- Function is the operation of individual components as part of the structure

Function

- All computer functions are:
  - Data processing
  - Data storage
  - Data movement
  - Control

Evolution of Computer (1)

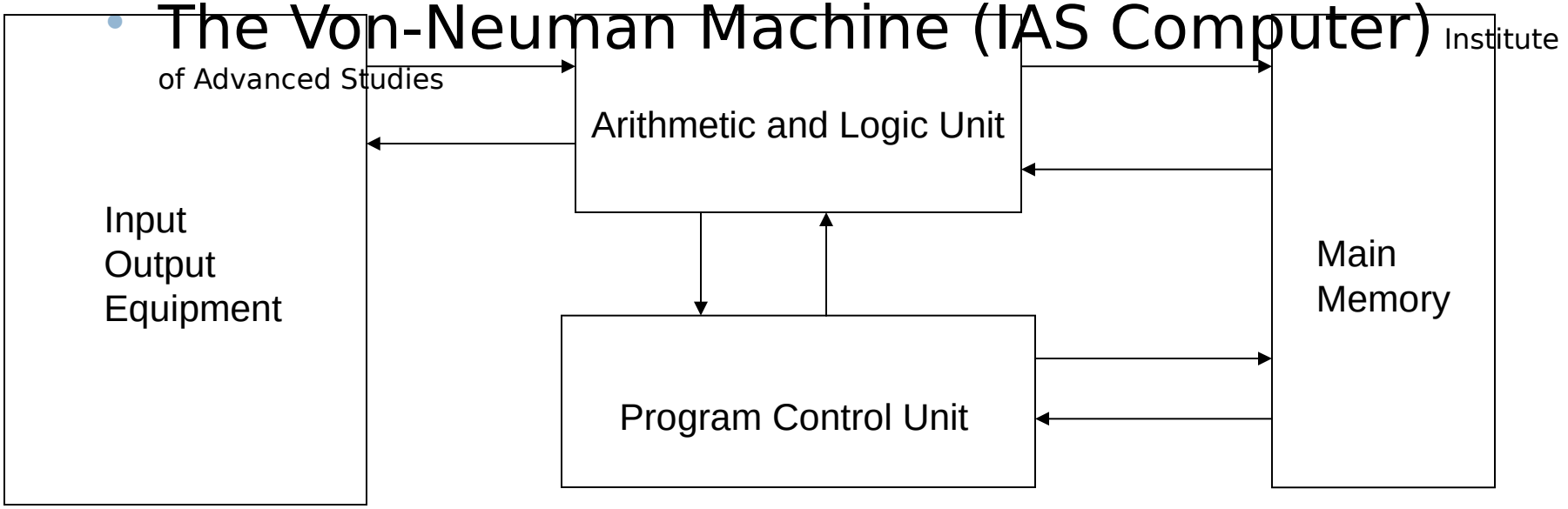
- Zeroth Generation : Mechanical Era (1600-1940)
  - Blaise Pascal (1642) Pascaline – Calulator 1652
  - Von Leibniz (1646-1716) Automatic Multiplication 1671
  - Babbage's Differential Engine (1822-36) tabulate polynomial fn, logarithmic, trigonometric fn, computation of astronomical and mathematical tables
  - Babbage's Analytical Engine (1837) first design for a general-purpose compute, arithmetic logic unit, control flow, integrated memory
  - Herman Hollerith (1880) – Punched Cards, tabulating and sorting machines as well as the first key punch, IBM
  - Konrad Zuse (1938):Automatic Calculating Machine based on Electromagnetic mechanism, Z1, world's first program-controlled computer.
  - Howard Aiken (1943) : Howard Mark1 (72 words of 23 decimal digit), Automatic Sequence Controlled Calculator (ASCC), World War-II

Evolution of Computer (2)

- First Generation : Vacuum Tubes (1946-1957)
  - ENIAC (Electronic Numerical Integrator and Calculator)
    - Started in 1943
    - For preparing trajectory tables of weapons.
    - John Mauchly and John Eckert in University of Pennsylvania
    - 30 tones, 15,000 square feet, 18,000 tubes, 140 KW
    - 5000 addition/sec, Decimal, 20 Accumulators
    - Programming by Plugging and Unplugging of switches
    - Results were punched on cards or printed on type writer.
    - Completed in 1946

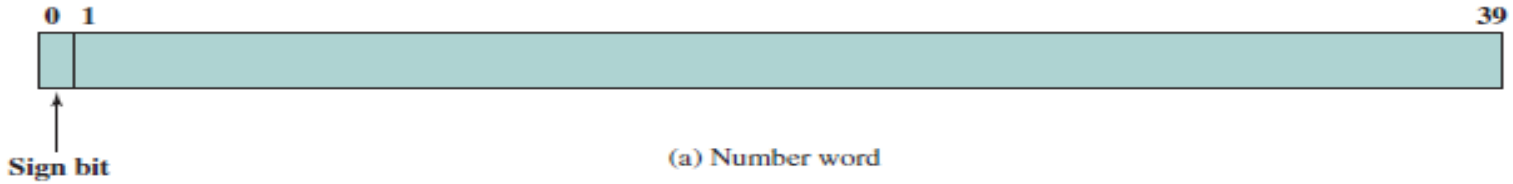
Evolution of Computer (3)

- First Generation (contd.)
  - Stored Program Concept
  - EDVAC (Electronic Discrete Variable Automatic Computer)
  - The Von-Neuman Machine (IAS Computer) Institute of Advanced Studies

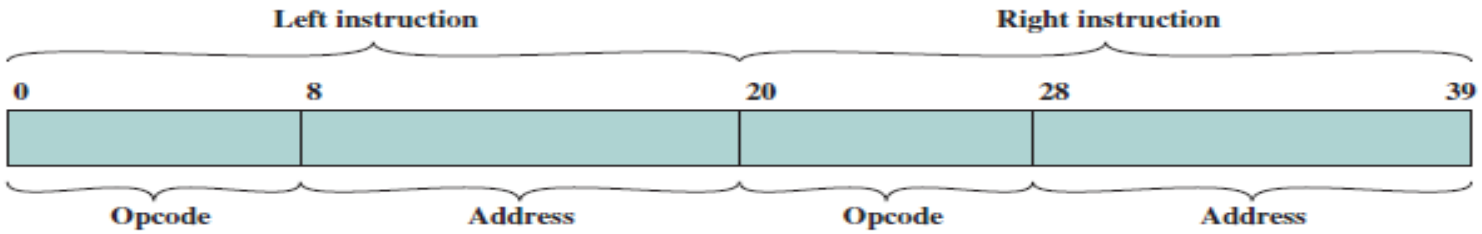


Evolution of Computer (4)

- First Generation (Von-Neuman Machine contd..)
  - 1000 x 40 bit words memory
    - Binary number



