Computer Architecture
UEC509

Dr. Debabrata Ghosh
Assistant Professor, ECED
Thapar University

- *Use this PPT sensibly. It is for reference only. By no means, it is comprehensive and self sufficient for your exams. Use the course syllabus along with the text books for comprehensive knowledge.*

- Textbooks:

Computer Architecture A Quantitative Approach: Hennessy and Patterson (2nd edition)

Computer Organization: Hamacher, Vranesic, Zaky

- Office:

Conference room (FB-05) in Faculty cabin in front of Directorate

Computer Architecture vs Computer Organization

- Set of rules (also called instructions) and methods describing functionality, organization, and implementation of a computer system

i.e. program

Deals with operational units of computer and their interconnection

Computer Types

Based on size, cost, computational power, intended use

1) Personal Computer/ Desktop Computer: Operational units (processing unit, storage unit, video o/p, audio o/p, keyboard) are clearly distinguishable
2) Notebook Computer: Operational units are packed in a compact size
3) Workstation: High resolution graphics i/o capability. Size comparable to desktop computer, but more computational power. Engineering applications: interactive design
4) Enterprise System/ Mainframe: Business data analysis. Computational power and storage capacity higher than workstation
5) Server: Stores database. Capable of handling large no of clients. Requests and responses transported over internet communication
6) Super Computer: Large scale numerical calculation. Weather forecasting, aircraft design, simulation

Historical Perspective

- Computers of today developed over past 65 years
- 1930's: Slow mechanical calculating devices
- 1940's: Electromechanical devices
- First electronic computer using vacuum tubes: University of Pennsylvania
- Development of technology (processor, I/O, memory) divided into 4 generations:
    - First generation (1945-1955)
    - Second generation (1955-1965)
    - Third generation (1965-1975)
    - Fourth generation (1975-present day)

First generation computers

- Idea of storing programs in memory: Von Neumann
- Program and data stored in the same memory
- Assembly language used to prepare program, translated into machine language instruction for program execution
- Vacuum tubes for arithmetic and logic operations
- Magnetic core memory

Second generation computers

- Transistors replaced vacuum tubes
- Magnetic core memory
- High level language developed: easy to prepare program
- Compilers to translate high level language program to assembly and to machine language instructions

Third generation computers

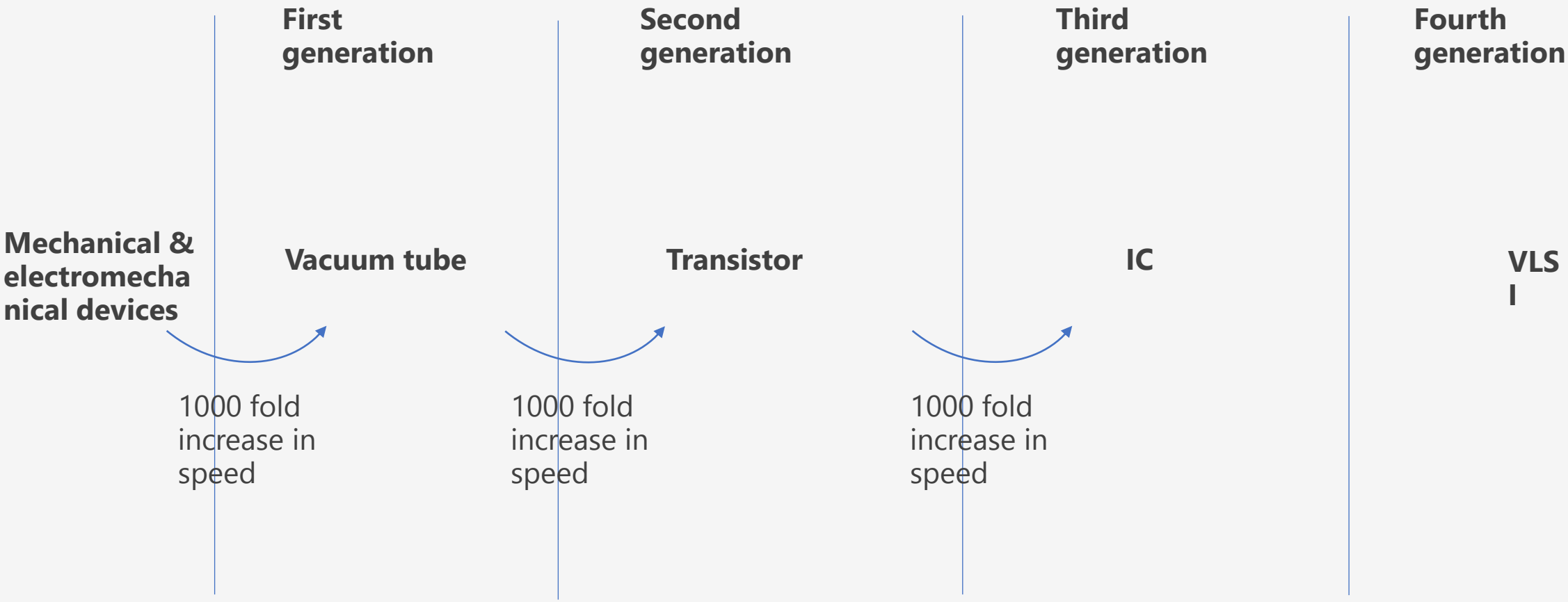
- Invention of IC
- Cheap and fast processor and memory elements
- IC replaced magnetic core memory
- Introduction of parallelism, pipelining
- Introduction of cache memory, virtual memory

