Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

Department of Computer Science and Engineering

COURSE CODE – TITLE: 1151CS110 – COMPUTER ORGANIZATION AND ARCHITECTURE

Course Instructor
Mr. M. Rajeev Kumar
Associate Professor (CSE)

COURSE CODE	COURSE TITLE	L	T	P	C
1151CS110	COMPUTER ORGANIZATION AND ARCHITECTURE	3	0	0	3

Course Category: Program Core

A. Preamble:

This course provides the basics of organizational and architectural issues of a digital computer, analyze performance issues in processor and memory design of a digital computer. It also analyses various data transfer techniques in digital and performance improvement using instruction level parallelism.

B. Prerequisite Course(s):

1151CS104 – Digital Electronics

C. Related Course(s):

1151CS108 - Operating Systems

D. Course Outcomes:

Upon the successful completion of the course, students will be able to:

CO Nos.	Course Outcomes	Level of learning domain (Based on revised Bloom’s)
CO1	Identify the basic structure and functional units of a digital computer.	K2
CO2	Familiarize with arithmetic algorithms and procedure for implementing them in hardware.	K2
CO3	Design a pipeline for consistent execution of instructions with minimum hazards.	K3
CO4	Understand parallel organization’s as advanced computer architectures and the working principles of multiprocessor.	K2
CO5	Identify performance issues in processor and memory design of a digital computer.	K3

E. Correlation of COs with POs:

COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	H	M	M										M	L	M
CO2	H	M	M	L										M	M
CO3	H	H	H	M	L			L	L			L	L	L	L
CO4	H	M	M		L			L	L			L	M		
CO5	H	M	M					L	L			L	M		

H- Strong; M-Medium; L-Low

F. Course Content:

UNIT I INTRODUCTION 9

Eight ideas – Components of a computer system – Technology – Performance – Power wall – Uniprocessors to multiprocessors; Instructions – operations and operands – representing instructions –Addressing and addressing modes.

UNIT II : ARITHMETIC OPERATIONS 8

ALU - Addition and subtraction – Multiplication – Division – Floating Point operations .

UNIT III : PROCESSOR AND CONTROL UNIT 10

Basic MIPS implementation – Building datapath –Pipelining – Pipelined datapath and control – Handling Data hazards & Control hazards – Exceptions.

UNIT IV: PARALLELISM 9

Instruction-level-parallelism – Parallel processing challenges – Flynn's classification – Hardware multithreading – Multicore processors

UNIT V: MEMORY AND I/O SYSTEMS 9

Memory hierarchy - Memory technologies – Cache basics –Cache Memory Mapping Techniques– Measuring and improving cache performance - Virtual memory, TLBs ,Page Replacement Techniques- Input/output system- DMA and interrupts, I/O processors.

TOTAL: 45 Periods

G. Learning Resources

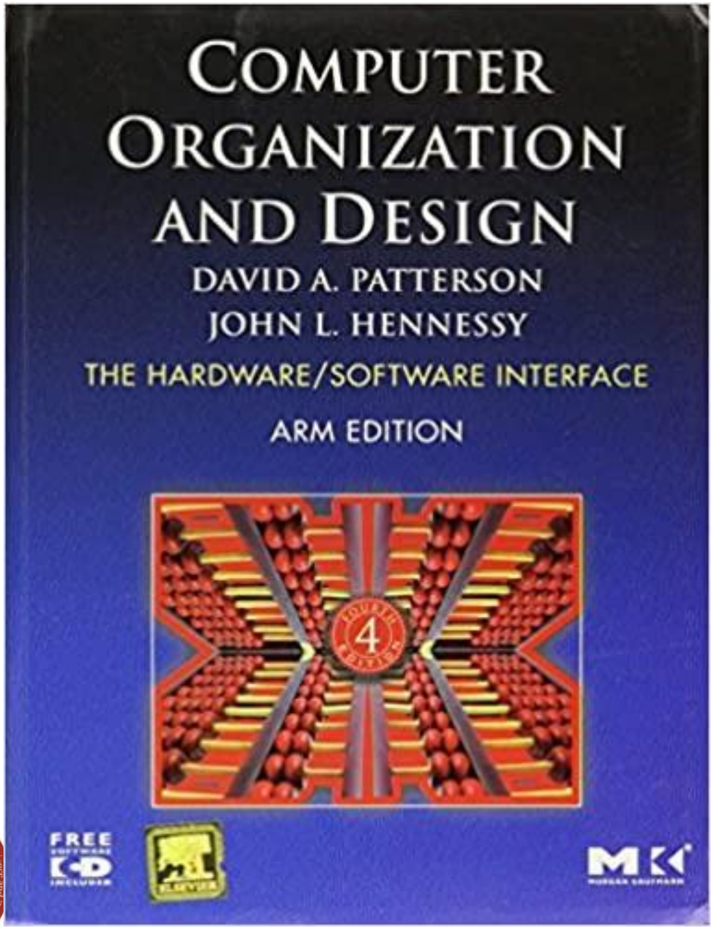
i. Text Books:

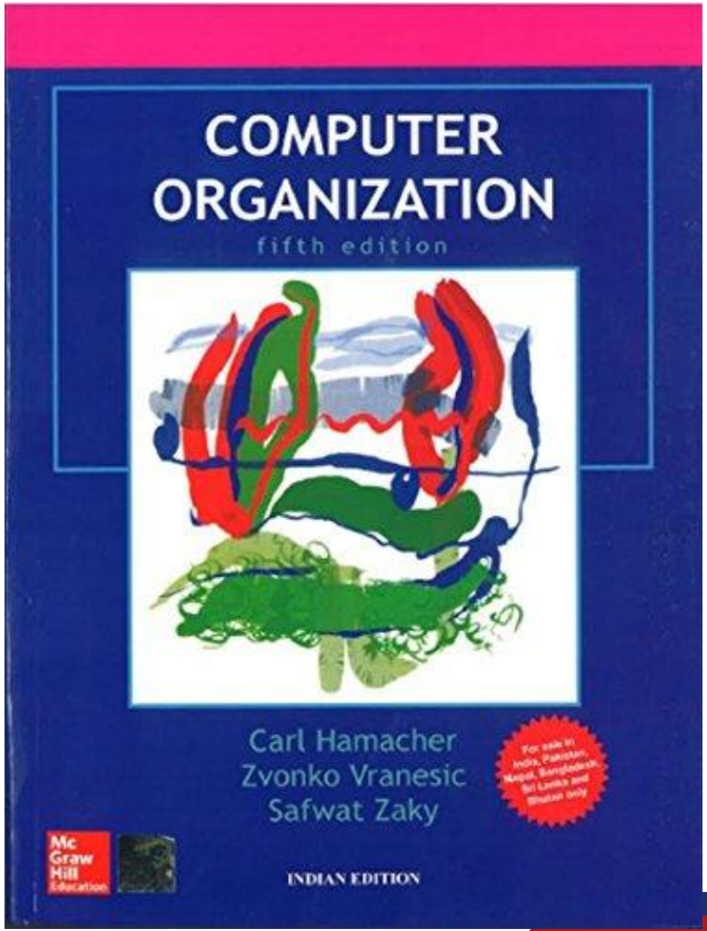
1. David A. Patterson and John L. Hennessy, “Computer Organization and Design: The Hardware/Software interface”, Fourth Edition, Elsevier, 2011.
2. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, Tata McGraw Hill, 2002.

ii. Reference Books:

1. M. Morris Mano, “Computer System Architecture”- Third Edition, Pearson Education, 2007.
2. BehroozParhami, “Computer Architecture”, Oxford University Press, 2007.
3. V.P. Heuring, H.F. Jordan, “Computer Systems Design and Architecture”, Second Edition, Pearson Education, 2004.
4. William Stallings, “Computer Organization and Architecture – Designing for Performance”, Sixth Edition, Pearson Education, 2003.
5. John P. Hayes, “Computer Architecture and Organization”, Third Edition, Tata McGraw Hill, 1998.