(a) Number word

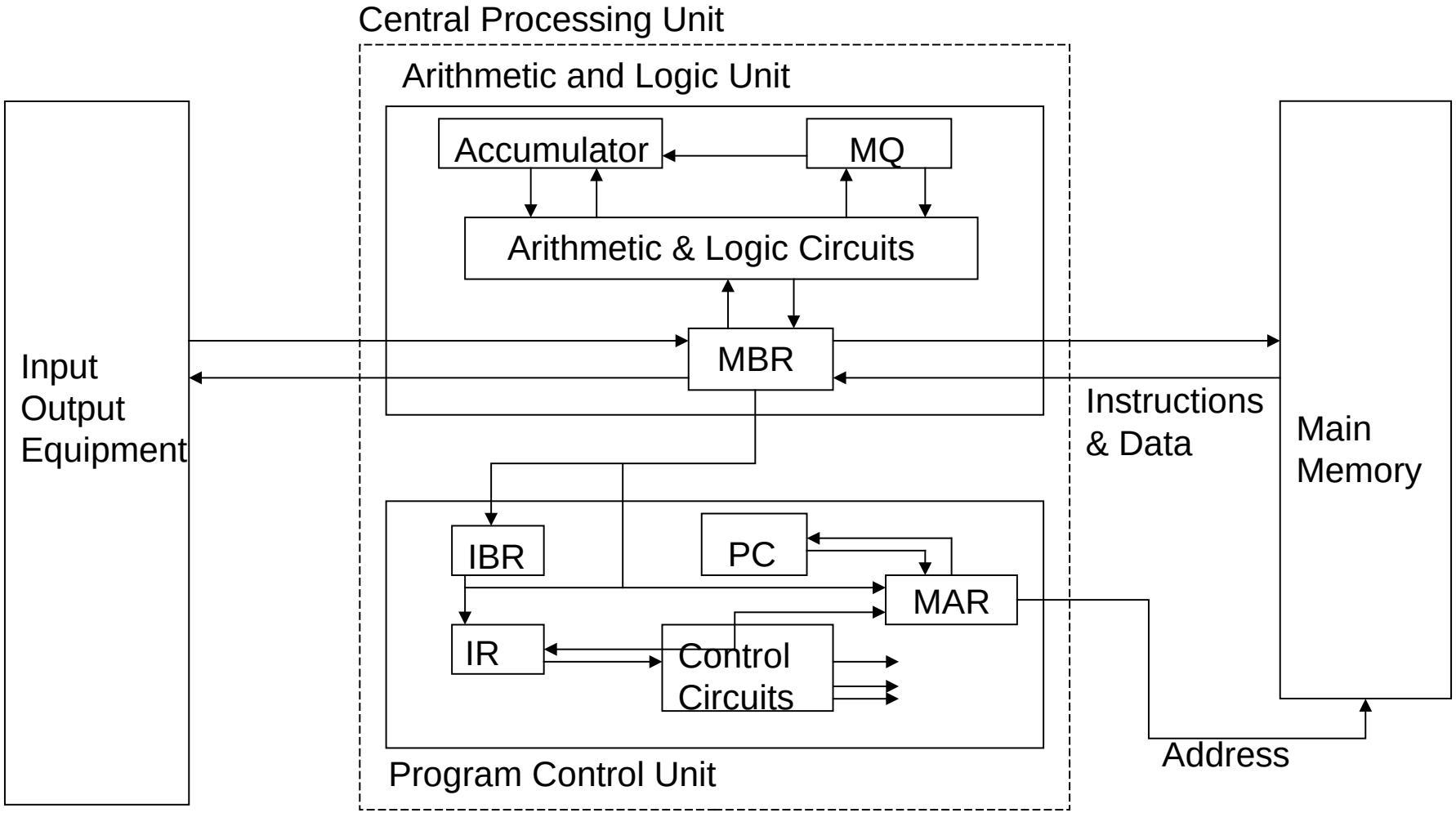


(b) Instruction word

Ref: Computer Organization and Architecture by William Stallings

Evolution of Computer (5)

- Detailed Von Neuman Architecture



Evolution of Computer (6)

- Von Neuman Instruction
  - Supports 21 different instruction.

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD – M(X)	Transfer –M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD – |M(X)|	Transfer –|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP + M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	JUMP + M(X,20:39)	If number in the accumulator is nonnegative, take next instruction from right half of M(X)

Ref: Computer Organization and Architecture by William Stallings

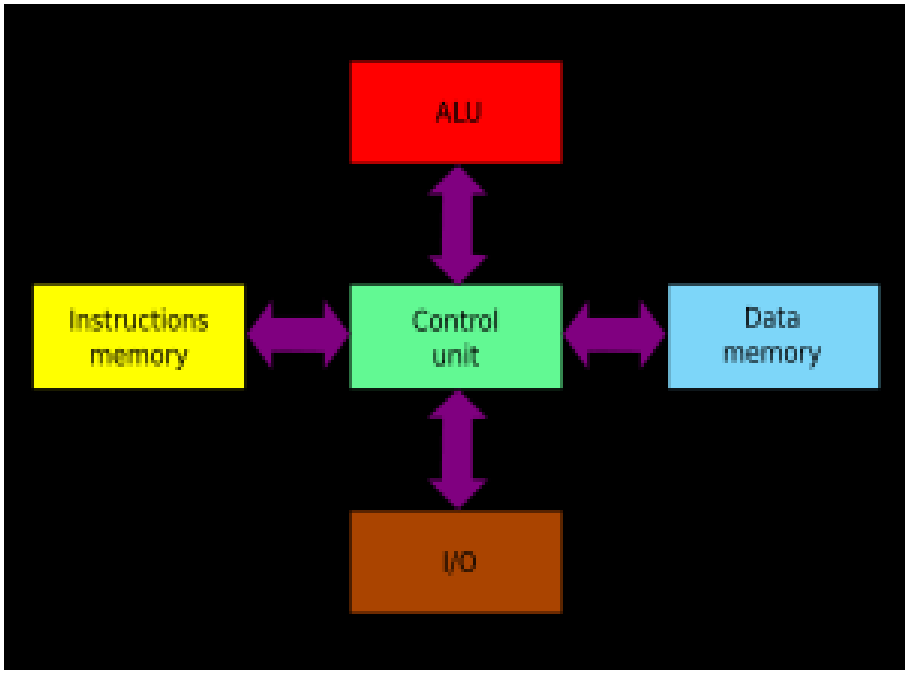
Evolution of Computer (7)

Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; that is, shift left one bit position
	00010101	RSH	Divide accumulator by 2; that is, shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

Evolution of Computer (8)

Harvard Architecture

- Computers with separate program and data memories implemented in ROMs and RAMs, respectively are called Harvard architecture.
- Harvard architecture has separate buses for data and program memory.
- Both memories can be accessed simultaneously.
- This reduces the chances of resource conflict related to memory accesses.
- This has improved bandwidth over traditional Von Neumann architecture.