Fourth generation computers

- Processor and large portion of main memory on a single chip
- Millions of transistors on a single chip: VLSI
- Processor evolved to microprocessor
- Intel, National Semiconductor, Motorola, TI
- Architectural concepts evolved
- Desktop computer, notebook computer

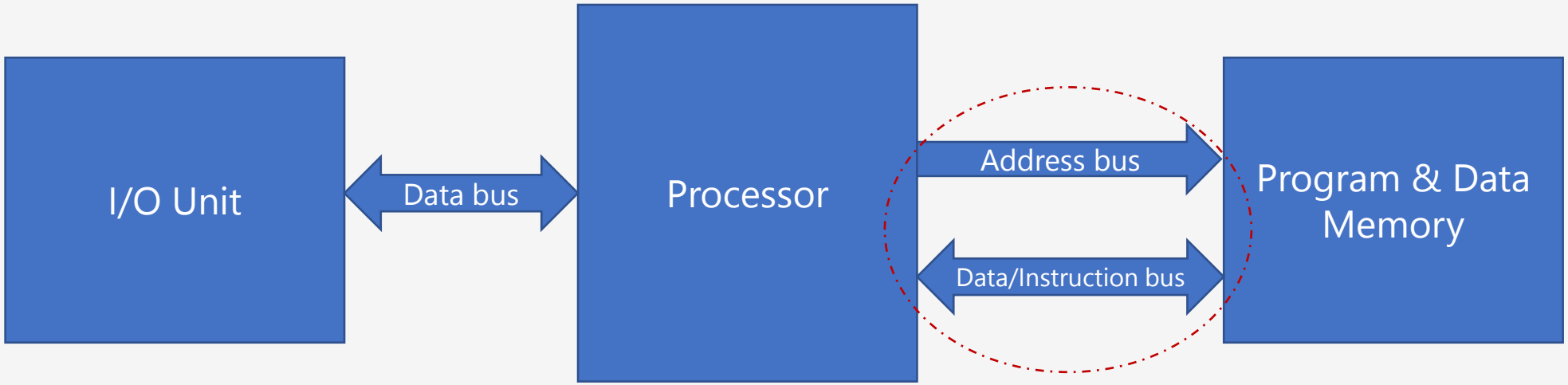
Beyond fourth generation: Massively parallel, extensively distributed systems

Historical Perspective



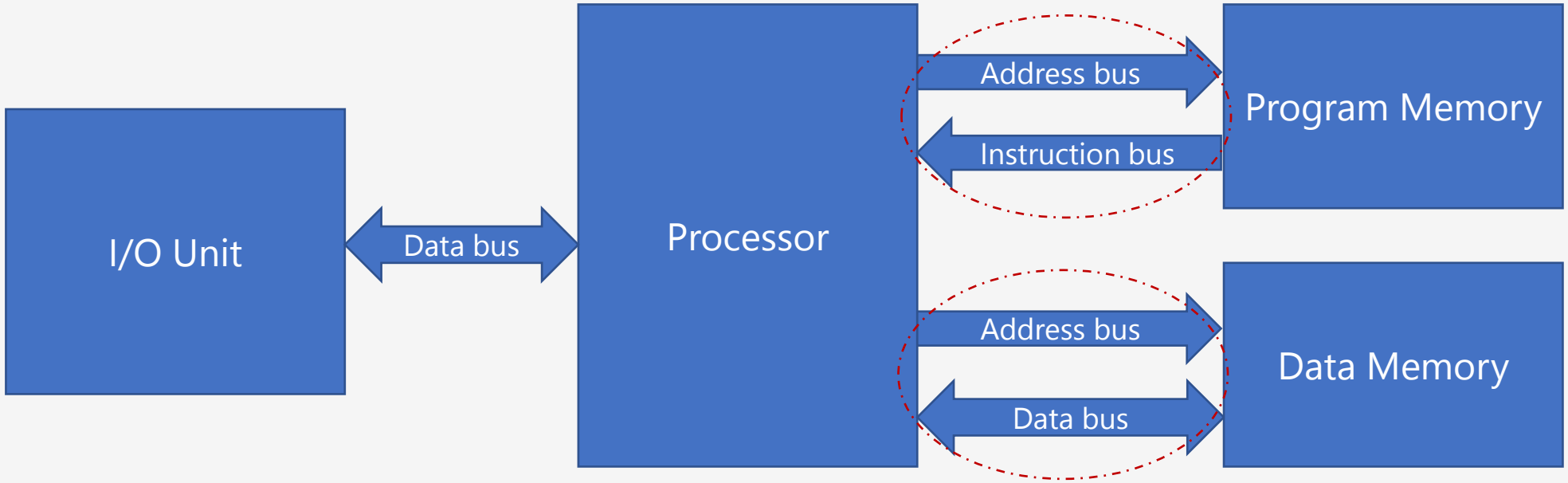
Computer Architecture: Von-Neumann/ Princeton

