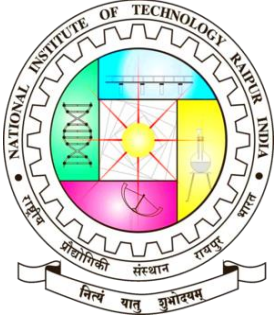


Computer System Architecture

Unit -1

Dr. Aakanksha Sharaff
Assistant Professor
Department of Computer Science and Engineering
National Institute of Technology Raipur

Outline

- **CPU Organization & Functionality**

- CPU Organization
- Fundamental and features
- Data Representation
  - Basic Formats
  - Fixed and Floating Representation
  - Instruction Set
  - Instruction Format
  - Type and Programming Consideration
  - Addressing Modes
- Fixed –Point Arithmetic Multiplication Algorithm
  - Hardware Algorithm
  - Booth Multiplication Algorithm

Computer Architecture and Organization

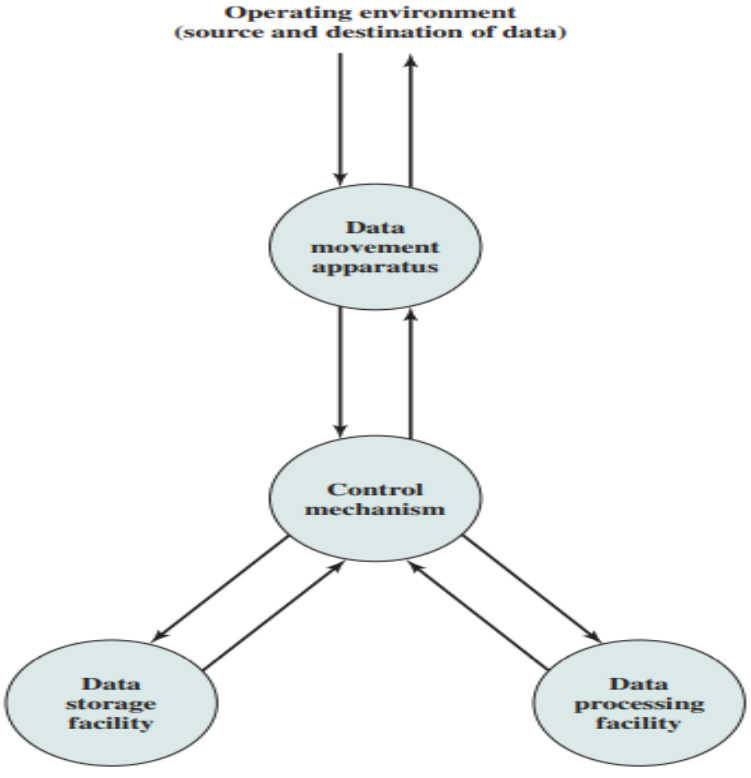
- Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program.
- Computer organization refers to the operational units and their interconnections that realize the architectural specifications.
- Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory.
- Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.

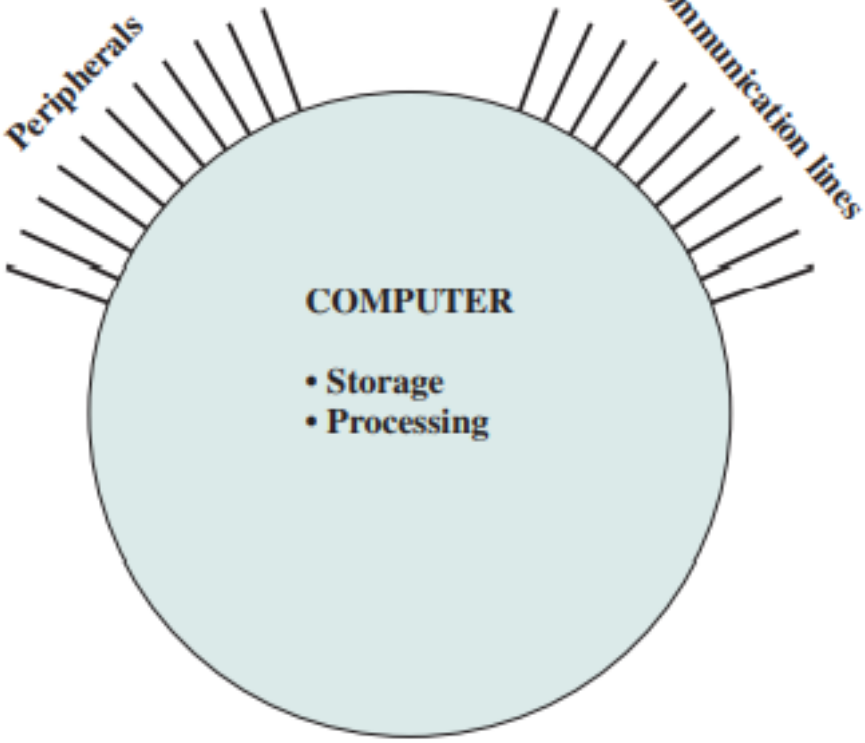
A Functional View of the Computer

Structure: The way in which the components are interrelated.

Function: The operation of each individual component as part of the structure.

- Data processing
- Data storage
- Data movement (Peripheral)
- Control



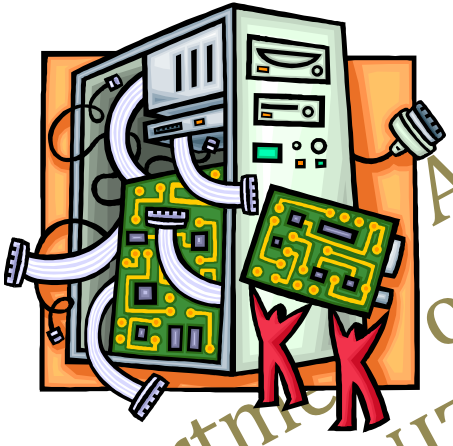


The Computer (Structure)

The computer interacts in some fashion with its external environment. In general, all of its linkages to the external environment can be classified as peripheral devices or communication lines.