Text Books


COMPUTER ORGANIZATION AND DESIGN
DAVID A. PATTERSON
JOHN L. HENNESSY
THE HARDWARE/SOFTWARE INTERFACE
ARM EDITION
FOURTH 4 EDITION
FREE SOFTWARE CD INCLUDED
ELSEVIER
MK MORGAN KAUFMANN


COMPUTER ORGANIZATION
fifth edition
Carl Hamacher
Zvonko Vranesic
Safwat Zaky
For sale in India, Pakistan, Nepal, Bangladesh, Sri Lanka and Bhutan only
Mc Graw Hill Education
INDIAN EDITION

UNIT I

CO Nos.	Course Outcome(s)	Level of learning domain (Based on revised Bloom’s)
CO1	Identify the basic structure and functional units of a digital computer.	K2

EIGHT IDEAS

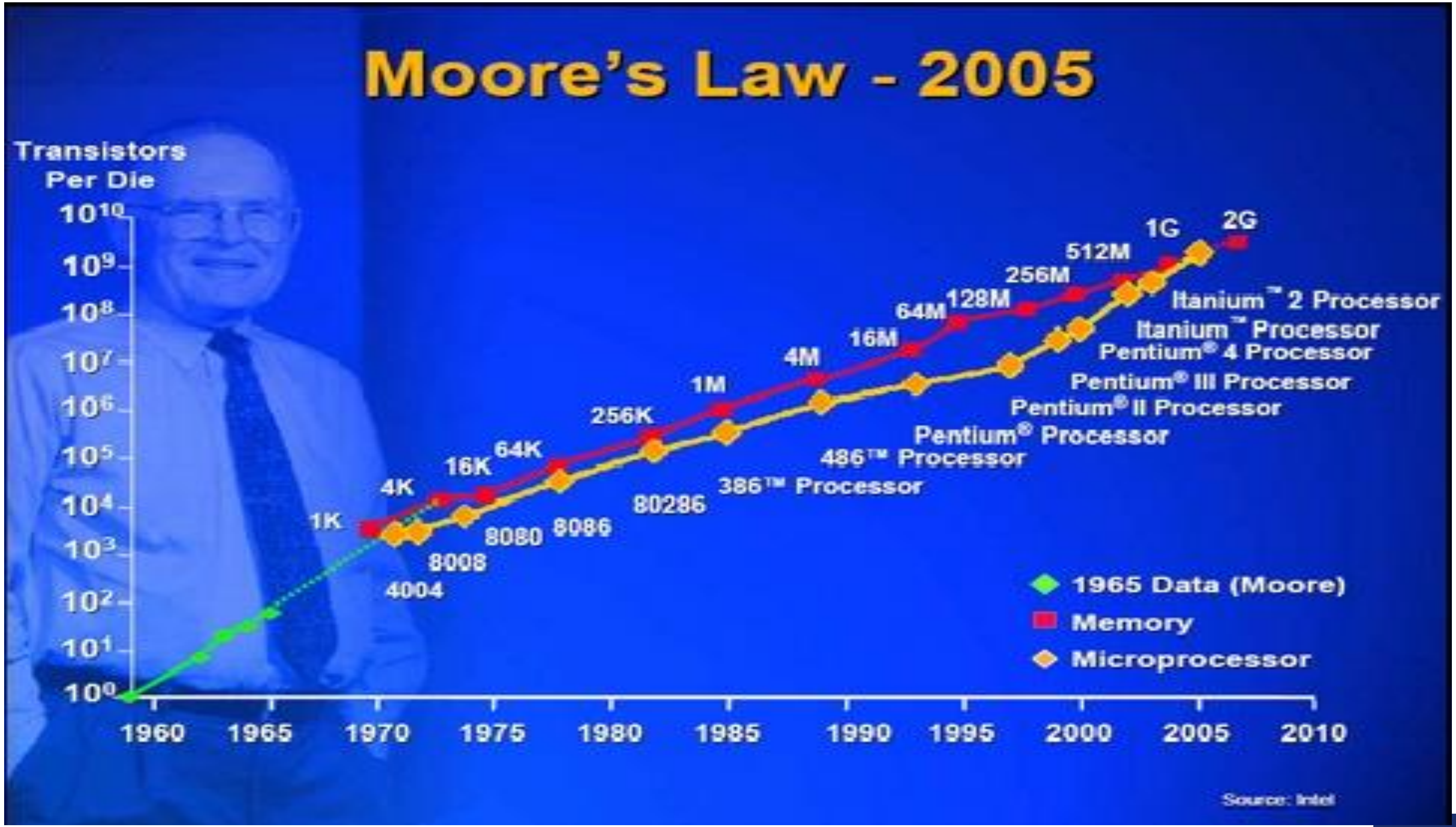
Eight Great Ideas

1. Design for Moore’s Law
2. Use Abstraction to Simplify Design
3. Make the common case fast
4. Performance via parallelism
5. Performance via pipelining
6. Performance via prediction
7. Hierarchy of memories
8. Dependability via redundancy

Idea1: Design for Moore's Law

- Gordon Moore, one of the founders of Intel.
- The integrated circuit resources (i.e no. of Transistors) double approximately every eighteen to twenty four months.

Idea1: Design for Moore's Law (Contd.,)



Idea2: Use Abstraction to Simplify Design

Idea2: Use Abstraction to Simplify Design

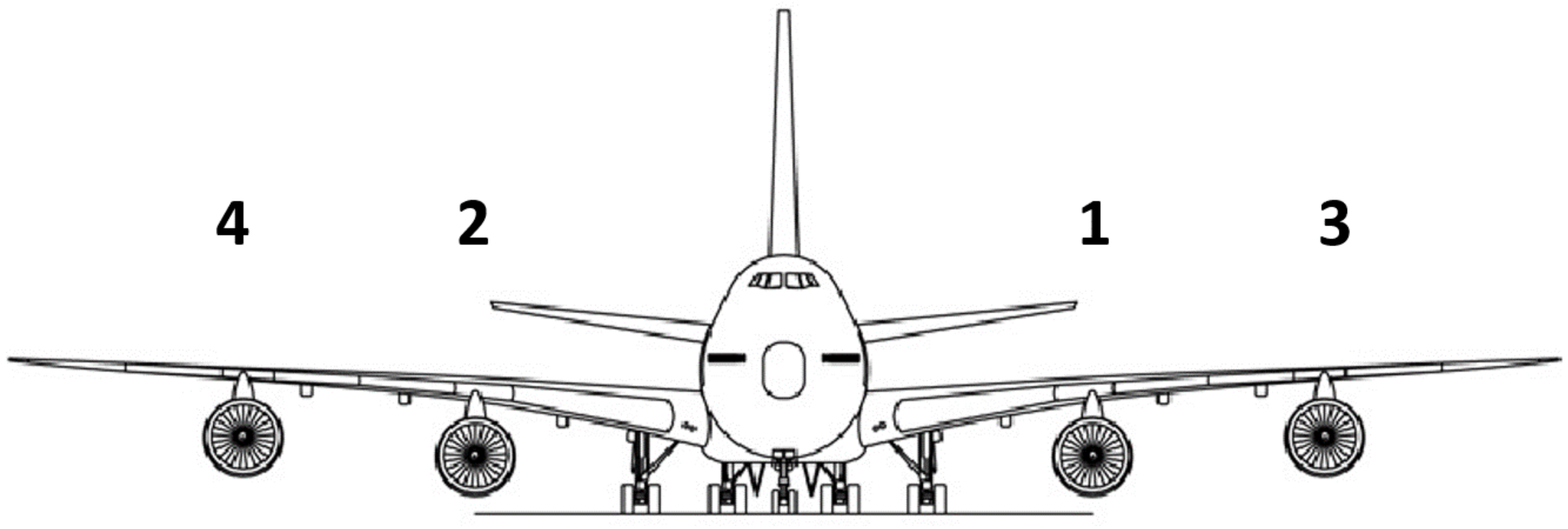
- Techniques are to be invented for more productivity else design time would be more length (Due to Moore’s Law).
- A major productivity technique for hardware and software is to use abstractions to represent the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels.
- abstract painting icon has been to represent this second great idea

Idea3: Make the common case fast

Idea3: Make the common case fast

- Making the common case fast will tend to enhance performance better than optimizing the rare case.
- Ironically, the common case is often simpler than the rare case and hence is often easier to enhance.
- This common sense advice implies that you know what the common case is, which is only possible with careful experimentation and measurement.
- A sports car has been used as the icon for making the common case fast, as the most common trip has one or two passengers, and it's surely easier to make a fast sports car than a fast minivan