Evolution of Computer (9)

- 1947 - Eckert-Mauchly Computer Corporation
  - UNIVAC I (Universal Automatic Computer) for US Bureau of Census 1950 calculations
- Late 1950s - UNIVAC II
  - Faster
  - More memory
- Birth of IBM and Sperry in 1950s.
  - 1953-IBM 701 for scientific calculations
  - 1955-IBM 702 for Business Application

Evolution of Computer (10)

- Second Generation (1955-1965) -Transistors
  - Transistor Replaced vacuum tubes
    - Smaller, Cheaper, Less heat dissipation, Solid State device, Made from Silicon (Sand),Invented 1947 at Bell Labs
  - Multiplexor, Data Channel
  - Concept of System Software
  - PDP-1(Programmed Data Processor) by DEC, IBM 7094 with instruction backup register, large-scale scientific and technological applications

Evolution of Computer (11)

- Third Generation (1965-1971)
  - Concept of Small Scale Integration
  - IBM System 360, designed to cover the complete range of applications, from small to large, both commercial and scientific
  - PDP-8 from DEC 12 bit minicomputer
  - Semiconductor based memory

Evolution of Computer (12)

- Later Generation
  - LSI,VLSI
  - 1971 – 4004 :
    - Developed by Intel
    - First microprocessor
    - All CPU components on a single chip
    - 4 bit
  - Followed in 1972 by 8008
    - 8 bit
    - Both designed for specific applications
  - 1974 - 8080
    - Intel’s first general purpose microprocessor

Pentium Evolution (1)

- 80486
  - sophisticated powerful cache and instruction pipelining
  - built in maths co-processor
- Pentium
  - Superscalar, more than one instruction in parallel
  - Multiple instructions executed in parallel
- Pentium Pro
  - Increased superscalar organization
  - branch prediction
  - data flow analysis
  - speculative execution

Pentium Evolution (2)

- Pentium II
  - MMX technology, runs a multimedia application up to 60% faster
  - graphics, video & audio processing
- Pentium III
  - Additional floating point instructions for 3D graphics
- Pentium 4
  - Further floating point and multimedia enhancements

Designing for Performance

- The cost of computer systems continues to drop dramatically, while the performance and capacity of those systems continue to rise equally dramatically

- The basic building blocks for today's computer miracles are virtually the same as those of the IAS computer from over 50 years ago, while techniques to improve performance are increasingly sophisticated

Driving factors behind the need to design for performance (1)

- Microprocessor Speed
  - Raw speed of the microprocessor will not achieve its potential unless it is fed a constant stream of work to do in the form of computer instructions
    - Pipelining : For example, while one instruction is being executed, the computer is decoding the next instruction
    - Branch prediction: branch prediction increases the amount of work available for the processor to execute

Driving factors behind the need to design for performance (2)

- Microprocessor Speed
  - Raw speed of the microprocessor will not achieve its potential unless it is fed a constant stream of work to do in the form of computer instructions
    - Data flow analysis : The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions
    - Speculative execution : This enables the processor to keep its execution engines as busy as possible by executing

Driving factors behind the need to design for performance (3)

- Performance Balance

An adjusting of the organization and architecture to compensate for the mismatch among the capabilities of the various components

- Interface between processor and main memory

  If memory or the pathway fails to keep pace with the processor's insistent demands, the processor stalls in a wait state, and valuable processing time is lost.

- Solution: Use wide DRAMs & paths

  Include Cache memory or buffering

Driving factors behind the need to design for performance (4)

- The key in all this is balance

Designers constantly strive to balance the throughput and processing demands of the processor components, main memory, I/O devices, and the interconnection structures. This design must constantly be rethought to cope with two constantly evolving factors:

- ✓ The rate at which performance is changing differs greatly from one type of element to another.
- ✓ New applications and new peripheral devices constantly change the nature of the demand on the system

Driving factors behind the need to design for performance (5)

- Improvements in Chip Organization and Architecture
- Approaches to achieving increased processor speed
- Increase the hardware speed of the processor
- Increase the size and speed of caches
- Using parallelism

Microprocessor Vs Microcontroller

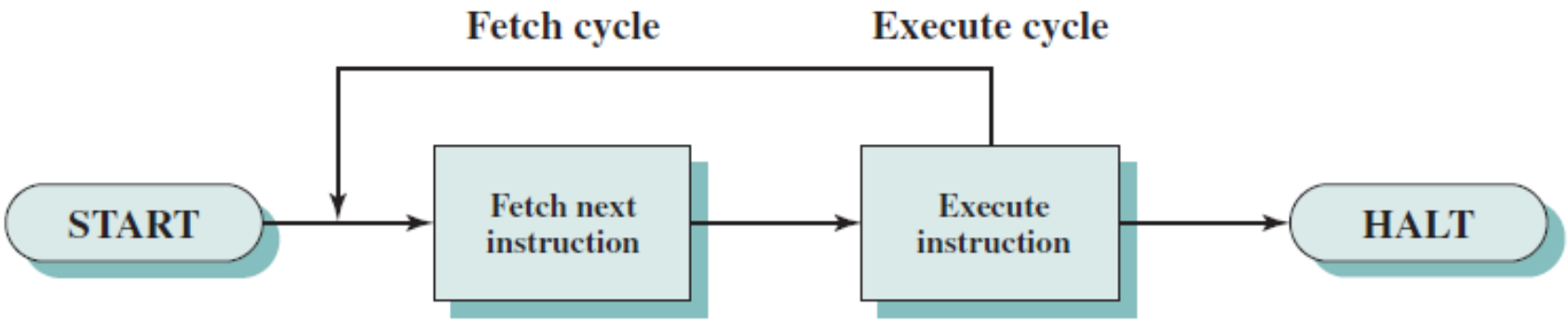
Microprocessor

- CPU is stand-alone, RAM, ROM, I/O, timer are separate
- designer can decide on the amount of ROM, RAM and I/O ports.
- expensive
- versatility
- general-purpose
- High processing power
- High power consumption
- Instruction sets focus on processing-intensive operations
- Typically 32/64 – bit
- Typically deep pipeline (5-20 stages)

Microcontroller

- CPU, RAM, ROM, I/O and timer are all on a single chip
- fixed amount of on-chip ROM, RAM, I/O ports
- for applications in which cost, power and space are critical
- single-purpose (control-oriented)
- Low processing power
- Low power consumption
- Bit-level operations
- Instruction sets focus on control and bit-level operations
- Typically 8/16 bit
- Typically single-cycle/two-stage pipeline