Structure of the computer

There are four main structural components of the computer:

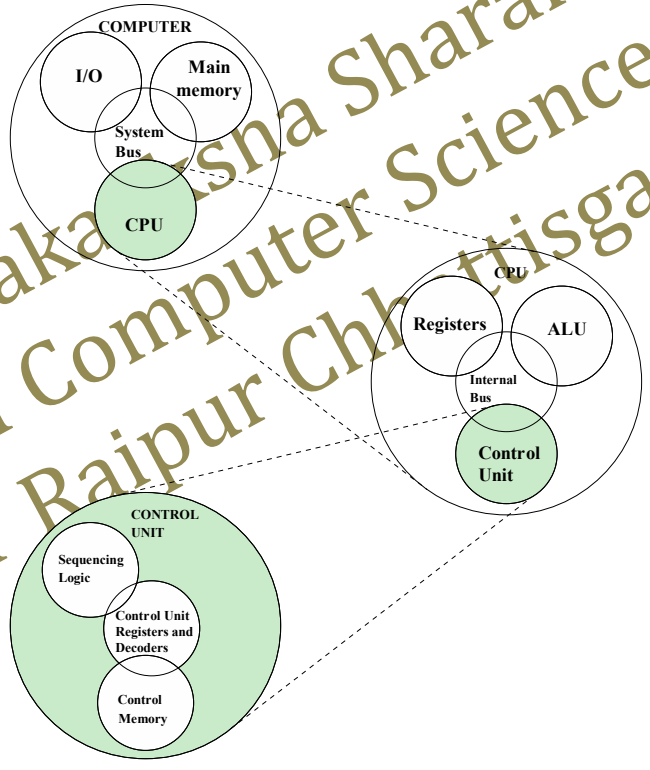


- CPU – controls the operation of the computer and performs its data processing functions
- Main Memory – stores data
- I/O – moves data between the computer and its external environment
- System Interconnection – some mechanism that provides for communication among CPU, main memory, and I/O

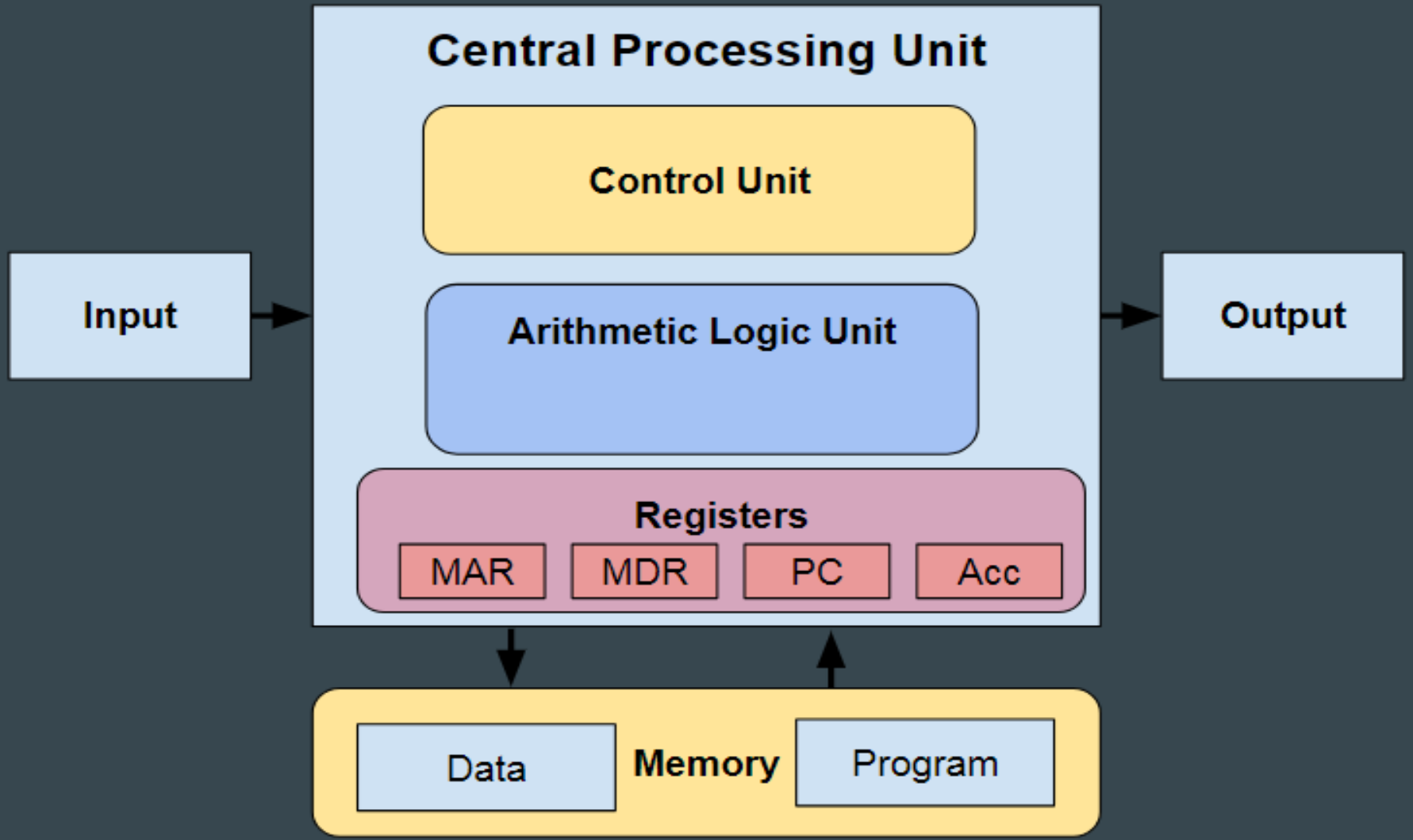
CPU (Central Processing Unit)

- Control Unit
  - →Controls the operation of the CPU and hence the computer
- Arithmetic and Logic Unit (ALU)
  - →Performs the computer's data processing function
- Registers
  - →Provide storage internal to the CPU. It is a temporary storage area in CPU.
- CPU Interconnection
  - →Some mechanism that provides for communication among the control unit, ALU, and registers

Structure of the computer: Top Level Structure



Von Neumann Architecture Diagram



Registers

Memory buffer register (MBR): Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.

Memory address register (MAR): Specifies the address in memory of the word to be written from or read into the MBR.