Idea4: Performance via parallelism



Idea4: Performance via parallelism

- Since the dawn of computing, computer architects have offered designs that get more performance by performing operations in parallel
- Multiple jet engines of a plane has been used as icon for parallel performance

Idea5: Performance via pipelining

Idea5: Performance via pipelining

Idea6: Performance via prediction

Idea6: Performance via prediction

- It can be better to ask for forgiveness than to ask for permission
- In some cases it can be faster on average to guess and start working rather than wait until you know for sure
  - ✓ The mechanism to recover from a misprediction is not too expensive and your prediction is relatively accurate

Idea7: Hierarchy of memories

Idea8: Dependability via redundancy

Components of a computer system

Components of a computer system

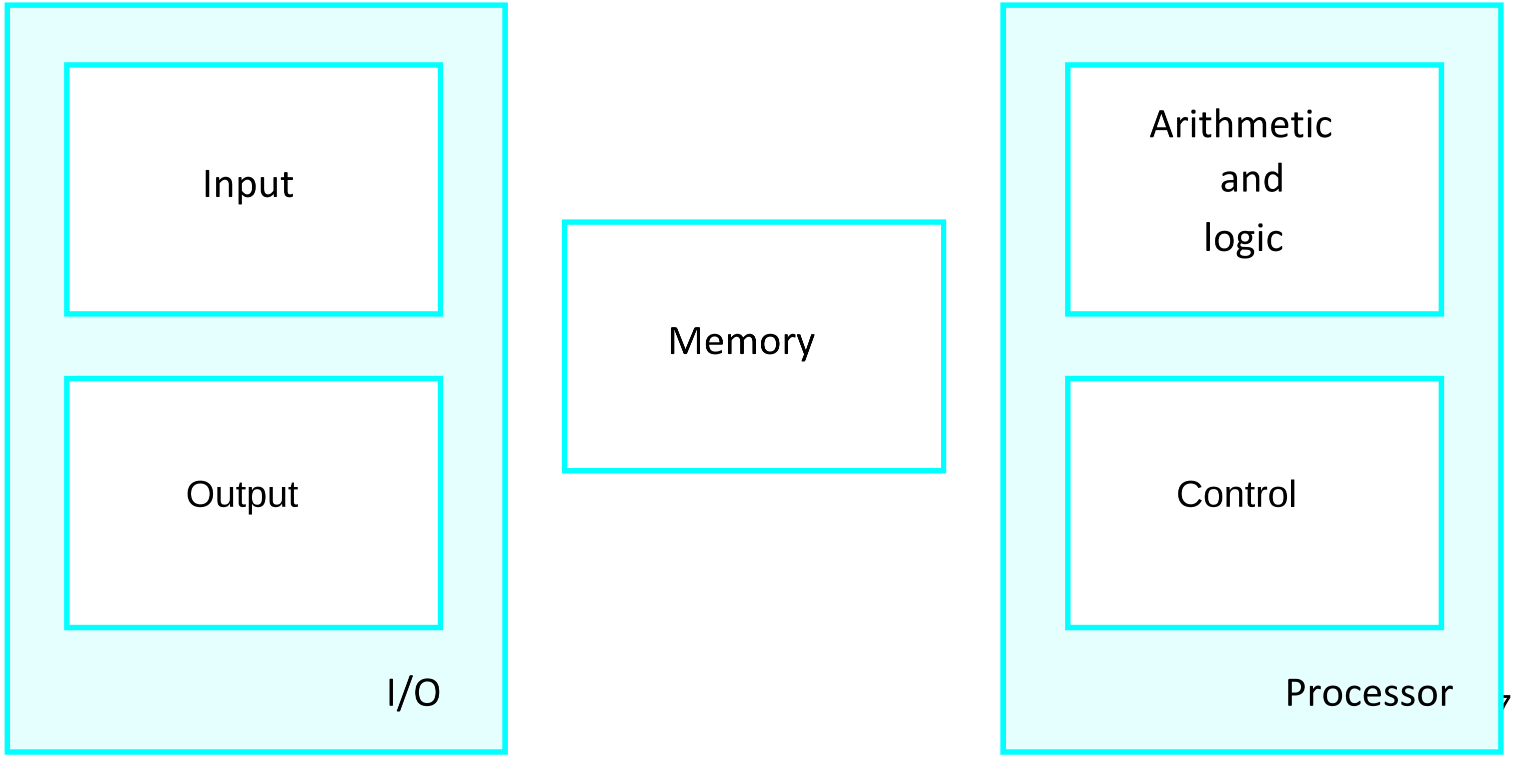


Figure. Basic functional units of a computer

Information Handled by a Computer

Instructions/machine instructions

- Govern the transfer of information within a computer as well as between the computer and its I/O devices
- Specify the arithmetic and logic operations to be performed
- Program

Data

- Used as operands by the instructions
- Source program

Encoded in binary code – 0 and 1

Memory Unit

Store programs and data

Two classes of storage

- Primary storage
  - Fast
  - Programs must be stored in memory while they are being executed
  - Large number of semiconductor storage cells
  - Processed in words
  - Address
  - RAM and memory access time
  - Memory hierarchy – cache, main memory
- Secondary storage – larger and cheaper

Arithmetic and Logic Unit (ALU)

- Most computer operations are executed in ALU of the processor.
- Load the operands into memory – bring them to the processor – perform operation in ALU – store the result back to memory or retain in the processor.
- Registers
- Fast control of ALU

Control Unit

- All computer operations are controlled by the control unit.
- The timing signals that govern the I/O transfers are also generated by the control unit.
- Control unit is usually distributed throughout the machine instead of standing alone.
- Operations of a computer:
    - ➢ Accept information in the form of programs and data through an input unit and store it in the memory
    - ➢ Fetch the information stored in the memory, under program control, into an ALU, where the information is processed
    - ➢ Output the processed information through an output unit
    - ➢ Control all activities inside the machine through a control unit

Technologies for Building Processors and Memory

Technology

Year	Technology used in Computer	Relative performance/ unit cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated Circuits	900
1995	Very large-scale integrated circuit	2400000
2005	Ultra large-scale integrated circuit	6200000000

Performance

Performance

- The most important measure of a computer is how quickly it can execute programs.
- Three factors affect performance:
  - ➢Hardware design
  - ➢Instruction set
  - ➢Compiler
- Processor time to execute a program depends on the hardware involved in the execution of individual machine instructions.
  - ✓ Elapsed Time
  - ✓ Throughput