- Program and data memory same
- Single bus (for address and data)
- Instruction fetch and data transfer can’t be same time



Computer Architecture: Harvard

- Separate program and data memory
- Separate buses for program and data memory
- Instruction fetch and data transfer can be same time
- Free data memory can’t be used for program, complicated CU



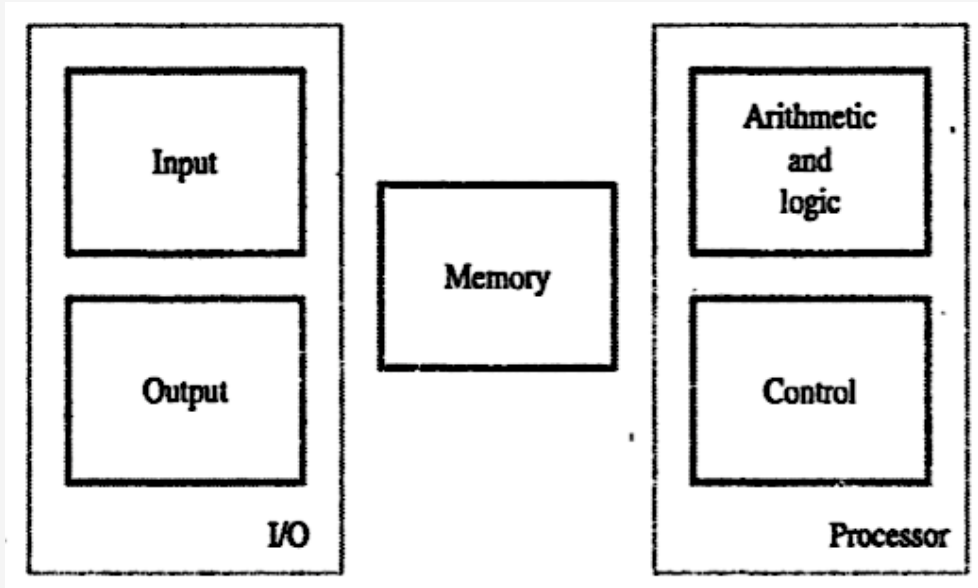
- Modified Harvard: separate cache memory, but same main memory

Functional Units

- Input unit: Accepts coded information and transfers to memory or processor
- Output unit: Sends processed result to the outside world
- Memory unit: Stores program and data
    - Primary memory
        - Operates in electronic speed
        - Program must be stored in PM while being executed
        - Collection of semiconductor storage cells (1bit/cell)
        - Group of cells make a word
        - Word length = no of bits in each word (usually 16-64 bits)
        - Each word has an unique address, i.e. location in memory
        - Time to access a word in memory known as memory access time
        - PM with short and constant memory access time: RAM
        - RAM organized as memory hierarchy: 3-4 levels of RAM (different size and speed)
        - Smallest and fastest RAM: cache memory
        - Largest and slowest RAM: main memory
    - Secondary memory

Functional Units

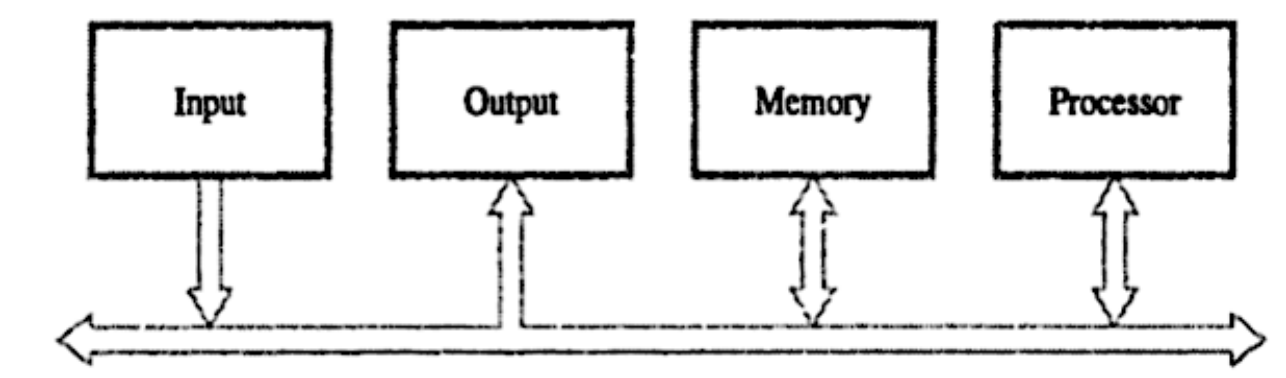
- ALU: Performs arithmetic and logic operations. Operands are stored in registers while executing instruction. Access time to register < access time to cache memory
- CU: Coordinates operations of other units. Control wires carry timing and synchronization signals to other units.