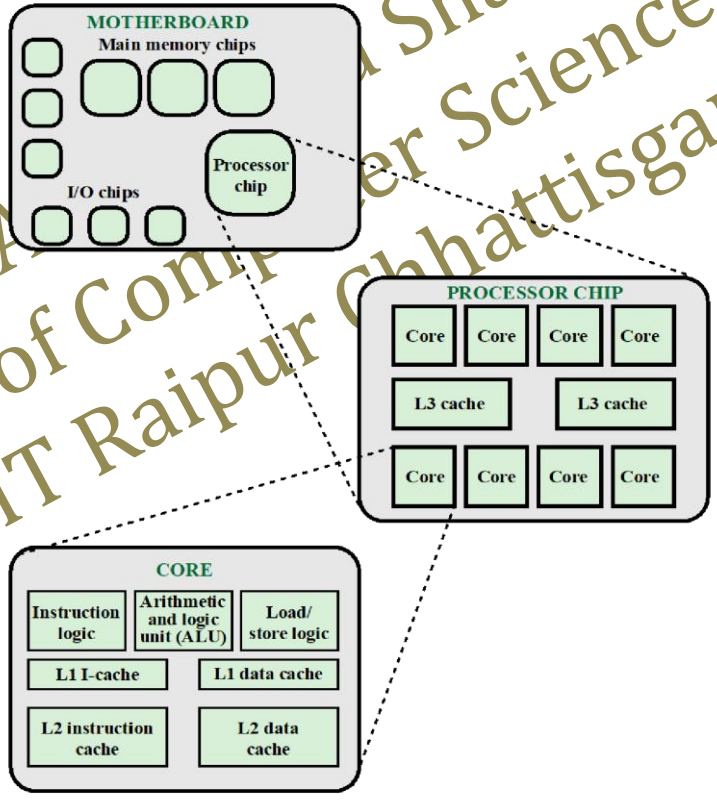
Instruction register (IR): Contains the 8-bit opcode instruction being executed.

Instruction buffer register (IBR): Employed to hold temporarily the righthand instruction from a word in memory.

Program counter (PC): Contains the address of the next instruction pair to be fetched from memory.

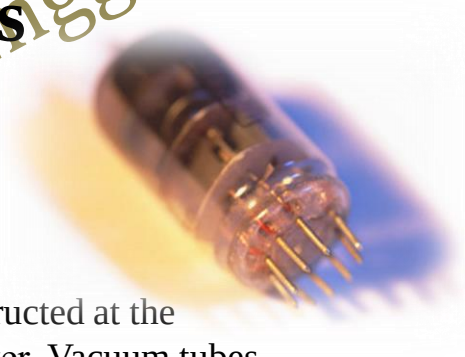
Accumulator (AC) and multiplier quotient (MQ): Employed to hold temporarily operands and results of ALU operations. For example, the result of multiplying two 40-bit numbers is an 80-bit number; the most significant 40 bits are stored in the AC and the least significant in the MQ.

Major Component of Multicore computer



History of Computers

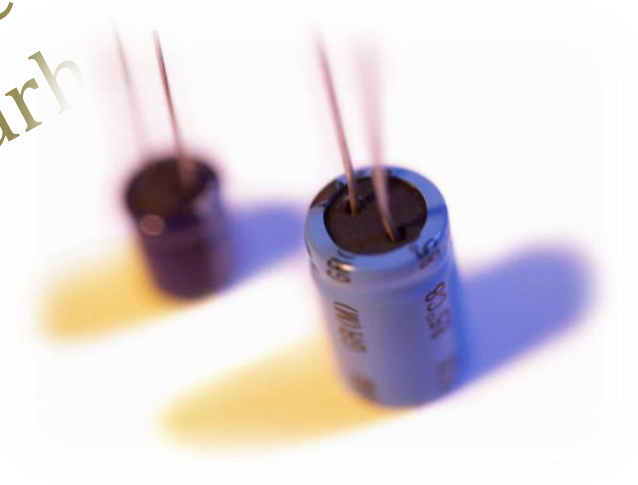
First Generation (1941-1956): Vacuum Tubes



- Vacuum tubes: performs computations
- Magnetic Drums: Storage of data
- ENIAC The ENIAC (Electronic Numerical Integrator And Computer), designed and constructed at the University of Pennsylvania, was the world's first general purpose electronic digital computer. Vacuum tubes were used for digital logic elements and memory
- THE VON NEUMANN MACHINE Fundamental design approach was the stored program concept
    - Attributed to the mathematician John von Neumann
    - Design began at the Princeton Institute for Advanced Studies
    - Completed in 1952
    - Prototype of all subsequent general-purpose computers

Second Generation (1956-1963): Transistors

- Transistor: was invented at Bell Labs in 1947 by Shockley Brattain and Bardeen
- Made from semiconductor materials
- Consumed much less power than vacuum tubes
- Smaller, Cheaper, Faster, Reliable
- Dissipates less heat than a vacuum tube
- Is a *solid state device* made from silicon
- It was not until the late 1950's that fully transistorized computers were commercially available



Second Generation Computers

- Introduced:
  - More complex arithmetic and logic units and control units
  - The use of high-level programming languages
  - Provision of *system software* which provided the ability to:
    - Load programs
    - Move data to peripherals
    - Libraries perform common computations
- Different types of peripheral devices such as printers, magnetic tape drives, magnetic disk storage and punch cards for program input started to appear, and system programs such as rudimentary operating systems were developed.

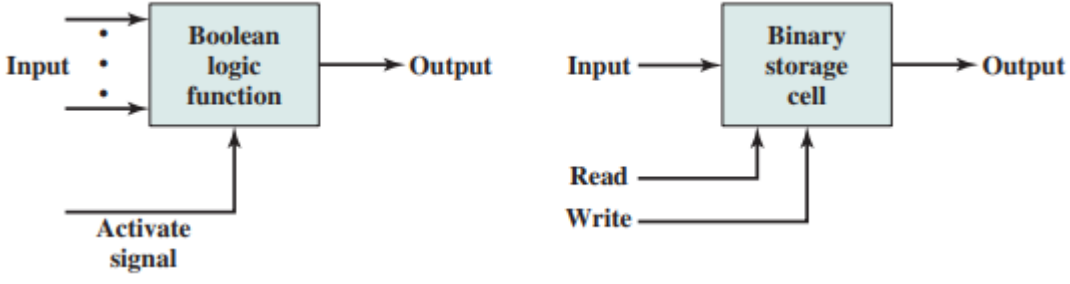
Third Generation (1964-1971): Integrated Circuits

- 1958 – the invention of the integrated circuit by Jack Kilby
- *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
- High-level languages such as COBOL (Command Business Oriented language) and FORTRAN (Formula Transistor) were used to write large programs