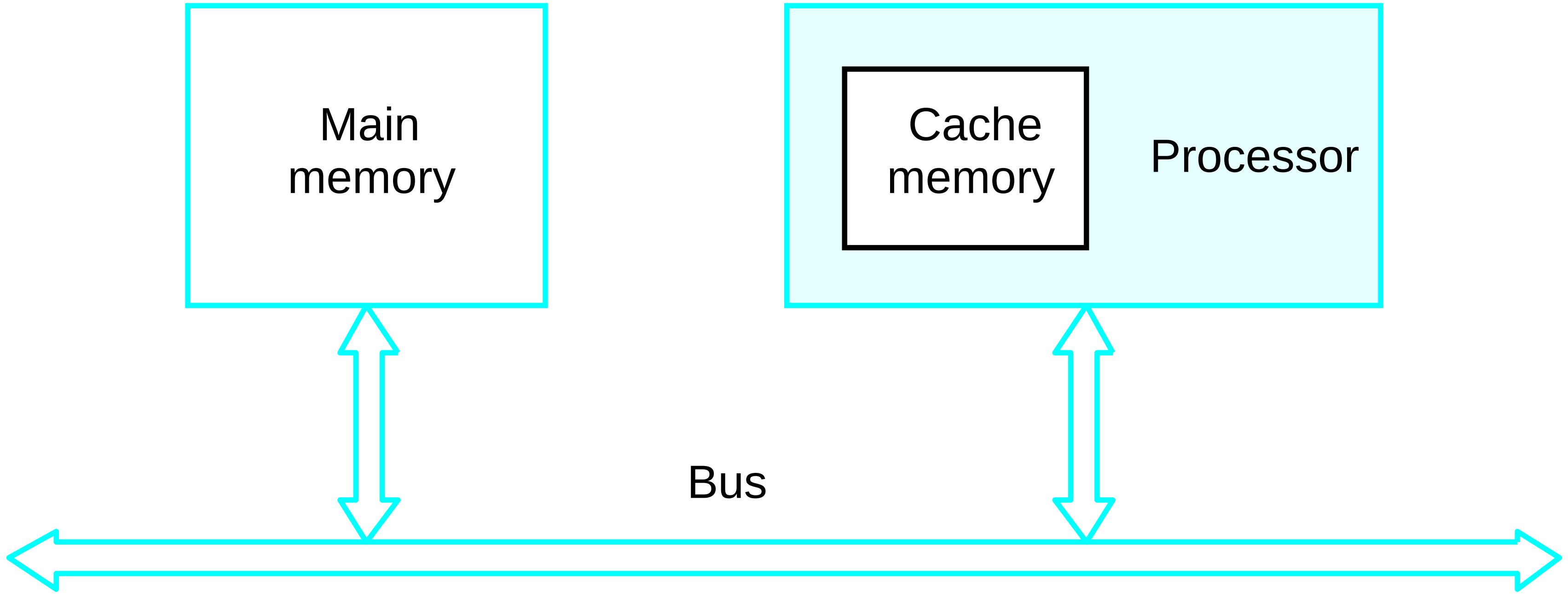


Figure. The processor cache.

Performance

- The processor and a relatively small cache memory can be fabricated on a single integrated circuit chip.
- Speed
- Cost
- Memory management

1. Processor Clock

- Clock, clock cycle, and clock rate
- Length (P)
- clock rate (R = 1/P) which is measured in cycles per second
- Basic steps (S): The execution of each instruction is divided into several steps, each of which completes in one clock cycle.
- cycles per second - Hertz (Hz)
- million cycles per second – Mega Hertz (MHz)
- billion cycles per second – Giga Hertz (GHz)

2. Basic Performance Equation

$$T = \frac{N \times S}{R}$$

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate
- Note: these are not independent to each other

How to improve T?

3. Pipeline and Superscalar Operation

- Instructions are not necessarily executed one after another.
- The value of S doesn’t have to be the number of clock cycles to execute one instruction.
- Pipelining – overlapping the execution of successive instructions.
- Add R1, R2, R3
- Superscalar operation – multiple instruction pipelines are implemented in the processor.
- Goal – reduce S (could become <1!)

4. Clock Rate

- Increase clock rate
  - ➢ Improve the integrated-circuit (IC) technology to make the circuits faster
  - ➢ Reduce the amount of processing done in one basic step (however, this may increase the number of basic steps needed)
- Increases in R that are entirely caused by improvements in IC technology affect all aspects of the processor’s operation equally except the time to access the main memory.

5. CISC and RISC

- Tradeoff between N and S
- A key consideration is the use of pipelining
  - S is close to 1 even though the number of basic steps per instruction may be considerably larger
  - It is much easier to implement efficient pipelining in processor with simple instruction sets
- Reduced Instruction Set Computers (RISC)
- Complex Instruction Set Computers (CISC)

6. Compiler

- A compiler translates a high-level language program into a sequence of machine instructions.
- To reduce N, we need a suitable machine instruction set and a compiler that makes good use of it.
- Goal – reduce N×S
- A compiler may not be designed for a specific processor; however, a high-quality compiler is usually designed for, and with, a specific processor.
- Optimized compiler

7. Performance Measurement

- T is difficult to compute.
- Measure computer performance using benchmark programs.
- System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Compile and run (no simulation)
- Reference computer

$$SPEC\,rating = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

$$SPEC\,rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Power Wall

Power Wall

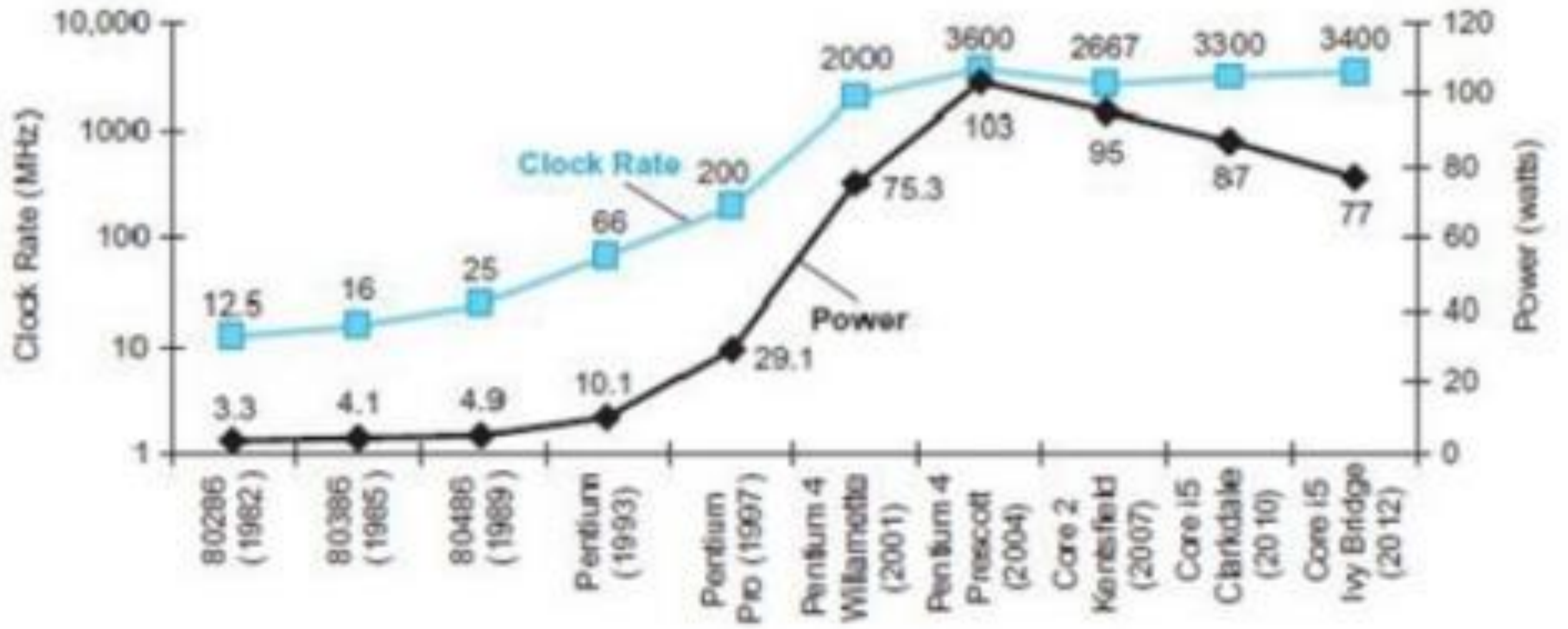


Figure. Clock rate and power wall for intel x86 microprocessors over eight generations and 25 years

Power Wall *(Contd.,)*

Power & Energy in Integrated circuits:

- Power is the biggest challenge facing the computer designer for every class of computer.
  - ✓ First, power must be brought in and distributed around the chip which includes hundreds of pins and multiple interconnection layers just for power and ground.
  - ✓ Second, power is dissipated as heat and must be removed.
- How should a system architect think about performance, power and energy?
- There are three primary concerns.
  - ✓ What is the maximum power a processor ever requires?
  - ✓ What is the sustained Power consumption?
  - ✓ Which metric is the right one for comparing processors: energy or power?

Power Wall *(Contd.,)*

- The dominant technology for integrated circuits is called CMOS (complementary metal oxide semiconductor).
- For CMOS, the primary source of energy consumption is so-called dynamic energy—that is, energy that is consumed when transistors switch states from 0 to 1 and vice versa.
- The dynamic energy depends on the capacitive loading of each transistor and the voltage applied:

$$Power = Capactive\ load\ \times Voltage^2 \times Frequency\ switched$$

- Frequency switched is a function of the clock rate.
- The capacitive load per transistor is a function of both the number of transistors connected to an output (called the fan out) and the technology, which determines the capacitance of both wires and transistors.