Bus structure

- For an operational system, functional units must be connected in organized way
- For reasonable speed, all units must handle one word (16-64 bits) of data at a given time
- Bits of a word of data are transferred in parallel between units over multiple lines, one bit per line. Group of lines: Bus
- Bus have lines for carrying address and control signals too
- Single bus: all units are connected, only two units can actively use the bus, low cost, flexibility
- Multiple bus: More concurrency, high cost

Bus structure



- Units have variable speed of operation. Electromechanical devices$_{speed}$ < Magnetic/optical disks$_{speed}$ < Memory and processor$_{speed}$
- Efficient transfer mechanism needed: use buffer register with the slow speed unit
- Ex: Transfer of character from processor to character printer

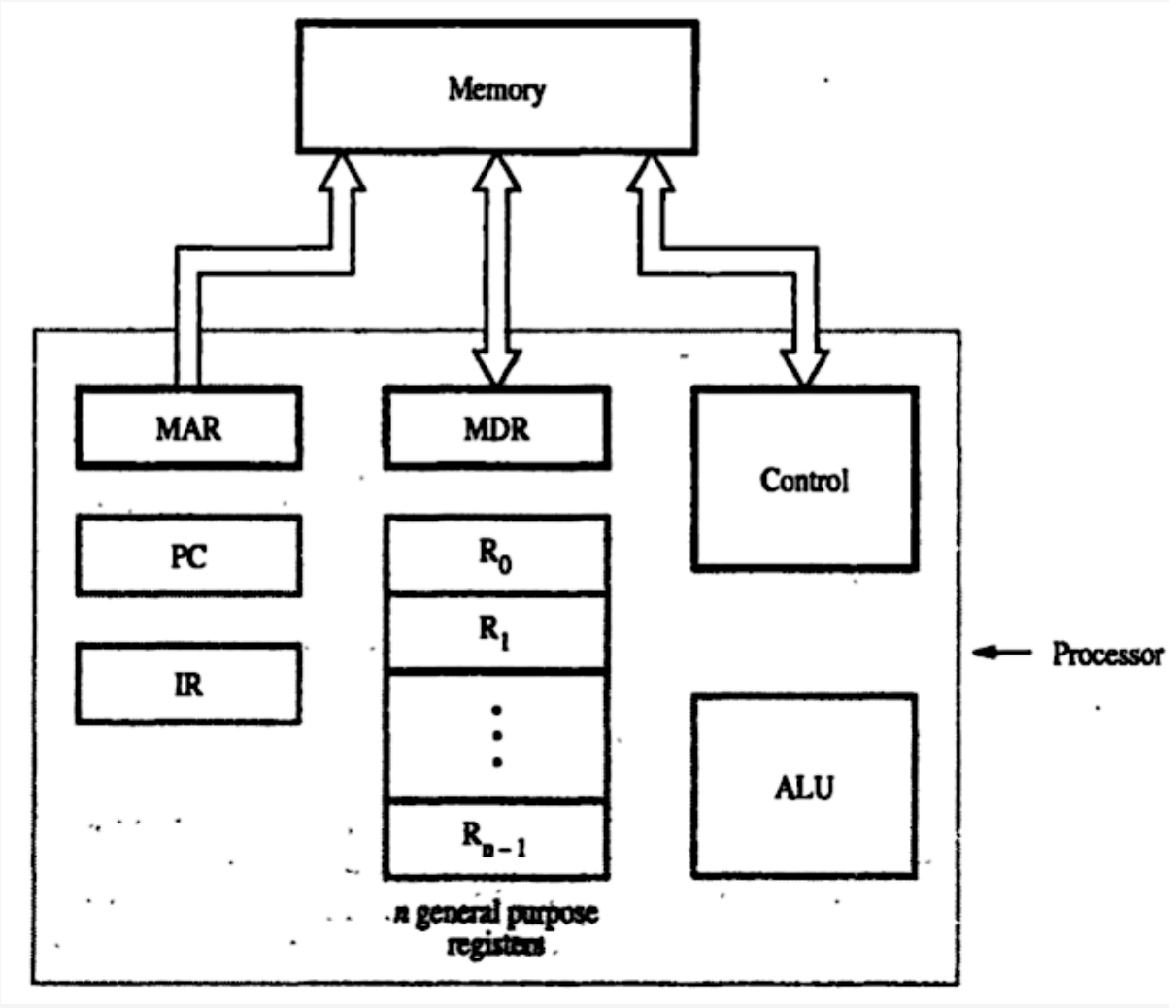
Basic operational concept (Fetch-decode-execution cycle)

- *Program*: list of instructions to perform a specific task
- Program: stored in memory (through i/p unit)
- Data used as operands: stored in memory
- To execute a program:
    1) Fetch the instructions from memory into processor, one-by-one
    2) Decode the instruction: what is the operation, how many operands
    3) Fetch the operands from memory into processor
    4) Execute