Third Generation: Integrated Circuits

- Data storage – provided by memory cells
- Data processing – provided by gates
- Data movement – the paths among components are used to move data from memory to memory and from memory through gates to memory
- Control – the paths among components can carry control signals



Fundamental Computer Elements

Integrated Circuits

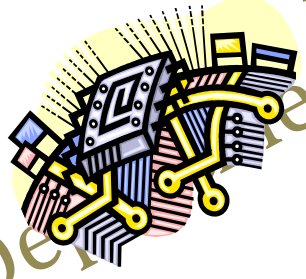
- A computer consists of gates, memory cells, and interconnections among these elements
- The gates and memory cells are constructed of simple digital electronic components
- The integrated circuit exploits the fact that such components as transistors, resistors, and conductors can be fabricated from a semiconductor such as silicon.
- Many transistors can be produced at the same time on a single wafer of silicon
- Initially, only a few gates or memory cells could be reliably manufactured and packaged together. These early integrated circuits are referred to as smallscale integration (SSI).

Later Generations

VLSI
Very Large
Scale
Integration

LSI
Large
Scale
Integration

ULSI
Ultra Large
Scale
Integration



Semiconductor Memory
Microprocessors

Fourth Generation Computers (1971-Present)

- Very large scale integration (VLSI)
- Ultra large scale integration (ULSI) Ics made it possible to pack millions of components into a small chip.
- This reduced the size and price of computers at the same time increased their power, efficiency and reliability.
- The early fourth generation computers were the minicomputers.
- Computer sizes kept shrinking from minicomputers to desktops, to laptops and more recently to hand held devices.

Fifth Generation Computers (Present-Beyond)

- These computers are expected to heavily incorporate artificial intelligence (AI) techniques and are expected to take audio and visual user commands and carry out the instructions.
- Use parallel processing and have enormous computing powers at their disposal.

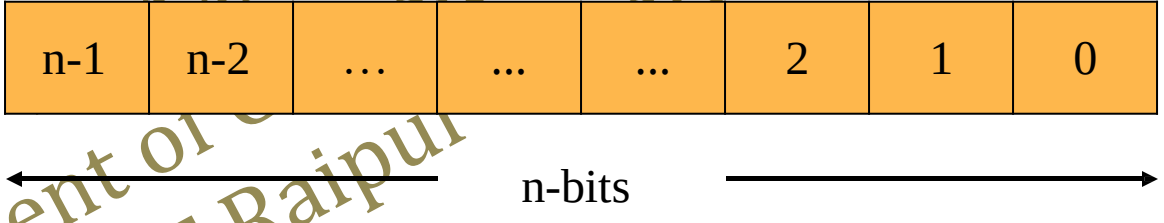
Computer Generations

Generation	Approximate Dates	Technology	Typical Speed (operations per second)
1	1946–1957	Vacuum tube	40,000
2	1957–1964	Transistor	200,000
3	1965–1971	Small and medium scale integration	1,000,000
4	1972–1977	Large scale integration	10,000,000
5	1978–1991	Very large scale integration	100,000,000
6	1991-	Ultra large scale integration	>1,000,000,000

(William Stalling book)

Data Representation in Computers

- Data are stored in Registers
- Registers are limited in number & size



- With a *n*-bit register
  - Min value 0
  - Max value 2^n-1

Data Representation

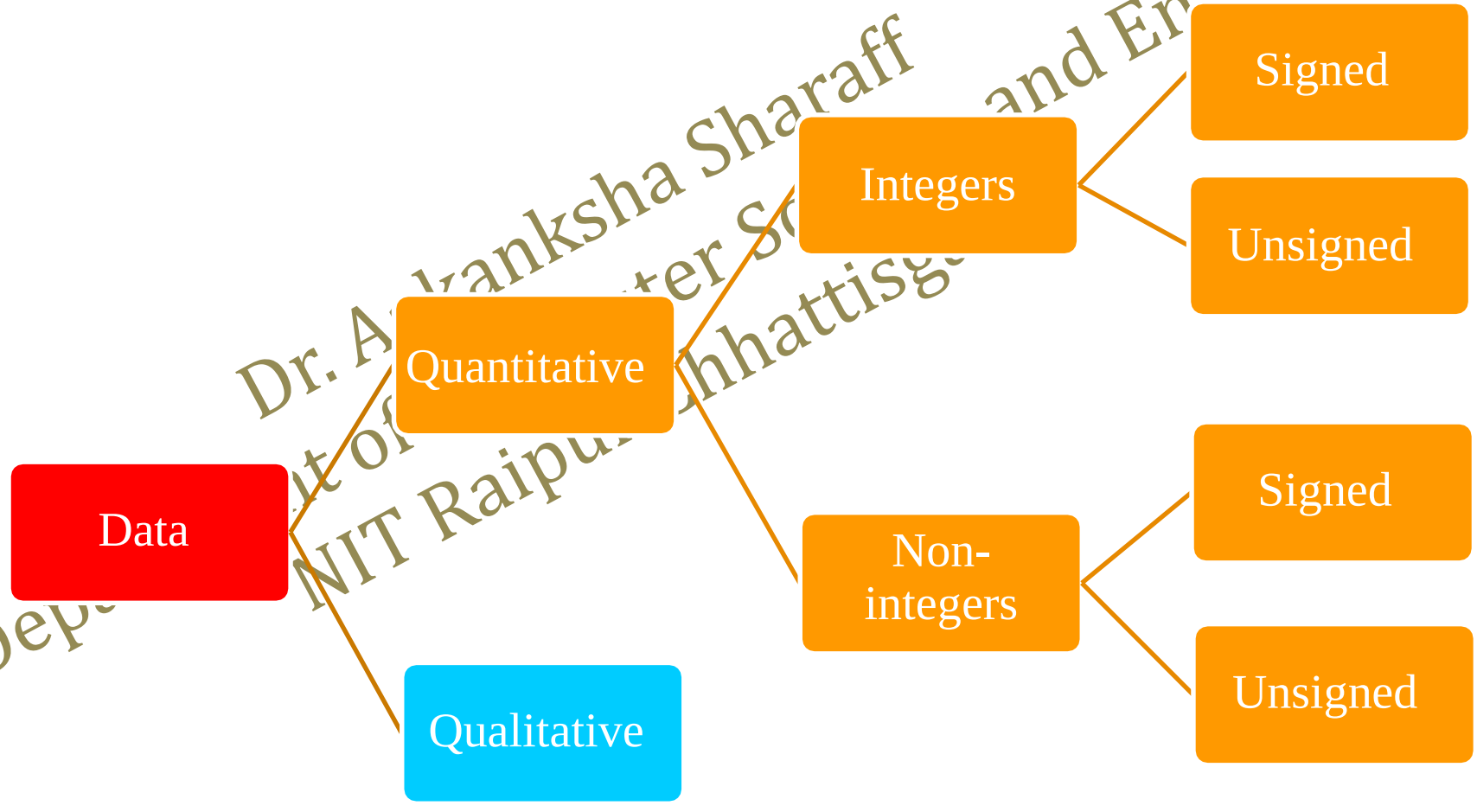
Qualitative

- Represents quality or characteristics
- Not proportional to a value
- Name, NIC no, index no, Address