Power Wall *(Contd.,)*

- Although dynamic energy is the primary source of energy consumption in CMOS, static energy consumption occurs because of leakage current that flows even when a transistor is off.
- In servers, leakage is typically responsible for 40% of the energy consumption.
- Thus, increasing the number of transistors increases power dissipation, even if the transistors are always off.
- A variety of design techniques and technology innovations are being deployed to control leakage, but it’s hard to lower voltage further.

Power Wall – Example Problem

Suppose we developed a new, simpler processor that has 85% of the capacitive load of the more complex older processor. Further, assume that it has adjustable voltage so that it can reduce voltage 15% compared to processor B, which results in a 15% shrink in frequency. What is the impact on dynamic power?

$$\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = \frac{(\text{Capacitive load} \times 0.85) \times (\text{Voltage} \times 0.85)^2 \times (\text{Frequency switched} \times 0.85)}{\text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}}$$

Multiprocessors and Multicomputers

Multiprocessor computer

- Execute a number of different application tasks in parallel
- Execute subtasks of a single large task in parallel
- All processors have access to all of the memory – shared-memory multiprocessor
- Cost – processors, memory units, complex interconnection networks

Multicomputers

- Each computer only have access to its own memory
- Exchange message via a communication network – message-passing multicomputers

Basic Operational Concepts

Basic Operational Concept – A Review

- Activity in a computer is governed by instructions.
- To perform a task, an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from the memory into the processor, which executes the specified operations.
- Data to be used as operands are also stored in the memory.

A Typical Instruction

- Add LOCA, R0
- Add the operand at memory location LOCA to the operand in a register R0 in the processor.
- Place the sum into register R0.
- The original contents of LOCA are preserved.
- The original contents of R0 is overwritten.
- Instruction is fetched from the memory into the processor – the operand at LOCA is fetched and added to the contents of R0 – the resulting sum is stored in register R0.

Separate Memory Access and ALU Operation

- Load LOCA, R1
- Add R1, R0
- Whose contents will be overwritten?

Connection Between the Processor and the Memory

Registers

- Instruction register (IR)
- Program counter (PC)
- General-purpose register (R_0 – R_{n-1})
- Memory address register (MAR)
- Memory data register (MDR)

Typical Operating Steps

- Programs reside in the memory through input devices
- PC is set to point to the first instruction
- The contents of PC are transferred to MAR
- A Read signal is sent to the memory
- The first instruction is read out and loaded into MDR
- The contents of MDR are transferred to IR
- Decode and execute the instruction
- Get operands for ALU
  - ➢ General-purpose register
  - ➢ Memory (address to MAR – Read – MDR to ALU)
- Perform operation in ALU
- Store the result back
  - ➢ To general-purpose register
  - ➢ To memory (address to MAR, result to MDR – Write)
- During the execution, PC is incremented to the next instruction

Interrupt

- Normal execution of programs may be preempted if some device requires urgent servicing.
- The normal execution of the current program must be interrupted – the device raises an *interrupt* signal.
- Interrupt-service routine
- Current system information backup and restore (PC, general-purpose registers, control information, specific information)

Bus Structures

- There are many ways to connect different parts inside a computer together.
- A group of lines that serves as a connecting path for several devices is called a *bus*.
- Address/data/control

Speed Issue

- Different devices have different transfer/operate speed.
- If the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.
- How to solve this?
- A common approach – use buffers.

Instruction and Instruction Sequencing

“Must-Perform” Operations

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Register Transfer Notation

- Identify a location by a symbolic name standing for its hardware binary address (LOC, R0,…)
- Contents of a location are denoted by placing square brackets around the name of the location (R1←[LOC], R3 ←[R1]+[R2])
- Register Transfer Notation (RTN)

Assembly Language Notation

- Represent machine instructions and programs.
- Move LOC, R1 = R1←[LOC]
- Add R1, R2, R3 = R3 ←[R1]+[R2]

CPU Organization

- Single Accumulator
  - Result usually goes to the Accumulator
  - Accumulator has to be saved to memory quite often
- General Register
  - Registers hold operands thus reduce memory traffic
  - Register bookkeeping
- Stack
  - Operands and result are always in the stack

Instruction Formats

- Three-Address Instructions
  - ADD A, B, C — $C \leftarrow A + B$
- Two-Address Instructions
  - ADD A, B — $B \leftarrow A + B$
- One-Address Instructions
  - ADD M — $AC \leftarrow AC + M[AR]$
- Zero-Address Instructions
  - ADD — $TOS \leftarrow TOS + (TOS - 1)$
- RISC Instructions
  - Lots of registers. Memory is restricted to Load & Store

Example: Evaluate $E = (A+B) * (C+D)$

- Three-Address

1.	ADD	A, B, F	; $M[F] \leftarrow M[A] + M[B]$
2.	ADD	C, D, G	; $M[G] \leftarrow M[C] + M[D]$
3.	MUL	F, G, E	; $M[E] \leftarrow M[F] * M[G]$

- Two-Address

1.	ADD	A, B	; $M[B] \leftarrow M[A] + M[B]$
2.	ADD	C, D	; $M[D] \leftarrow M[C] + M[D]$
3.	MUL	B, D	; $M[D] \leftarrow M[B] * M[D]$
4.	LOAD	D, E	; $M[E] \leftarrow M[D]$

Example: Evaluate $E = (A+B) * (C+D)$

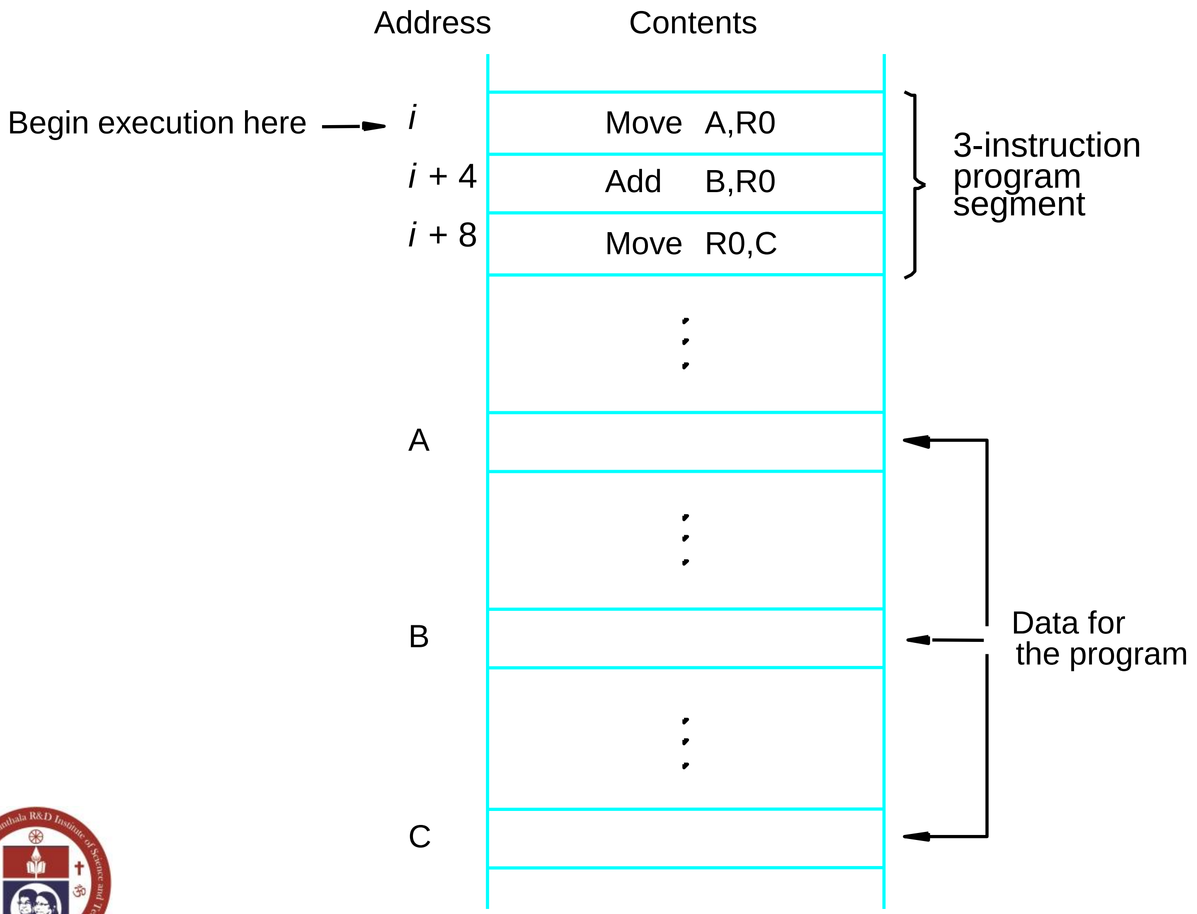
One-Address

1.	LOAD	A	; $AC \leftarrow M[A]$
2.	ADD	B	; $AC \leftarrow AC + M[B]$
3.	STORE	F	; $M[F] \leftarrow AC$
4.	LOAD	C	; $AC \leftarrow M[C]$
5.	ADD	D	; $AC \leftarrow AC + M[D]$
6.	MUL	F	; $AC \leftarrow AC * M[F]$
7.	STORE	E	; $M[E] \leftarrow AC$