- Ex: Add LOCA, R0

Connection between processor and memory

- Other than CU, ALU processor has different registers
- General purpose vs specialized registers
- IR: holds the instruction currently being executed
- PC: holds the address (memory location) of the next instruction to be fetched and executed
- MAR: holds the address to be accessed
- MDR: holds the content at MAR address



Connection between processor and memory

Ex: Execute the instruction 'Add LOCA, R0'

Instruction fetch:
1) PC points to the instruction, i.e. holds the address of the instruction
2) Content of PC transferred to MAR
3) Read control signal sent to memory
4) Content of MAR address (the instruction) transferred to MDR
5) Content of MDR transferred to IR

Decode:
6) Instruction decoded. Addition operation, two operands

Operand fetch:
7) Similar to instruction fetch, operand from LOCA fetched to MDR, then to ALU
8) Operand from R0 fetched to ALU

Execution:
9) ALU performs addition

Write-back:
10) Result goes to MDR from ALU
11) Address of R0 transferred to MAR
12) Write control signal sent to R0
13) Content of MDR transferred to MAR address
14) PC incremented to point to the next instruction

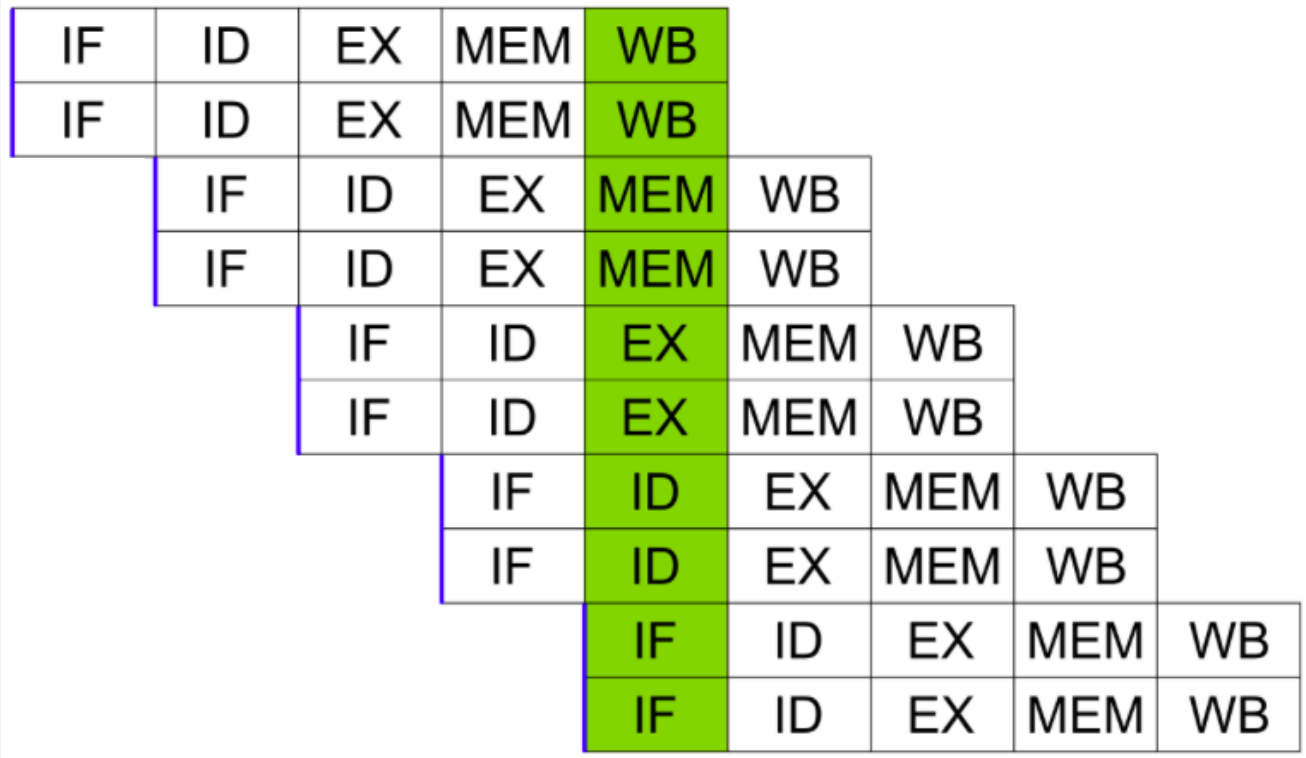
Pipelining and superscalar execution

- Pipelining: Divide instruction in number of basic steps (1 cc) and overlap execution of successive instructions

Inst No	Pipeline stages								
1	IF	ID	EX	MEM	WB				
2		IF	ID	EX	MEM	WB			
3			IF	ID	EX	MEM	WB		
4				IF	ID	EX	MEM	WB	
5					IF	ID	EX	MEM	WB
Clock cycles	1	2	3	4	5	6	7	8	9

- Ideal case: All instructions are maximally overlapped, i.e. one instruction in one clock cycle
- Superscalar execution: Multi-instruction pipelining. Two or more independent instructions in one clock cycle by using multiple execution units