Quantitative

- Quantifiable
- Proportional to value α CS2052, GPA
- No of students, marks for

Data Representation (Cont.)



Quantitative Numbers

- Integers
  - Unsigned 20
  - Signed +20, -20
- Non-integers
  - Floating point numbers - 10.25, 3.33333…, 1/8 = 0.125

Data Representation in Computers: Number Systems

- Decimal
- Binary
- Octal
- Hexadecimal

Number Systems (Unsigned Number Representation)

- Decimal number system

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- Binary number system

  0, 1

- Octal number system

  0, 1, 2, 3, 4, 5, 6, 7

- Hexadecimal number system

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Data Representation in Computers: Number Systems (Conversion)

- Decimal to Binary
- Decimal to Octal
- Decimal to Hexadecimal
- Binary to Decimal
- Binary to Octal
- Binary to Hexadecimal
- Octal to Binary
- Octal to Decimal
- Octal to Hexadecimal
- Hexadecimal to Binary
- Hexadecimal to Decimal
- Hexadecimal to Octal

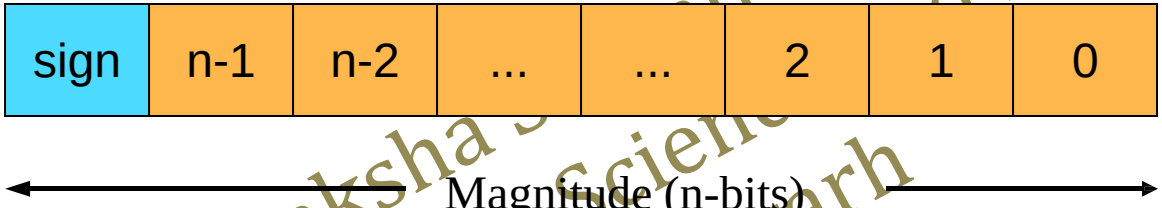
Number Systems (Conversion)

- 724.5 decimal to binary = 1011010100.1
- 153 d to octal =231
- 0.6875 d to binary = .1011
- 101101 b to decimal = 45
- 101101.1101010 b to octal = 55.65
- 101101.1101010 b to hexadecimal = 2D.D4
- 7365.652 to binary = 1110111110101.110101010
- 7365.652 to decimal = 3829.83203125
- 7365.652 to hexadecimal = EF5.D5
- F3D.E2A to decimal = 3901.885253
- F3D.E2A to binary = 111100111101.11100010101
- F3D.E2A to octal = 7475.7052

Signed Integers

- We need a way to represent negative values
- 3 representations
  - Signed Magnitude representation (S&M)
  - One's Complement representation
  - Two's Complement representation

Signed Magnitude Representation



- *n*-bit unsigned magnitude & sign bit (S)
- If S
  - 0 – Integer is positive or zero
  - 1 – Integer is negative or zero
- Range **$-(2^{n-1}-1)$** to **$+(2^{n-1}-1)$**

Signed Magnitude Representation

- If 8-bit register is used what are min & max numbers?
- What are 0000 0000 and 1000 0000 in decimal?
  - Representation of zero is not unique

Signed Magnitude Representation

- Advantages
  - Sign reversal
  - Finding absolute value |a|
    - Flip sign bit
- Disadvantage
  - Adding a negative of a number is not the same as subtraction
    - e.g., add 2 and -3
    - Need different operations
  - Zero is not unique

Complement Method

- Base = Radix
  - Radix *r* system means *r* number of symbols
  - e.g., binary numbers have symbols 0, 1
- 2 types
  - *r*'s complement
  - (*r* – 1)'s complement
  - Where *r* is radix (base) of number system
- Examples
  - Decimal 9's & 10's complement
  - Binary 1's & 2's complement

Complement Method – Definition

- Given a number m in base/radix r & having n digits
  - (r – 1)'s complement of m is

    $$(r^n - 1) - m$$

  - r 's complement of n is

    $$(r^n - 1) - m + 1 = r^n - m$$

Example – Complement Method

- If *m* = 5982 & n = 4 digits
- 9's complement is

```
  9 9 9 9
- 5 9 8 2
---------
  4 0 1 7
```

- maximum representable no

- 10's complement

```
  9 9 9 9
-   5 9 8 2
-----------
  4 0 1 7
+       1
-----------
  4 0 1 8
```

or

```
  1 0 0 0 0
-   5 9 8 2
-----------
  4 0 1 8
```


$-2^{n-1}+1$ to $2^{n-1}-1$

-7 to $+7$

4
$+(-5)$
9

0110
1101
0011
1
0100

Example – Complement Method

- If $m = 382$ & $n = 3$
  - $-n = -382 =$

$$\begin{array}{r} 999 \\ -382 \\ \hline 617 \end{array} \qquad \begin{array}{r} 999 \\ -382 \\ \hline 617 \\ +\quad 1 \\ \hline 618 \end{array}$$

  - -382 = 617 or 618
    - Depending on which complement we use
      - These are called complementary pair

1’s complement

- Calculated by
  - **$(2^n - 1) - m$**
- If m = 0101
  - 1’s complement of m on a 4-bit system

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ -\ 0\ 1\ 0\ 1 \\ \hline 1\ 0\ 1\ 0 \end{array}$$

  - This represents -5 in 1’s complement

Finding 1's Complement – Short Cut

- Invert each bit of *m*
- Example
  - *m* = 0 0 1 0 1 0 1 1
  - 1's complement of *m* = 1 1 0 1 0 1 0 0
  - m = 0 0 0 0 0 0 0 0 = 0
  - 1's complement of *m* = 1 1 1 1 1 1 1 1 = -0
- Values range from -127 to +127