Using Registers

- Registers are faster
- Shorter instructions
- The number of registers is smaller (e.g. 32 registers need 5 bits)
- Potential speedup
- Minimize the frequency with which data is moved back and forth between the memory and processor registers.

Instruction Execution and Straight-Line Sequencing



Assumptions:

- One memory operand per instruction
- 32-bit word length
- Memory is byte addressable
- Full memory address can be directly specified in a single-word instruction

Two-phase procedure

-Instruction fetch

-Instruction execute

Branching

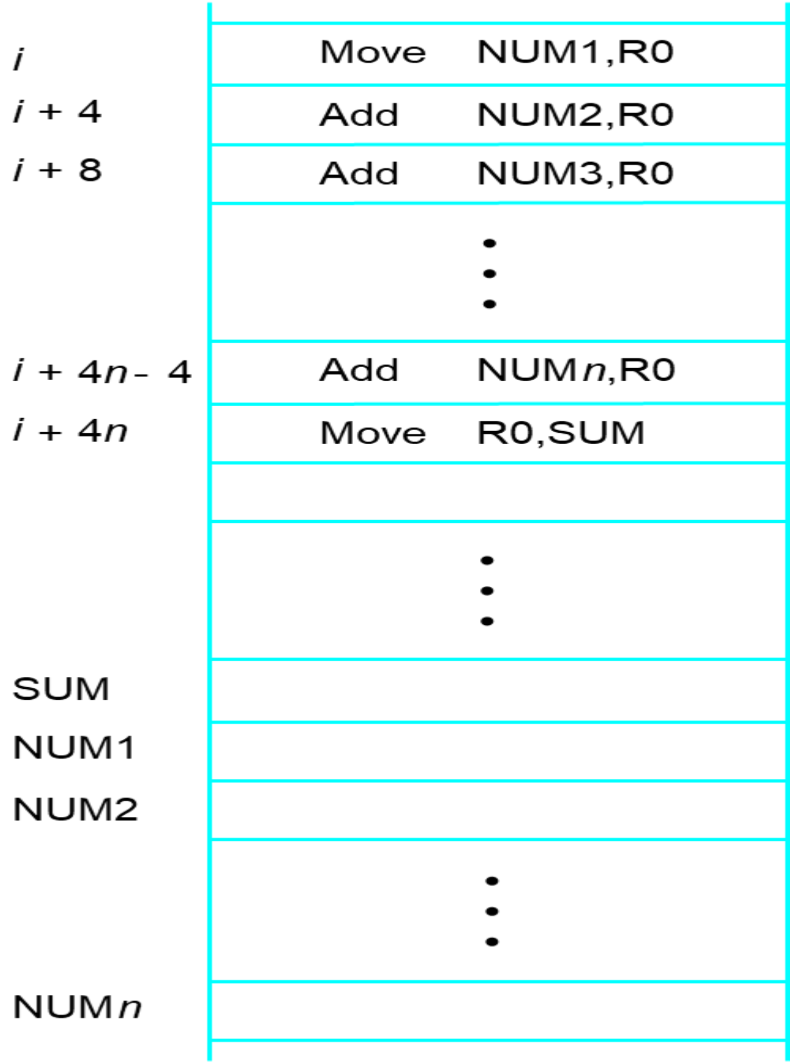
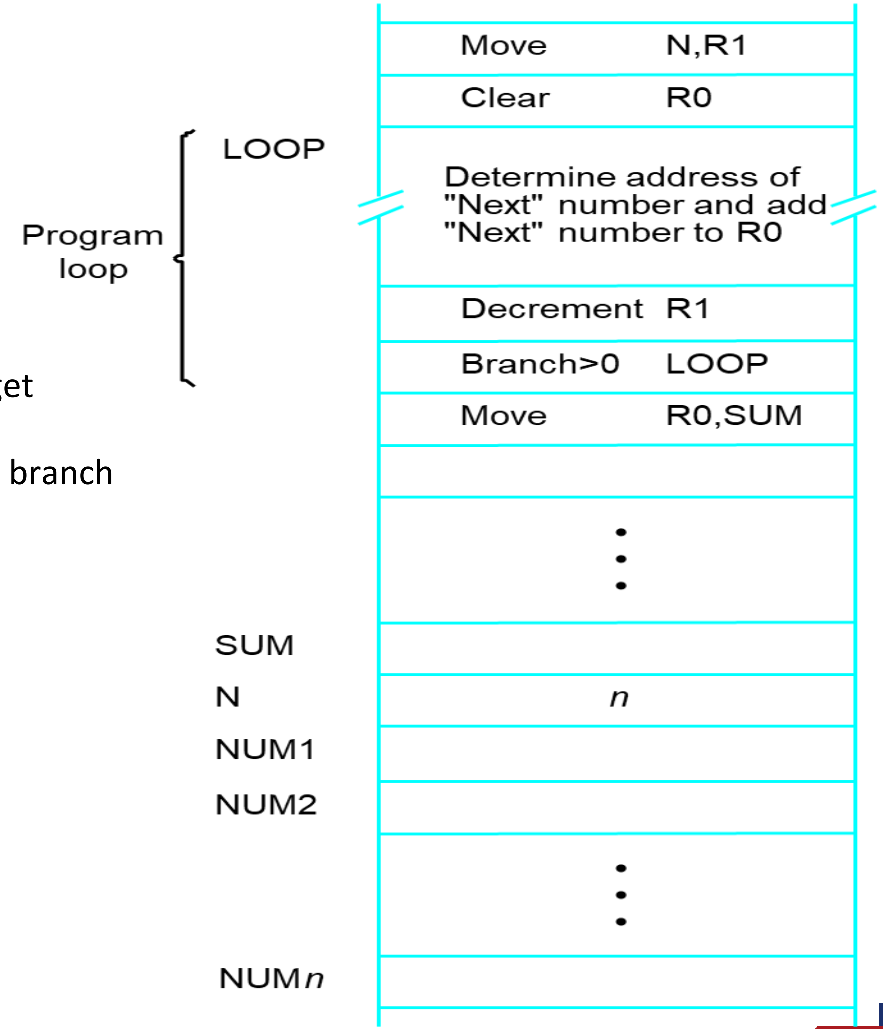


Figure. A straight-line program for adding *n* numbers.

Branching *(Contd.)*



Branch target

Conditional branch

Figure. Using a loop to add *n* numbers.

Condition Codes

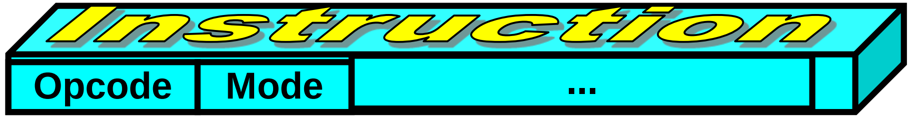
- Condition code flags
- Condition code register / status register
- N (negative)
- Z (zero)
- V (overflow)
- C (carry)
- Different instructions affect different flags

Addressing Modes

Generating Memory Addresses

- How to specify the address of branch target?
- Can we give the memory operand address directly in a single Add instruction in the loop?
- Use a register to hold the address of NUM1; then increment by 4 on each pass through the loop.