Pipelining and superscalar execution



Processor architecture: CISC vs RISC

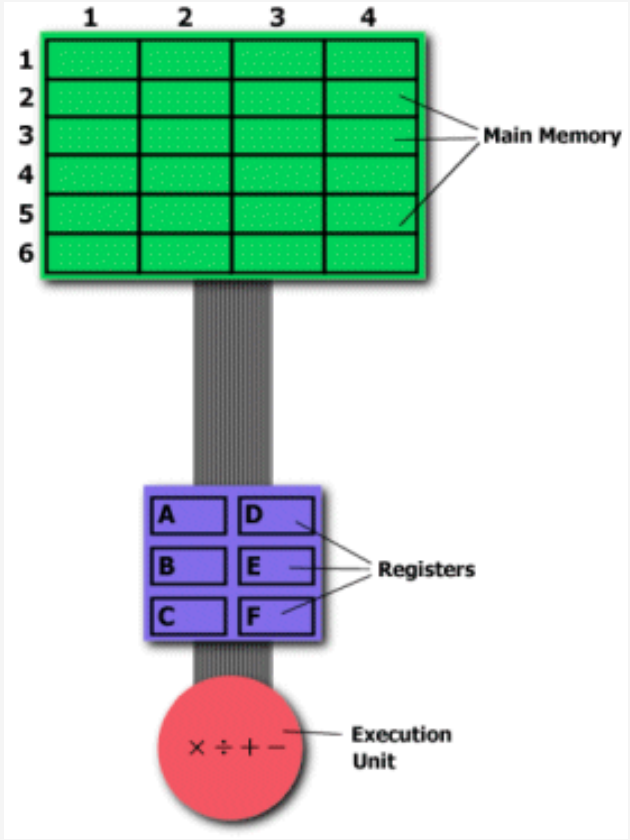
- Simple instruction: less number of basic steps to execute, more number of instructions to perform a given task
- Complex instruction: more number of basic steps, less number of instructions
- Complex instruction combined with pipelining achieves best performance
- Complex instructions execute in variable amount of time, whereas simple instructions execute in uniform amount of time-pipelining much easier using simple instructions
- Processor with complex instruction: CISC
- Processor with simple instruction: RISC

Processor architecture: CISC vs RISC

Multiply two numbers in memory: source code *a = a*b* (where a, b are memory locations, a= 2:3, b=5:2)

MULT 2:3, 5:2

1) CISC instructions operate directly on memory bank
2) Compiler has to do very little work
3) Very little RAM required to store CISC instructions
4) Pipeline harder
5) More emphasis on hardware
6) Microprogrammed control
7) Complex addressing modes
8) Large number of instructions



LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A

1) RISC instructions operate on registers
2) More work for compiler
3) More RAM required to store RISC instructions
4) Pipeline easier
5) More emphasis on software.
6) Hardwired control
7) Simple addressing modes
8) Small number of instructions

Measuring performance

- A is n times faster than B

$$\frac{program\ execution\ time\ in\ B}{program\ execution\ time\ in\ A} = n, \text{ i.e. } \frac{performance\ A}{performance\ B} = n$$

- Execution time defined as either elapsed time or CPU time
- Elapsed time: latency to complete a task, including disk access, memory access, i/o activities, OS overhead etc.
- CPU time: time CPU is computing, not waiting for i/o
- Execution time seen by user is elapsed time, not CPU time
- CPU time spent in program: *user CPU time*
- CPU time spent in OS performing tasks requested by program: *system CPU time*

Benchmarks

- Act of running a program in order to assess the relative performance of a computer
- Single program is seldom used as benchmark, collection of programs used-*benchmark suite*
- SPEC (Standard Performance Evaluation Corporation) benchmark
- **SPECRatio** = $\frac{execution\ time\ on\ \boldsymbol{reference}\ computer}{execution\ time\ on\ \boldsymbol{test}\ computer}$
- A, B: two test computers for relative performance assessment
- Say $SPECRatio_A$ is 1.25 times $SPECRatio_B$

$$1.25 = \frac{SPECRatio_A}{SPECRatio_B} = \frac{\boldsymbol{execution\ time\ on\ B}}{\boldsymbol{execution\ time\ on\ A}} = \frac{Performance\ of\ A}{Performance\ of\ B}$$

- Higher SPECRatio means better performance
- Choice of reference computer irrelevant
- **Overall SPECRatio** = $(\prod_{i=1}^{n} SPECRatio_i)^{\frac{1}{n}}$, n: no of programs in the benchmark suite
- Geometric mean (GM) of the performance ratio = ratio of the GMs