Addition with 1's Complement

- If results has a carry add it to LSB (Least Significant Bit)
- Example
  - Add 6 and -3 on a 3-bit system

  - 6 = 1 1 0
  - -3 = 1 0 0

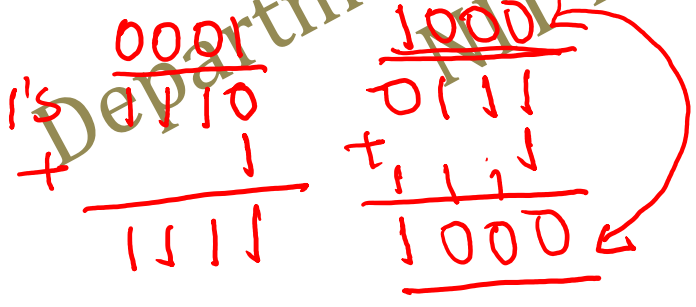
  = 1 0 1 0

  ↳ 1

  0 1 1

2’s Complement

- Doesn’t require end-around carry operation as in 1’s complement
- 2’s complement is formed by
  - Finding 1’s complement
  - Add 1 to LSB
- New range is from -128 to +127
  - -128 because of +1 to negative value



Example – 2's Complement

- Find 2's complement of 0101011
  - *m* = 0 1 0 1 0 1 1
    = 1 0 1 0 1 0 0
    1 +
  - 2's = 1 0 1 0 1 0 1

- Short-cut
  1. Search for the 1st bit with value 1 starting from LSB
  2. Inver all bits after 1st one

Example – 2's Complement (Cont.)

- Add 6 and -5 on a 4-bit system 5 = 0101

-5 = 1011

6 = 0 1 1 0

-5 = 1 0 1 1 +

1 0 0 0 1 → 0 0 0 1

Discard

1’s vs. 2’s Complement

- 1’s complement has 2 zeros (+0, -0)
- Value range is less than 2’s complement

- 2’s complement only has a single zero
- Value range is unequal

- No need of a separate subtract circuit

  Doing a NOT operation is much more cost effective interms of circuit design
- However, multiplication & division is slow

Fixed-Point Representation

old position	7	6	5	4	3	2	1	0
New position	4	3	2	1	0	-1	-2	-3
Bit pattern	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**

Contribution 2^4 2^1 2^0 2^{-1} 2^{-3} =19.625

Radix-point

Fixed-Point Representation

+14

- Signed magnitude representation 1 0001110
- Signed 1’s magnitude representation 1 1110001
- Signed 2’s magnitude representation 1 1110010

Fixed-Point Representation Arithmetic Addition

+6	00000110
+13	00001101
+19	00010011

-6	11111010
+13	00001101
+7	00000111

+6	00000110
-13	11110011
-7	11111001

-6	11111010
-13	11110011
-19	11101101

Limitation of Fixed-Point Representation

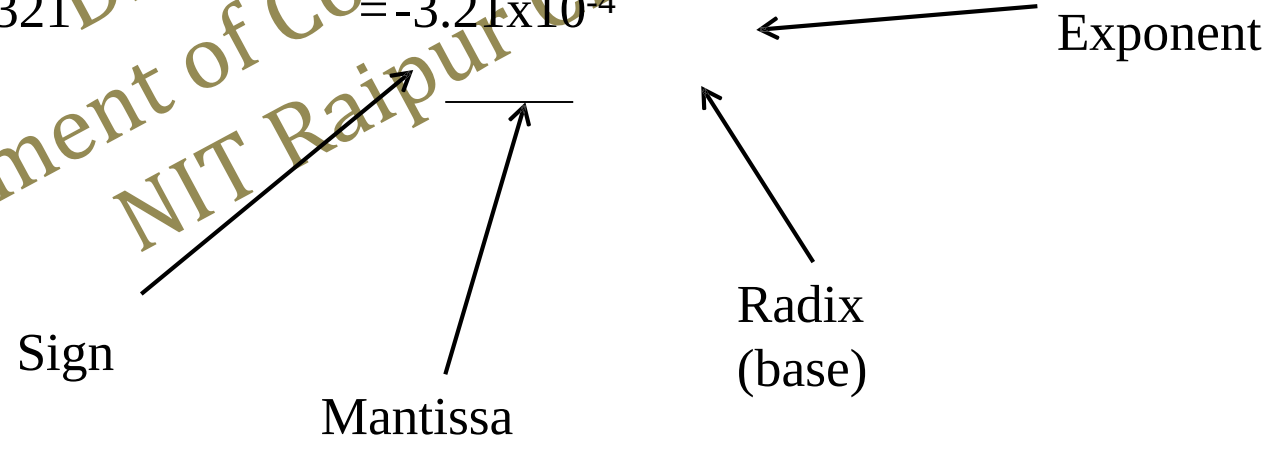
- To represent large numbers or very small numbers we need a very long sequences of bits.

- This is because we have to give bits to both the integer part and the fraction part.