Addresses Modes

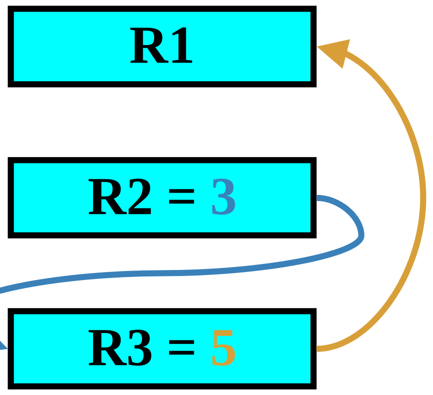


- Implied
  - AC is implied in "ADD   M[AR]" in "One-Address" instr.
  - TOS is implied in "ADD" in "Zero-Address" instr.
- Immediate
  - The use of a constant in "MOV   R1, 5", i.e. R1 ← 5
- Register
  - Indicate which register holds the operand

Addresses Modes *(Contd.,)*

- Register Indirect
  - ✓ Indicate the register that holds the number of the register that holds the operand

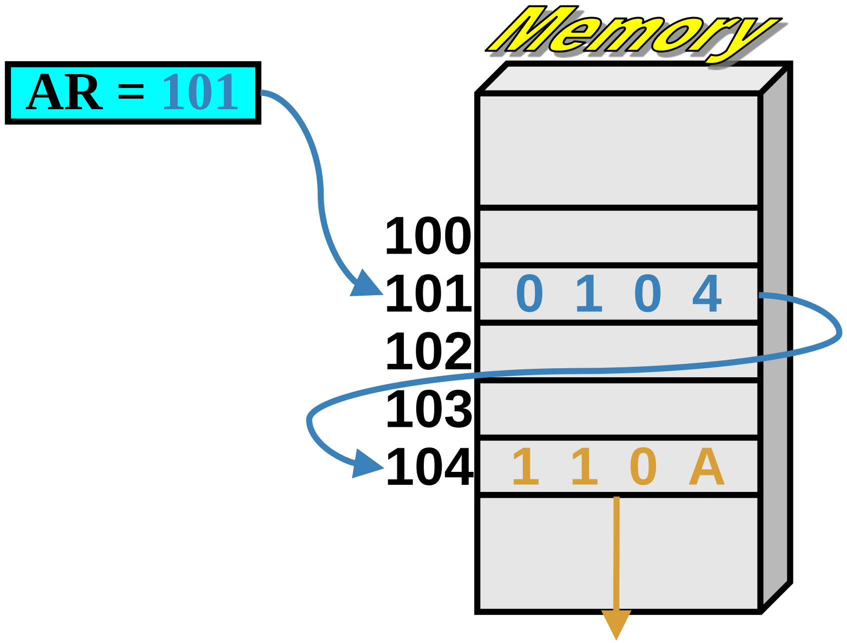
    MOV   R1, (R2)



- Autoincrement / Autodecrement
  - ✓ Access & update in 1 instr.
- Direct Address
  - ✓ Use the given address to access a memory location

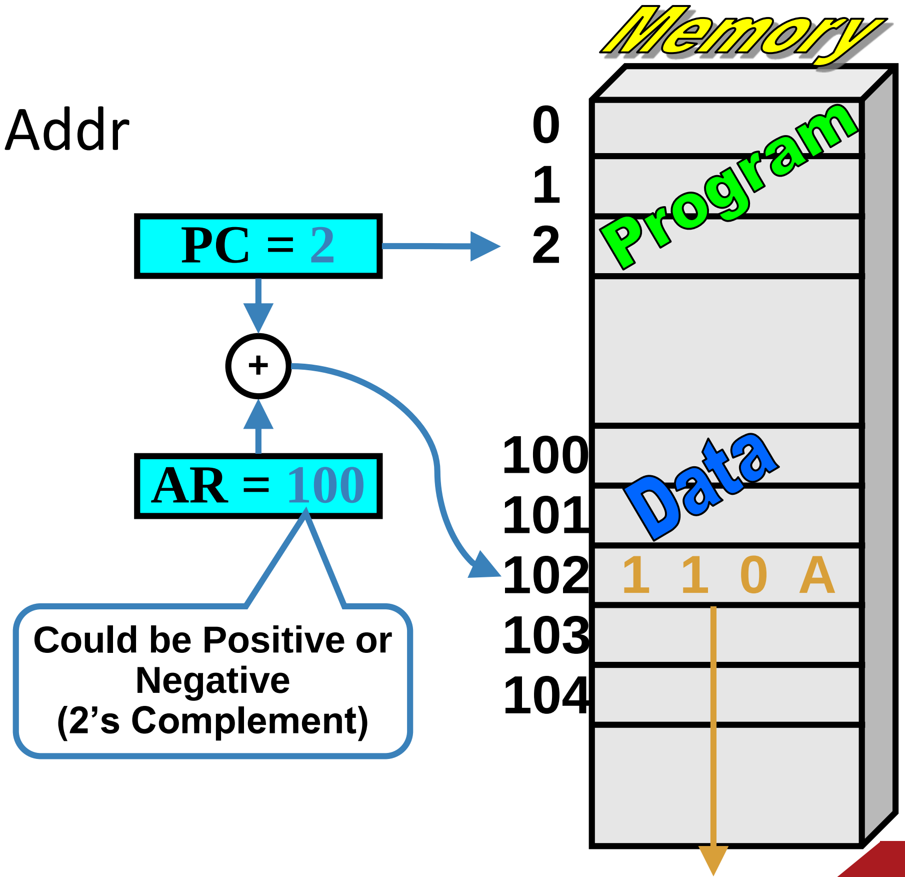
Addresses Modes *(Contd.,)*

- Indirect Address
  - ✓ Indicate the memory location that holds the address of the memory location that holds the data