Benchmarks

SPEC 200 Benchmark suite

GM of the performance ratio = 1/1.3 = 0.769

Ratio of GMs = 20.86/27.12 = 0.769

Benchmarks	Ultra 5 Time (sec)	Opteron Time (sec)	SPECRatio	Itanium 2 Time (sec)	SPECRatio	Opteron/Itanium Times (sec)
wupwise	1600	51.5	31.06	56.1	28.53	0.92
swim	3100	125.0	24.73	70.7	43.85	1.77
mgrid	1800	98.0	18.37	65.8	27.36	1.49
applu	2100	94.0	22.34	50.9	41.25	1.85
mesa	1400	64.6	21.69	108.0	12.99	0.60
galgel	2900	86.4	33.57	40.0	72.47	2.16
art	2600	92.4	28.13	21.0	123.67	4.40
equake	1300	72.6	17.92	36.3	35.78	2.00
facerec	1900	73.6	25.80	86.9	21.86	0.85
ammp	2200	136.0	16.14	132.0	16.63	1.03
lucas	2000	88.8	22.52	107.0	18.76	0.83
fma3d	2100	120.0	17.48	131.0	16.09	0.92
sixtrack	1100	123.0	8.95	68.8	15.99	1.79
apsi	2600	150.0	17.36	231.0	11.27	0.65
Geometric mean			20.86		27.12	1.30

Principles for computer design

- Make the common case fast, i.e. favor the frequent case over the infrequent case
- Improve performance: Improve the frequent case(usually simpler), rather than the rare case
- Add two numbers in ALU
    - Frequent case: no overflow
    - Rare case: overflow
    - Improve ALU performance: Optimize the common case
    - ALU slows down when overflow occurs, but that's rare
- Amdahl's law quantifies this principle

Amdahl's Law

- How much faster a program will execute in a machine with enhancement feature as compared to the original machine

$$\text{Speedup} = \frac{Performance\ by\ using\ the\ enhancement\ feature\ for\ the\ entire\ time}{Performance\ without\ using\ the\ enhancement\ feature}$$

$$\text{Speedup} = \frac{Execution\ time\ without\ using\ the\ enhancement\ feature}{Execution\ time\ when\ using\ the\ enhancement\ feature\ for\ the\ entire\ time}$$

- Performance improvement gained by using some enhancement feature is limited by the fraction of time the enhancement feature can be used

$$\text{Speedup}_{\text{overall}} = \frac{Execution\ time_{old}}{Execution\ time_{new}} = \frac{1}{(1-Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Amdahl's Law

Suppose that we are considering an enhancement that runs 10 times faster than the original machine but is only usable 40% of the time. What is the overall speedup gained by incorporating the enhancement?

$$\text{Fraction}_{\text{enhanced}} = 0.4$$

$$\text{Speedup}_{\text{enhanced}} = 10$$

$$\text{Speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

CPU performance equation

- CPU time for a program = CPU clock cycles for the program x clock cycle time

$$= \frac{CPU\ clock\ cycles\ for\ the\ program}{clock\ rate}$$

IC (instruction count): no of instructions to execute a program

CPI: clock cycles per instruction

- CPU time = $\frac{IC \times CPI}{clock\ rate}$

CPU clock cycles for the program = $\sum_{i=1}^{n} CPI_i \times IC_i$

IC_i : no of times instruction *i* is executed in the program

CPI_i : avg no of clock cycles for instruction *i*

- CPU time = $(\sum_{i=1}^{n} CPI_i \times IC_i)$ x clock cycle time
- Overall CPI = $\frac{(\sum_{i=1}^{n} CPI_i \times IC_i)}{Instruction\ count} = \sum_{i=1}^{n} CPI_i \left(\frac{IC_i}{Instruction\ count}\right)$

CPU performance equation

In a certain program 1000 instructions were executed on CPU running at 1 GHz. If the instruction counts and CPI for each class are given below, how long does it take to execute the program?

Instruction Class	Instruction Count	ClassCPI
1	200	2
2	300	3
3	500	1

$$\text{Overall CPI} = \frac{\left(\sum_{i=1}^{n} CPI_i \times IC_i\right)}{Instruction\ count}$$

$$\text{Overall CPI} = \frac{\left(\sum_{i=1}^{n} CPI_i \times IC_i\right)}{Instruction\ count} = 1.8$$

$$\text{Execution time} = \frac{IC \times CPI}{clock\ rate} = 1.8\ \text{ns}$$

CPU performance

- MIPS (Million Instructions Per Second): Rate of execution per unit time

$$\text{MIPS} = \frac{Instruction\ count\ (IC)}{Execution\ time \times 10^6}$$

$$\text{CPU time} = \frac{IC \times CPI}{clock\ rate}$$

Thus,

$$\text{MIPS} = \frac{clock\ rate}{CPI \times 10^6}$$

- Inversely proportional to execution time, proportional to performance: faster the machine, larger the MIPS rating
- Depends on instruction set: cannot compare machines with different instruction sets

CPU performance

For a machine (CPU) with instruction classes:

Instruction class	CPI
A	1
B	2
C	3

For a given high-level language program, two compilers produced the following executed instruction counts:

	Instruction counts (in millions) for each instruction class		
Code from:	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

The machine is assumed to run at a clock rate of 100 MHz.