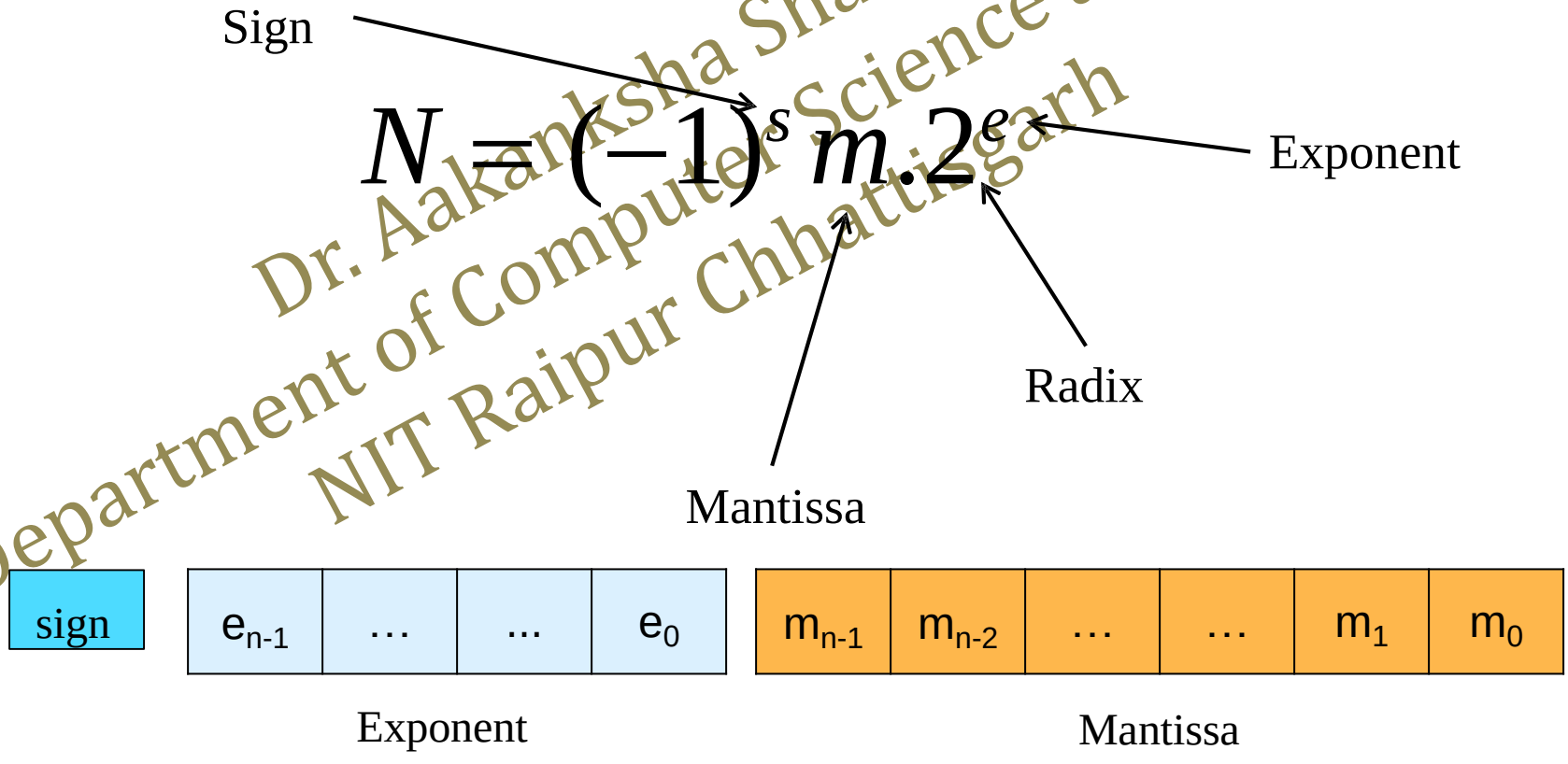
Floating Point Numbers

- We needed to represent fractional values & values beyond $2^n - 1$
- m x r^e

- +3207.23 = 3.20723×10^3
- -0.000321 = -3.21×10^{-4}



Floating Point Numbers



IEEE Floating Point Standard (Cont.)

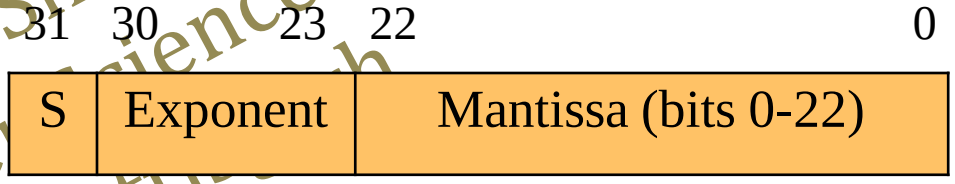
$$N = (-1)^s [1.m] \times 2^{E-127}$$

- E – 127 = e
- E = e + 127
- What is stored is E

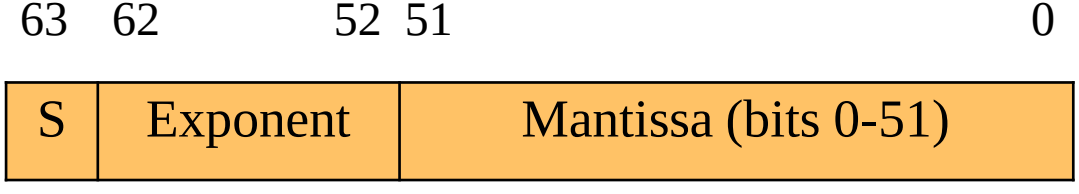
IEEE Floating Point Standard (FPS)