Basic Instruction Cycle



- Processor-memory:
- Processor-I/O:
- Data processing:
- Control:

e.g. Addition operation

- LOAD M(X)---- LOAD 940
- ADD M(X) ---- ADD 941
- STOR M(X) ---- STOR 941

Memory

content address

1004	001
2005	002
3006	003
0001	101
0005	102

CPU REGISTERS

PC--
AC--
IR --

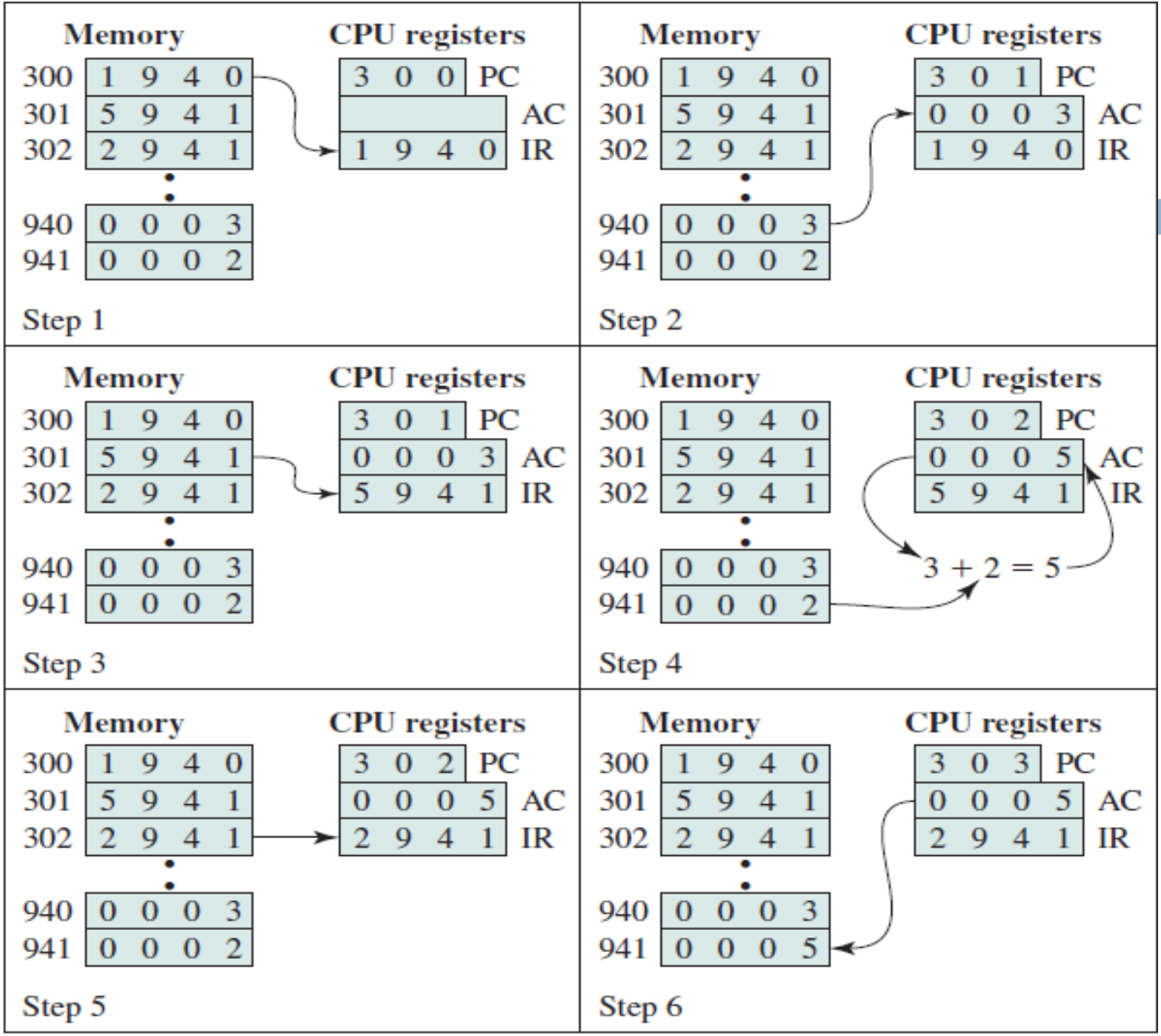
e.g. LOAD 101

ADD 102

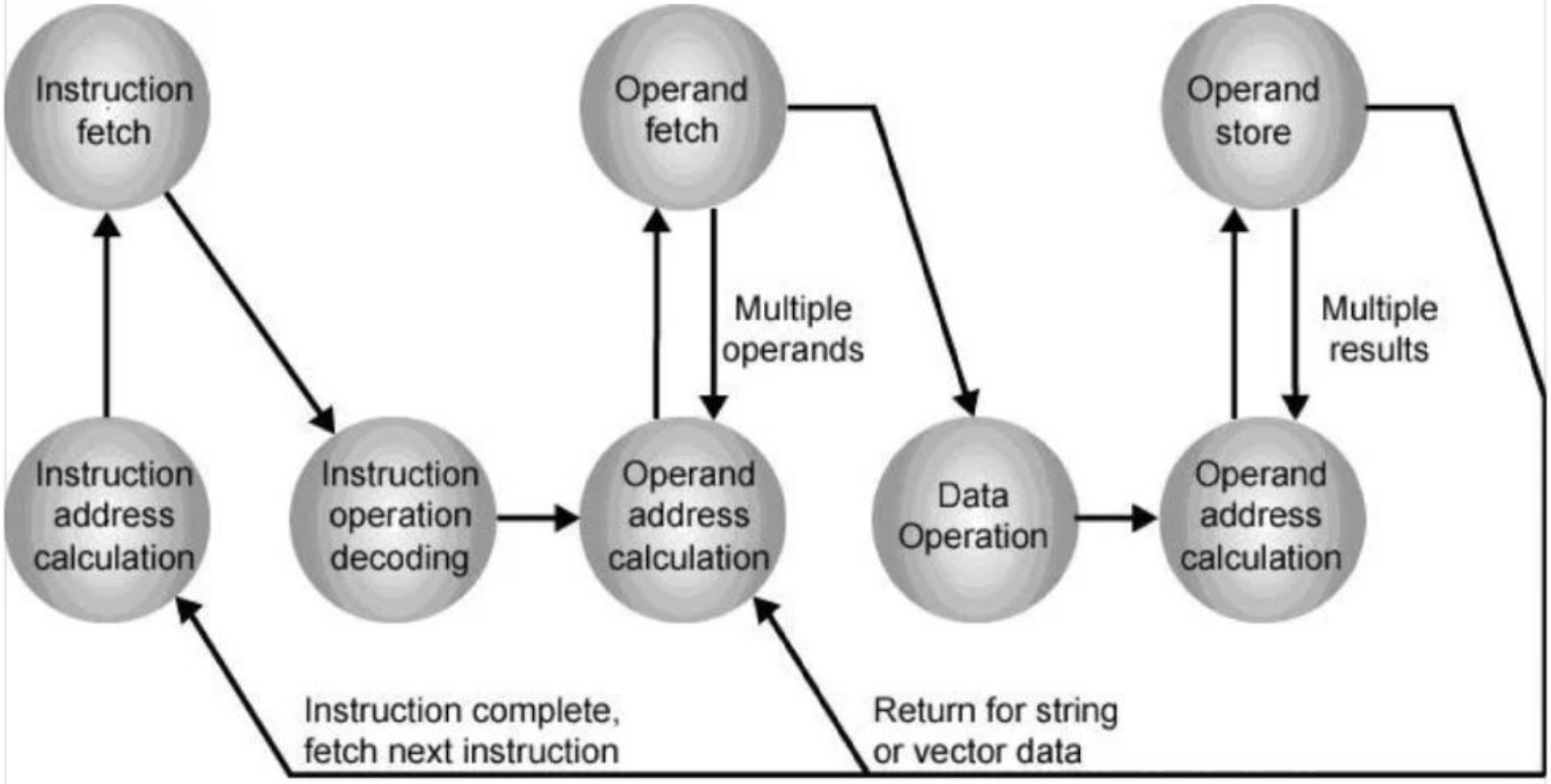
STOR 102

https://jamboard.google.com/d/1oTqgKfmeioIRrQ5KxNYVZbkiz6_G7tvrYfJrc90Gado/edit?usp=sharing

Basic Instruction Cycle



Instruction cycle state diagram



- Instruction address calculation (IAC): Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction.
- Instruction fetch (IF): Read instruction from its memory location into the processor.
- Instruction operation decoding (IOD): Analyse instruction to determine type of operation to he performed and operand(s) to be used.
- Operand address calculation (OAC): If the operation involves reference to an operand in memory or available via I/O. then determine the address of the operand.