Which compiler is better? According to MIPS? Is that correct in reality?

$$\text{CPI} = \frac{\left(\sum_{i=1}^{n} CPI_i \times IC_i\right)}{Instruction\ count}$$

MIPS = Clock rate / (CPI x 10^6) = 100 MHz / (CPI x 10^6)

CPU time = Instruction count x CPI / Clock rate

For compiler 1:

- **CPI_1 = (5 x 1 + 1 x 2 + 1 x 3) / (5 + 1 + 1) = 10 / 7 = 1.43**
- **MIPS $Rating_1$ = 100 / (1.428 x 10^6) = 70.0 MIPS**
- **CPU $time_1$ = ((5 + 1 + 1) x 10^6 x 1.43) / (100 x 10^6) = 0.10 seconds**

For compiler 2:

- **CPI_2 = (10 x 1 + 1 x 2 + 1 x 3) / (10 + 1 + 1) = 15 / 12 = 1.25**
- **MIPS $Rating_2$ = 100 / (1.25 x 10^6) = 80.0 MIPS**
- **CPU $time_2$ = ((10 + 1 + 1) x 10^6 x 1.25) / (100 x 10^6) = 0.15 seconds**

MIPS rating indicates that compiler 2 is better while in reality the code produced by compiler 1 is faster

A higher MIPS rating in many cases may not mean higher performance or better execution time. i.e. due to compiler design variations.

Define and quantify dependability

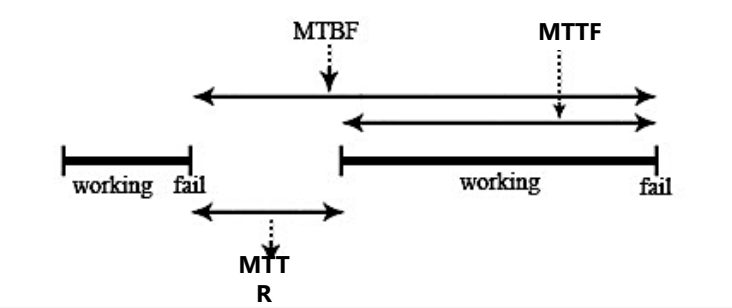
- System operating properly: dependable
- Service level agreement (SLA)
- System alternates between 2 states w.r.t. SLA

  1) Service accomplishment: service delivered as guaranteed in SLA

  2) Service interruption: service delivered different from SLA

- Measure the dependability
  - **Module reliability:** Measure of service accomplishment or service interruption

    Metrics used:



MTTF (mean time to failure): Avg time of continuous service accomplishment

FIT (failure in time): Rate of failures, i.e. 1/MTTF, reported as number of failures per billion hours of operation

MTTR (mean time to repair): Avg time to go from interrupt to accomplishment

MTBF (mean time between failure): (MTTF + MTTR)

  - **Module availability:** Measure of probability that system will be available any time

$$\frac{MTTF}{MTTF+MTTR}$$

Define and quantify dependability

Assume a disk subsystem with the following components and MTTF:

- 10 disks, each rated at 1,000,000-hour MTTF
- 1 SCSI controller, 500,000-hour MTTF
- 1 power supply, 200,000-hour MTTF
- 1 fan, 200,000-hour MTTF
- 1 SCSI cable, 1,000,000-hour MTTF

Using the simplifying assumptions that the lifetimes are exponentially distributed and that failures are independent, compute the MTTF of the system as a whole.

Define and quantify dependability

- Since lifetime of the components are exponentially distributed, age won't affect the probability of failure
- Overall failure rate = sum of the failure rates

$$\text{Failure rate}_{\text{system}} = 10 \times \frac{1}{1{,}000{,}000} + \frac{1}{500{,}000} + \frac{1}{200{,}000} + \frac{1}{200{,}000} + \frac{1}{1{,}000{,}000}$$

$$= \frac{10 + 2 + 5 + 5 + 1}{1{,}000{,}000 \text{ hours}} = \frac{23}{1{,}000{,}000} = \frac{23{,}000}{1{,}000{,}000{,}000 \text{ hours}}$$

or 23,000 FIT. The MTTF for the system is just the inverse of the failure rate:

$$\text{MTTF}_{\text{system}} = \frac{1}{\text{Failure rate}_{\text{system}}} = \frac{1{,}000{,}000{,}000 \text{ hours}}{23{,}000} = 43{,}500 \text{ hours}$$

or just under 5 years.

Instruction set architecture (ISA)

- Boundary between software and hardware
- Visible to programmer or compiler writer
- ISA based on complexity of instructions: CISC and RISC
- ISA based on where operands are kept other than memory:
- Stack architecture: Operands implicitly on top of stack
- Accumulator architecture: One operand implicitly in accumulator
- General purpose register architecture: Only explicit operands- either in registers or in memory locations
    - Register-memory: Memory can be accessed as part of any instruction
    - Register-register: Memory can only be accessed with load and store instructions. Load-store architecture
    - Memory-memory: Operands are in memory only

Instruction set architecture (ISA)

Write the assembly code for C = A + B, where A, B, C are memory locations

Stack architecture:

Push A

Push B

Add *(operands implicitly on top of stack)*

Pop C

Accumulator architecture:

Load A

Add B (one operand implicitly in acc)

Store C

GPR architecture: Register-memory

Load R1, A

Add R1, B

Store C, R1

GPR architecture: Register-register

Load R1, A

Load R2, B

Add R3, R1, R2

Store C, R3