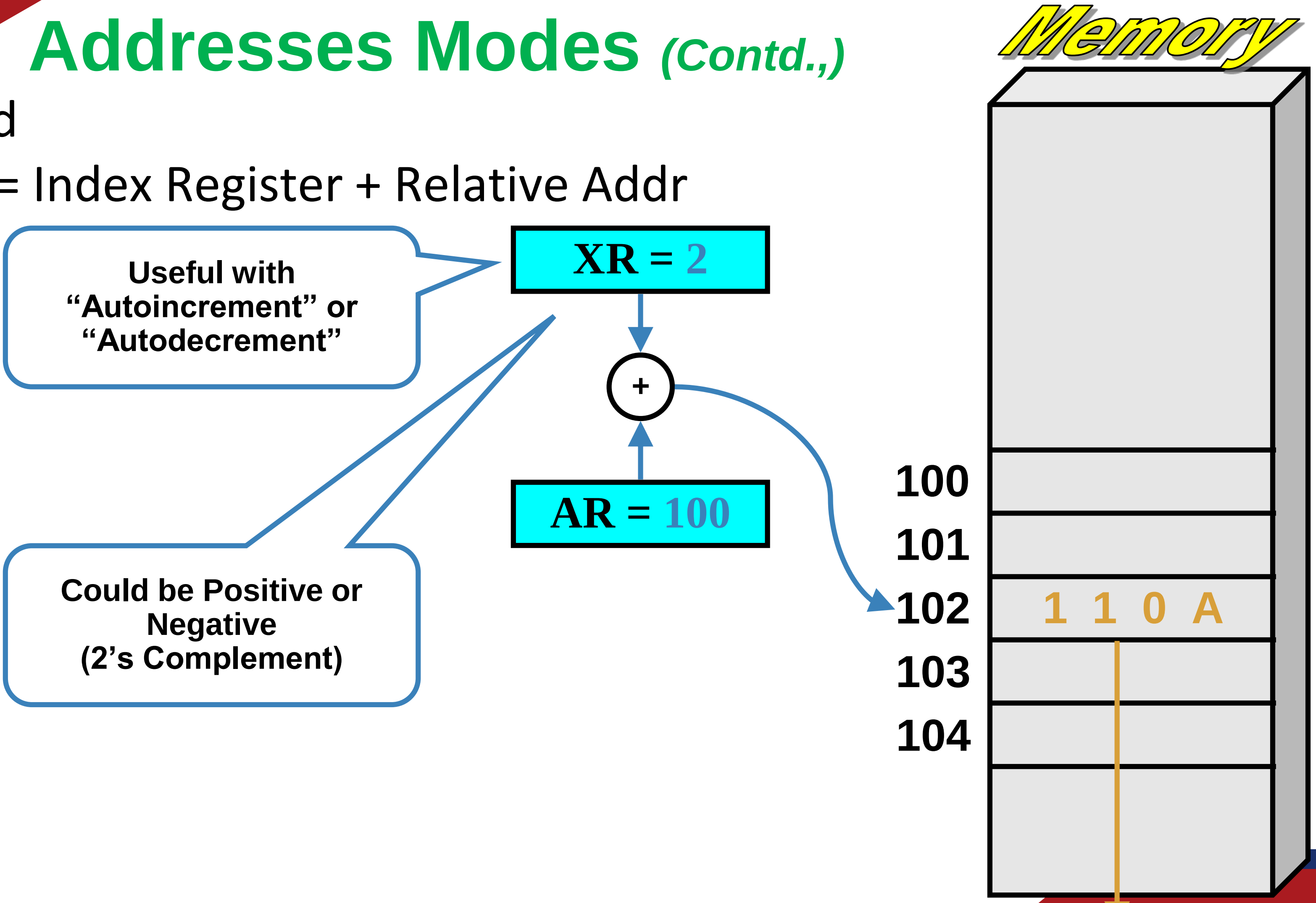
Addresses Modes *(Contd.,)*

- Relative Address
  - ✓ EA = PC + Relative Addr



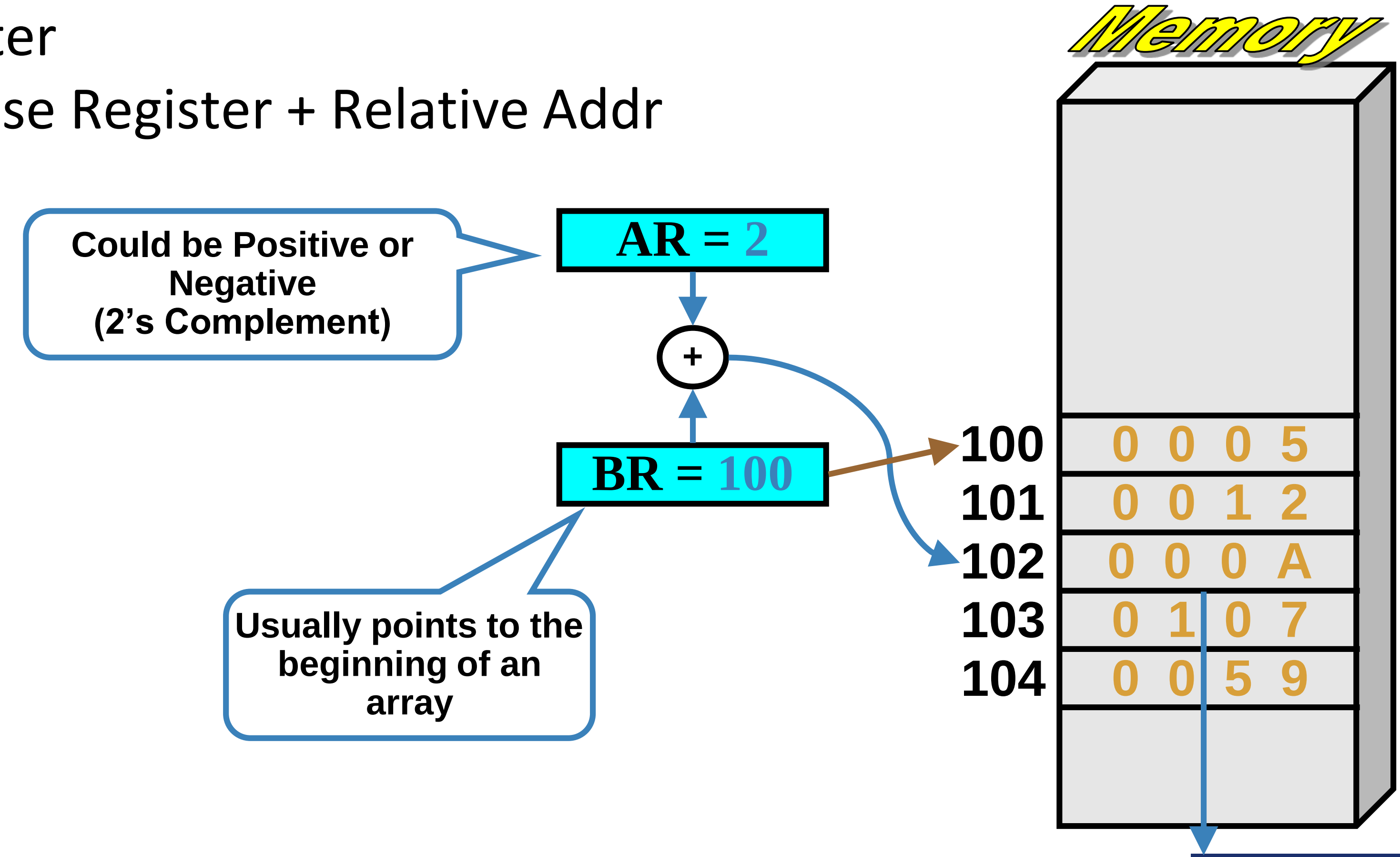
Addresses Modes *(Contd.,)*

- Indexed
  - ✓ *EA* = Index Register + Relative Addr


Memory
Useful with "Autoincrement" or "Autodecrement"
XR = 2
+
AR = 100
Could be Positive or Negative (2's Complement)
100
101
102
103
104
1 1 0 A

Addresses Modes *(Contd.,)*

- Base Register
  - ✓ *EA* = Base Register + Relative Addr



Memory
Could be Positive or Negative (2's Complement)
AR = 2
+
BR = 100
Usually points to the beginning of an array
100 0 0 0 5
101 0 0 1 2
102 0 0 0 A
103 0 1 0 7
104 0 0 5 9

Addresses Modes with Example, Meaning, and Usage

- Register Addressing Mode

  Add R4,R3

  Regs[R4] ← Regs[R4] + Regs[R3]

  When a value is in a register

- Immediate Addressing Mode

  Add R4,#3

  Regs[R4] ← Regs[R4] + 3

  For constants.

- Displacement Addressing Mode

  Add R4,100(R1)

  Regs[R4] ← Regs[R4] + Mem[100+Regs[R1]]

  Accessing local variables (+ simulates register indirect, direct addressing modes).

Addresses Modes with Example, Meaning, and Usage

- Register indirect Addressing Mode

  Add R4,(R1)

  Regs[R4] ← Regs[R4] + Mem[Regs[R1]]

  Accessing using a pointer or a computed address.

- Indexed Addressing Mode

  Add R3,(R1+R2)

  Regs[R3] ← Regs[R3] + Mem[Regs[R1]+Regs[R2]]

  Sometimes useful in array addressing: R1 = base of array; R2 = index amount.

- Direct or absolute Addressing Mode

  Add R1,(1001)

  Regs[R1] ← Regs[R1] + Mem[1001]

  Sometimes useful for accessing static data; address constant may need to be large.

Addresses Modes with Example, Meaning, and Usage

- Memory indirect Addressing Mode

  Add R1,@(R3)

  Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]

  If R3 is the address of a pointer p, then mode yields *p.

- Autoincrement Addressing Mode

  Add R1,(R2)+

  Regs[R1] ← Regs[R1] + Mem[Regs[R2]]

  Regs[R2] ← Regs[R2] + d

  Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, d.

Addresses Modes with Example, Meaning, and Usage

- Autodecrement Addressing Mode

  Add R1,–(R2)

  Regs[R2] ← Regs[R2] – d

  Regs[R1] ← Regs[R1] + Mem[Regs[R2]]

  Same use as autoincrement. Autodecrement/-increment can also act as push/pop to implement a stack.

- Scaled Addressing Mode

  Add R1,100(R2)[R3]

  Regs[R1] ← Regs[R1] + Mem[100+Regs[R2] + Regs[R3]*d]

  Used to index arrays. May be applied to any indexed addressing mode in some computers.