- Operand fetch (OF): Fetch the operand from memory or read it in from I/O
- Data operation (DO): Perform the operation indicated in the instruction.
- Operand store (OS): Write the result into memory or out to I/O

Parallel Processing

- Parallel processing can be described as a class of techniques which enables the system to achieve simultaneous data-processing tasks to increase the computational speed of a computer system.
- For instance, while an instruction is being processed in the ALU component of the CPU, the next instruction can be read from memory.
- The primary purpose of parallel processing is to enhance the computer processing capability and increase its throughput, i.e. the amount of processing that can be accomplished during a given interval of time.

Pipelining

- Pipelining

  Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined

Without pipelining = 9/3 minutes = 3m

```
I F S | | | | | |
| | | I F S | | |
| | | | | | I F S (9 minutes)
```

With pipelining = 5/3 minutes = 1.67m

```
I F S | |
| I F S |
| | I F S (5 minutes)
```

Inserting the bottle(I), Filling water in the bottle(F), and Sealing the bottle(S).

Control Flow Architecture

- Instructions of a program are executed in an implied **(sequential) order.**
- In early days instructions were processed **one at a time.**
- Single separate storage structure both instructions and data
- Programming languages also designed with sequential execution semantics.
- eg. von Neuman architrecture

– **bottleneck- the limited bandwidth between the CPU**
and memory compared to the amount of

Data Flow Architecture (1)

- **Motivation**
  - To exploit maximum parallelism inherent in a program.
  - Minimize ordering constraints. No ordering constraints other than data and control dependences.
- **Data Flow Principles**
  - Asynchronous
  - No program counter.
  - The execution of an instruction is based on the availability of its operands.
  - No constraints on sequencing except the

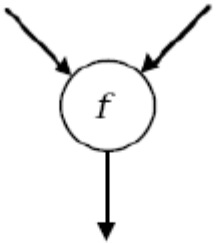
Data Flow Architecture (2)

Data Flow Principles

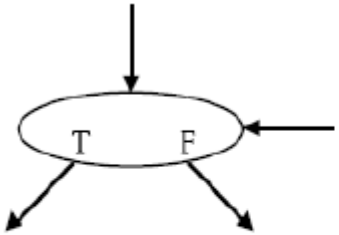
- Instruction fetch and execution are data-driven.
- A dataflow program is represented as a directed graph
- Nodes (or actors) represent instructions and arcs
represent data dependencies between the nodes.
- Objects (data structures or scalar values) are
consumed by an actor (instruction) yielding

Data Flow Architecture (3)

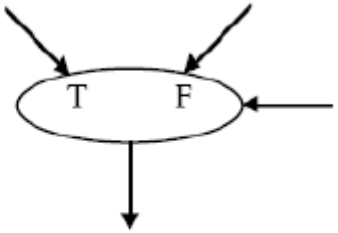
Data Flow Graphs



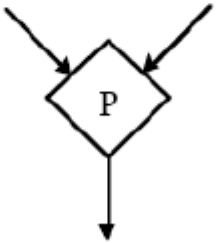
Operator



Switch



Merge



Predicate

Copy

Data Flow Architecture (4)

Data Flow Languages

- There is a special need to provide a high-level language for dataflow computers
- The dataflow graph is not an appropriate programming medium.
- Data flow or applicative languages
  - Value Algorithmic Language (VAL)
  - Irvine Dataflow language (Id)
  - Stream and Iteration in a Single-Assignment Language (SISAL)

Data Flow Architecture (5)

Types of Data Flow Machines

- Static dataflow machines
  - Not allow multiple instances of the same routines to be executed simultaneously
  - Conventional memory.
- Dynamic dataflow machines
  - Allow multiple instances
  - Tags to distinguish between different instances of a node. A tag is associated with each token that identifies the context in which a particular token was generated.
  - An actor is considered executable when its input arcs contain a set of tokens with identical tags.
  - Content addressable memory. (CAM)

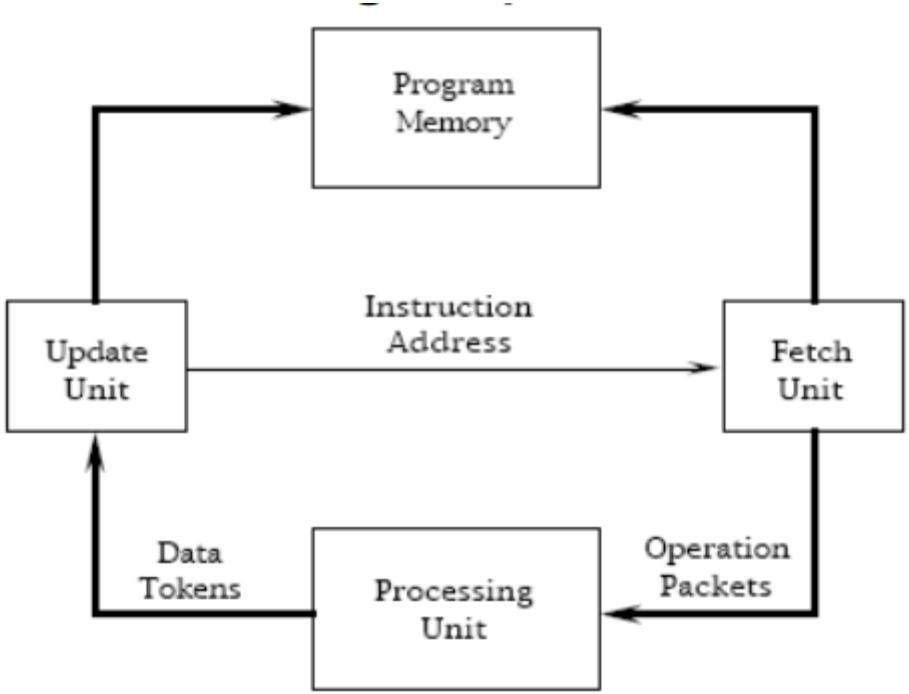
Data Flow Architecture (6)

Static Data Flow Machine

- Proposed by Dennis and his research group at MIT

- Program Memory contains instruction templates which represents the nodes in a dataflow graph.

- Contains an operation



Opcode	
PB	Operand 1
PB	Operand 2
Destination 1	
Destination 2	

Data Flow Architecture (7)

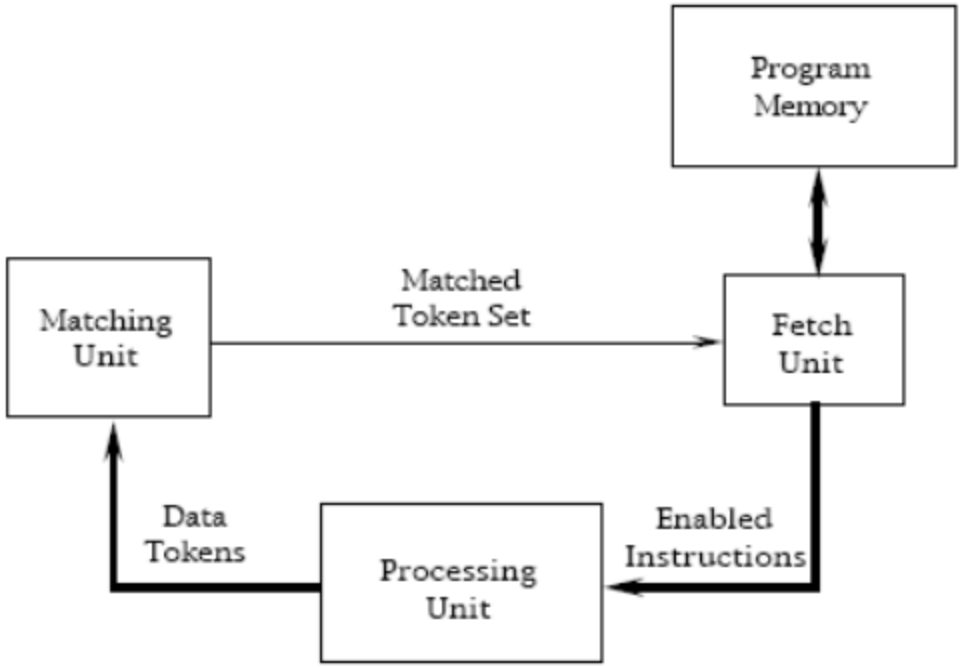
Static Data Flow Machine

- The Update Unit
  - – responsible for detecting the executability of instructions.
  - – sends the address of the enabled instruction to the Fetch Unit.
- The Fetch Unit
  - – fetches and sends a complete operation packet containing the corresponding opcode, data, & destination list to the Processing Unit
  - – clears the presence bits.
- The Processing Unit
  - – performs the operation, forms a result packets, and sends them to the Update Unit.
- The Update Unit
  - – stores each result in the appropriate operand slot and

Data Flow Architecture (8)

Dynamic Data Flow Machine

- Proposed by Arvind at MIT and by Gurd and Watson at the University of Manchester – Tagged-Token Dataflow Architecture



Data Flow Architecture (9)

Dynamic Data Flow Machine

- The Matching Unit
  - a memory containing a pool of waiting tokens
  - Tokens are received by it.
  - brings together tokens with identical tags.
  - If a match exists, the matched token set is passed on to the Fetch Unit.
  - If no match is found, the token is stored in the Matching Unit to await a partner.
- The Fetch Unit
  - the tags of the token pair uniquely identify an instruction to be fetched from the Program Memory.
  - The instruction together with the token pair forms the enabled instruction and

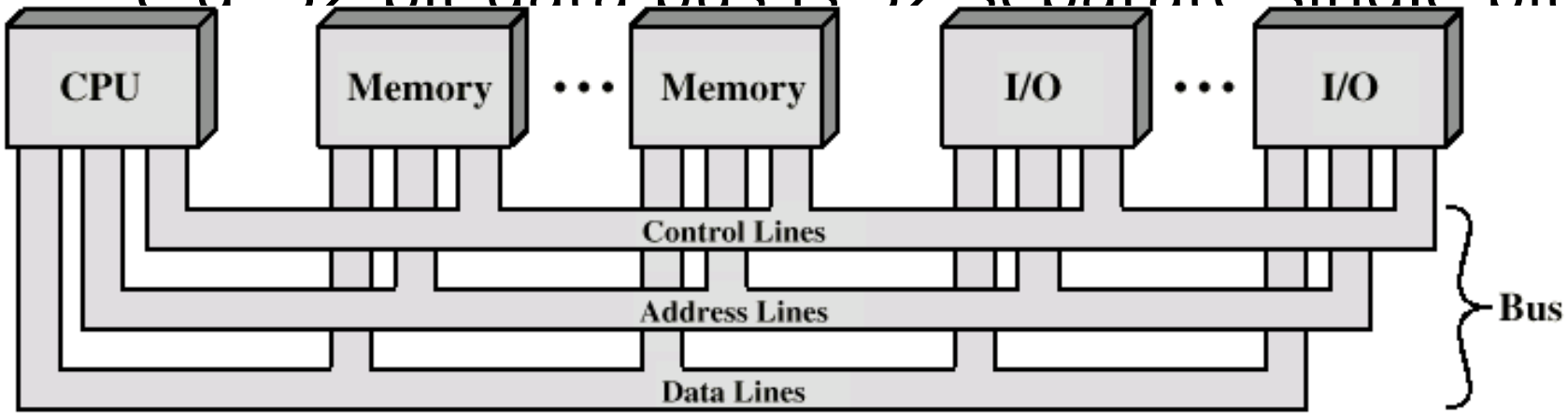
Data Flow Architecture (10)

Drawback with Data Flow Architecture

- Too fine-grained parallelism
  - Incurs more overhead in the execution of an instruction cycle
  - Poor performance in applications with low degree of parallelism.
- Inefficiency in representing and handling data structures
- Need for large token stores to handle all operations waiting for execution
- Difficulty of debugging
- Means to schedule hundreds or thousands of operations that are ready to execute in a limited amount of available

Bus Interconnection

- Bus
  - A communication pathway connecting two or more devices
  - A number of channels in one bus
  - e.g. 32 bit data bus is 32 separate single bit

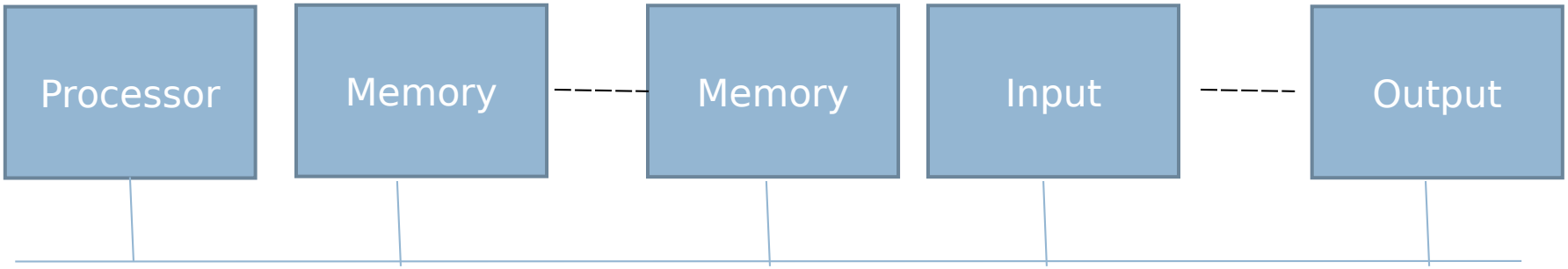


Classification of Buses in detail

- Data Bus
  - Carries data
  - Width of bus is a key determinant of performance
- Address Bus
  - Identify the source or destination of data
  - Bus width determines maximum memory capacity of system e.g. 16 bit address bus giving 64k address space
- Control Bus
  - Control and timing information
    - Memory read/write signal
    - Interrupt request
    - Clock signals

Types of Bus Structure

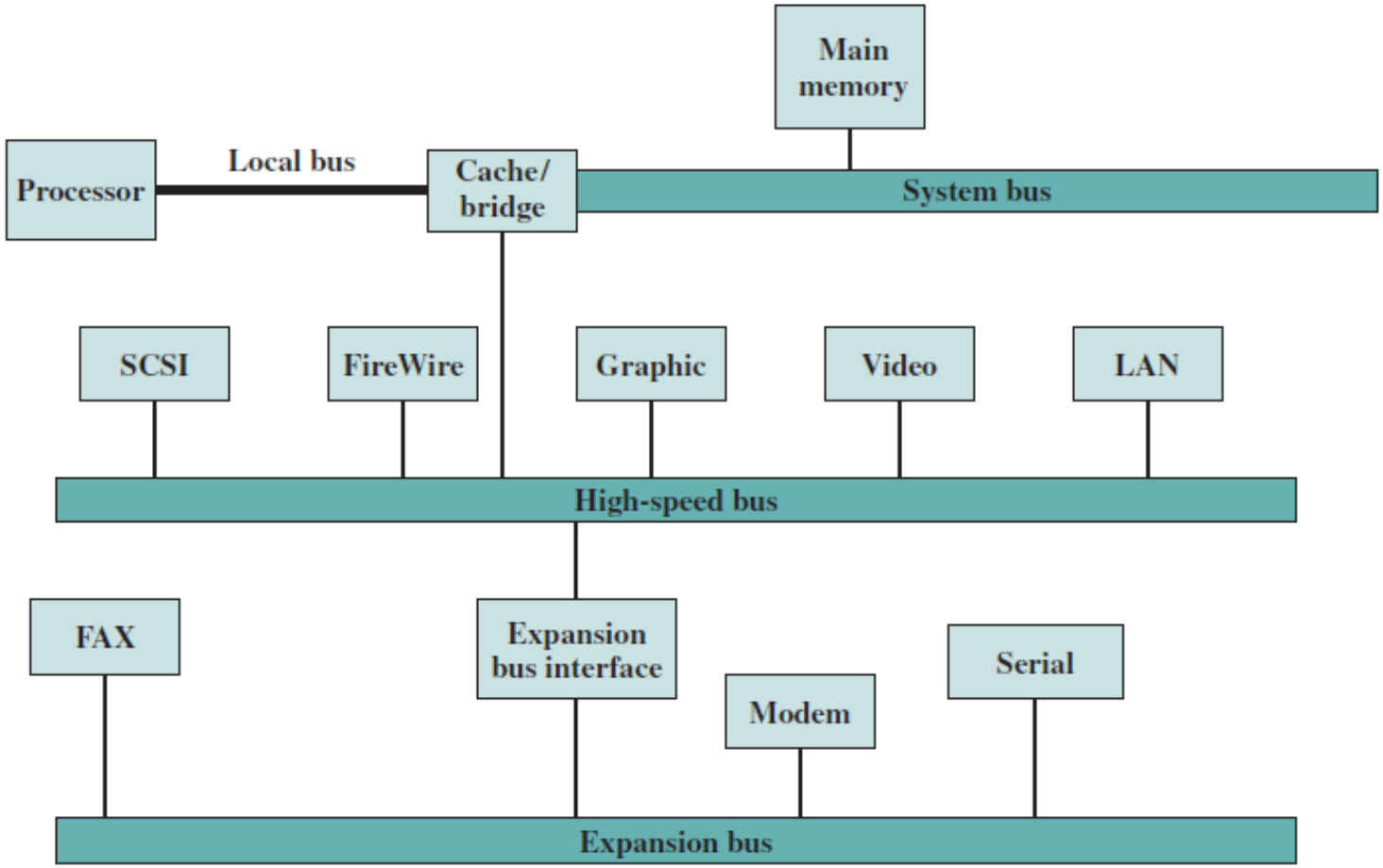
- Single Bus Structure



- Lots of devices on one bus leads to: Propagation delays
- Solution on that to use multiple buses

Types of Bus Structure

- Multi-Bus Structure



Introduction to I/O

- The computer system's input/output (I/O) architecture is its interface to the outside world.
- The key component of a computer system is a set of I/O modules
- Each I/O module interfaces to the system bus and controls one or more peripheral devices.

Introduction to I/O Devices (1)

- Classification of I/O Devices
- Man readable
  - Printers, graphical terminals, screen, keyboard, mouse, pointing devices,…
- Machine readable
  - Disks, tapes, sensors, controllers, actuators
- Communication
  - Modems

Introduction to I/O Devices (2)

- Differences
  - Data transfer rates
    - Range from 0,01 kbytes/sec (keyboard) till 30.000 kbytes/second (graphical screen)
    - Disks typically have 500 kbytes/sec (optical) till 2000 kbytes/sec (magnetic)
  - Application
    - Disk for files or for virtual memory
    - Influences disk scheduling and caching strategy
  - Complexity
    - OS is isolated from most of the complexity by an I/O module (controller)

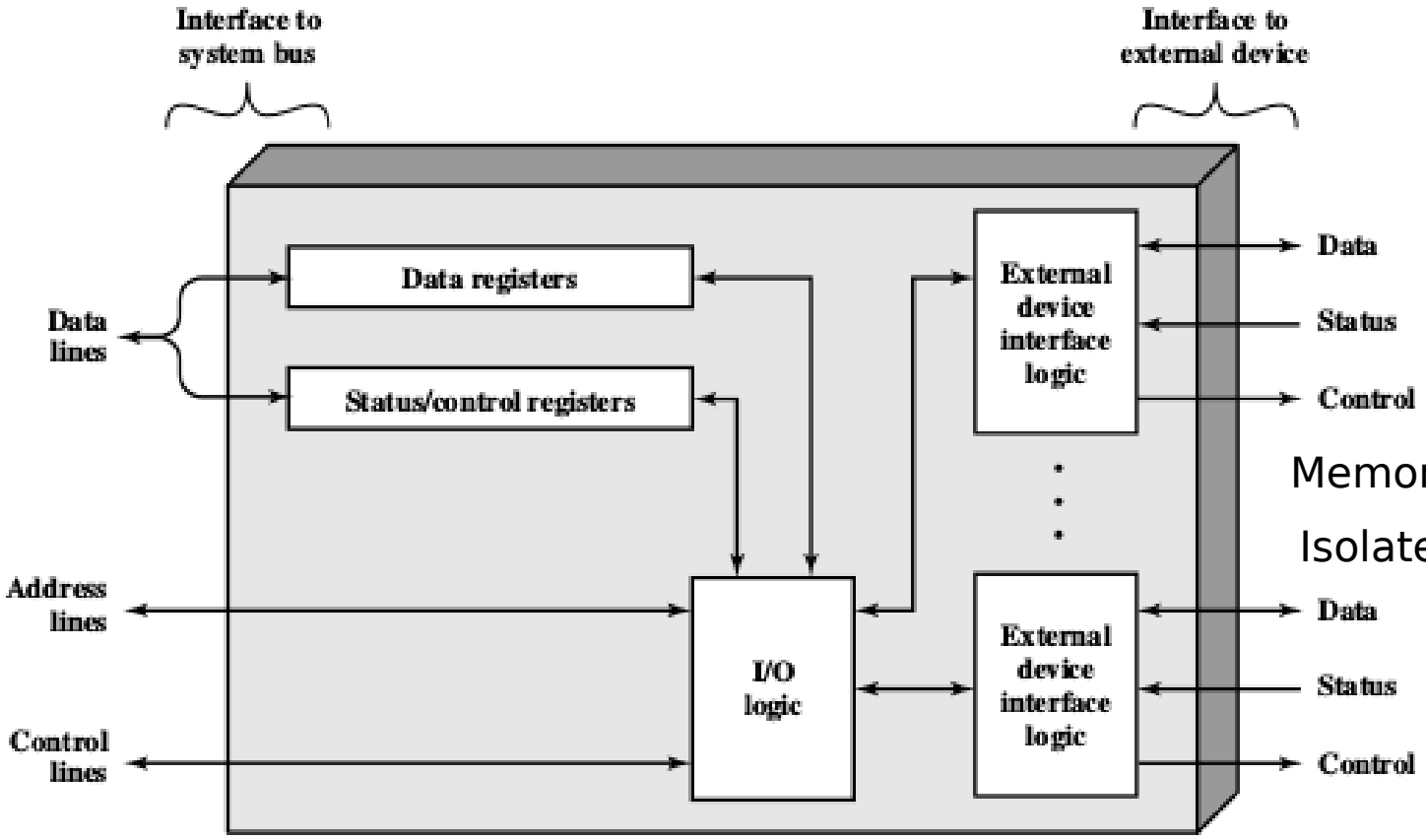
Introduction to I/O Devices (3)

- Differences
  - Unit of transfer
    - Blocks of characters versus streams of characters
  - Representation of data
    - Coding conventions, parity,…
  - Error states
    - Kind of errors, reporting modes,…
- Need a uniform approach to I/O as seen from the user and from the operating system point of view

I/O Module's

- What is I/O Module?
- Why I/O Modules?
  - Variety of Devices, Transfer rate, Data formats
- Functions of I/O Modules
  - Control and timing
  - Processor & Device Communication
    - Command Decoding, Data, Status Reporting, Address Recognition
  - Data Buffering
  - Error Detection

Structure of I/O Module



Memory-mapped I/O

Isolated I/O

Ref: Computer Organization and Architecture 9th Edition by William Stallings

Memory mapped I/O & Isolated I/O

Memory Mapping of I/O device	I/O Mapping of I/O device
1. 16-bit addresses are provided for I/O devices.	1. 8-bit addresses are provided for I/O devices.
2. The devices are accessed by memory read or memory write cycles.	2. The devices are accessed by I/O read or I/O write cycle. During these cycles the 8-bit address is available on both low order address lines and high order address lines.
3. The I/O ports or peripherals can be treated like memory locations and so all instructions related to memory can be used for data transfer between I/O device and the processor.	3. Only IN and OUT instructions can be used for data transfer between I/O device and the processor.
4. In memory mapped ports the data can be moved from any register to ports and vice-versa.	4. In I/O mapped ports the data transfer can take place only between the accumulator and ports.
5. When memory mapping is used for I/O devices, the full memory address space cannot be used for addressing memory. Hence memory mapping is useful only for small systems, where the memory requirement is less.	5. When I/O mapping is used for I/O devices then the full memory address space can be used for addressing memory. Hence it is suitable for systems which requires large memory capacity.
6. In memory mapped I/O devices, a large number of I/O ports can be interfaced.	6. In I/O mapping only 256 ports ($2^8 = 256$) can be interfaced.
7. For accessing the memory mapped devices, the processor executes memory read or write cycle. During this cycle IO/$\overline{M}$ is asserted **low** (IO/$\overline{M} = 0$).	7. For accessing the I/O mapped devices, the processor executes I/O read or write cycle. During this cycle IO/$\overline{M}$ is asserted **high** (IO/$\overline{M} = 1$).

Addressing Modes

- 1. Immediate addressing mode: In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes. Example: MOV AX, 0005H.
- In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.
- 2. Direct addressing mode: In the direct addressing mode, a 16-bit memory address (offset) directly specified in the instruction as a part of it.
- Example: MOV AX, [5000H].

- 3. Register addressing mode: In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.
- Example: MOV BX, AX
- 4. Register indirect addressing mode: Sometimes, the address of the memory l o c a t i o n which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES. Example: MOV AX, [BX].

- 5. Indexed addressing mode: In this addressing mode, offset of the operand is stored one of the index registers. DS & ES are the default segments for index registers SI & DI respectively.
- Example: MOV AX, [SI]

Here, data is available at an offset address stored in SI in DS.

- 6. Register relative addressing mode: In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the register BX, BP, SI & DI in the default (either in DS & ES) segment.
- Example: MOV AX, 50H [BX]

- 7. Based indexed addressing mode: The effective address of data is formed in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.
- Example: MOV AX, [BX][SI]
- 8. Relative based indexed: The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any of the base registers (BX or BP) and any one of the index registers, in a default segment.
- Example: MOV AX, 50H [BX] [SI]

RISC Vs. CISC

Parameter	RISC	CISC
Instruction Types	Simple	Complex
No of Instructions	Reduced(30-40)	Extended(100-200)
Duration of an Instruction	One Cycle	More Cycles (4-120)
Instruction Format	Fixed	Variable
Instruction execution	In Parallel(Pipeline)	Sequential
Addressing Modes	Simple	Complex
Instruction Accessing the memory	Two: Load and Store	Almost all from set
Register set	Multiple	Unique
Complexity	In compiler	In CPU

References

- Computer Architecture and Organization, John P Hays, 3nd Edition, McGraw-Hill Publication, 2001,ISBN 0071004793
- W. Stallings, —Computer Organization and Architecture: Designing for performance‖, Pearson Education/ Prentice Hall of India, 2003, ISBN 978-93-325-1870-4, 7th Edition.