- 2 standards

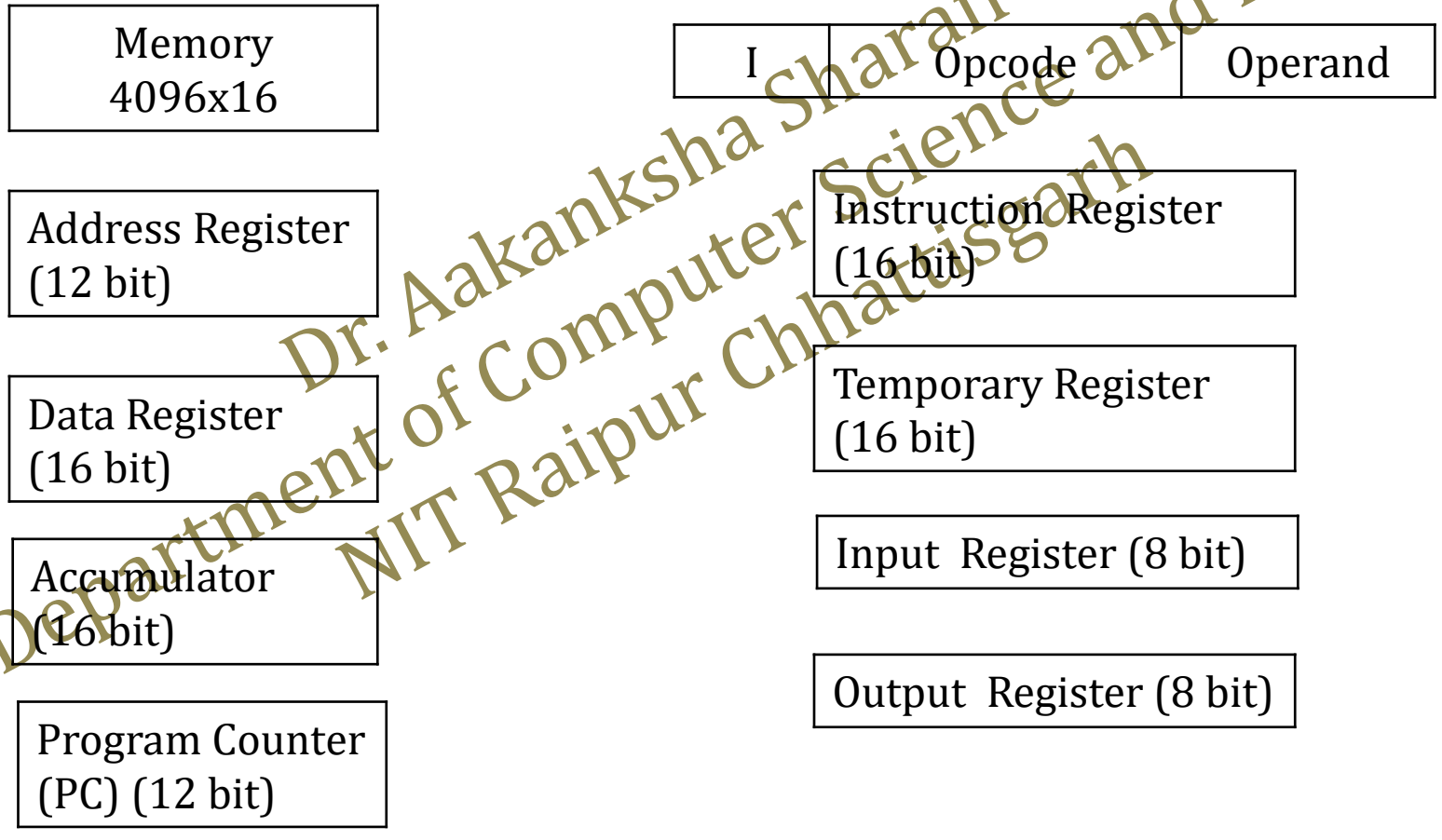
1. Single precision
   - 32-bits
     - 23-bit mantissa
     - 8-bit exponent
     - 1-bit sign



2. Double precision
   - 64-bits
     - 52-bit mantissa
     - 11-bit exponent
     - 1-bit sign

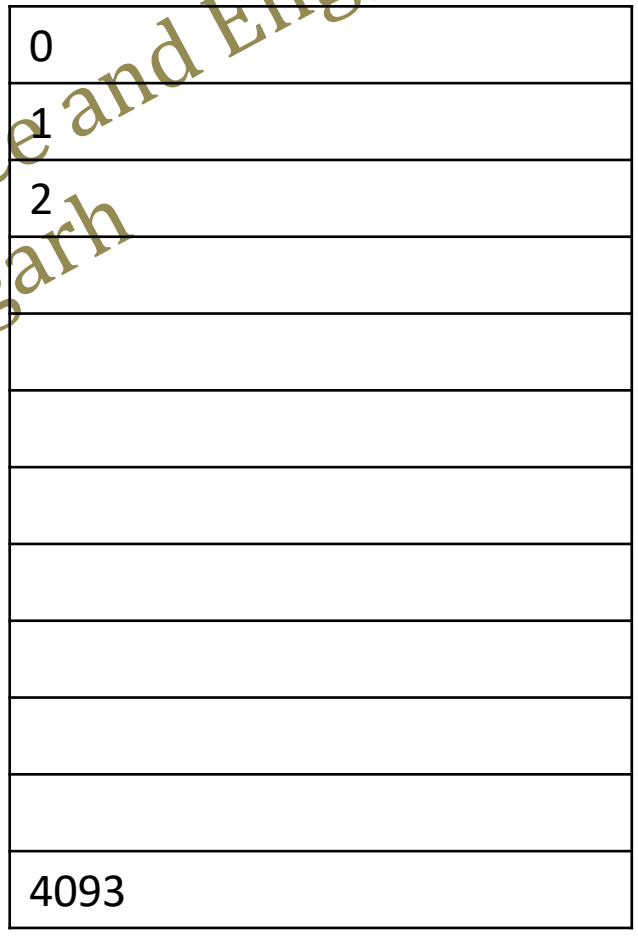


General Purpose Registers



General Purpose Registers

$4096 = 2^{12}$



Types of Instructions

- Data Transfer Instructions
- Data Manipulation Instructions
  - Arithmetic instructions
  - Logical instructions
  - Shift and rotate instructions
- Program Control Instructions
  - If
  - Loop
  - For
  - While
  - Branch
  - Call
  - Return

Data Transfer Instruction

- MOV-
  Move byte or word to register or memory
- LOAD-
  Load word from memory to register generally
- STORE (STA)-
  Store word from register (ACC) to memory
- INPUT, OUTPUT (IN/OUT)-
  Input byte or word from port, output word to port
- PUSH, POP-
  Push word onto stack, pop word off stack
- EXCHANGE (XCHG)-
  Exchange byte or word

Arithmetic Instruction

ADD – ADD Destination, Source

ADC – ADC Destination, Source

These instructions add a number from some source to a number in some destination and put the result in the specified destination.

SUB – SUB Destination, Source

SBB – SBB Destination, Source

These instructions subtract the number in some source from the number in some destination and put the result in the destination.

The SBB instruction also subtracts the content of carry flag from the destination.

INC – INC Destination

The INC instruction adds 1 to a specified register or to a memory location. AF, OF, PF, SF, and ZF are updated, but CF is not affected.

Arithmetic Instruction

❖**MUL – MUL Source**

This instruction multiplies an unsigned byte in some source with an unsigned byte in AL register or an unsigned word in some source with an unsigned word in AX register.

❖**IMUL – IMUL Source**

This instruction multiplies a signed byte from source with a signed byte in AL or a signed word from some source with a signed word in AX .

❖**DIV – DIV Source**

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word (32 bits) by a word.

❖**IDIV – IDIV Source**

This instruction is used to divide a signed word by a signed byte, or to divide a signed double word by a signed word.

Arithmetic Instruction

- **DEC – DEC Destination**

  This instruction subtracts 1 from the destination word or byte.

- **Negate**

Logical Instruction

- **AND – AND Destination, Source**

  This instruction ANDs each bit in a source byte or word with the same numbered bit in a destination byte or word.

- **OR – OR Destination, Source**

  This instruction ORs each bit in a source byte or word with the same numbered bit in a destination byte or word.

- **XOR – XOR Destination, Source**

  This instruction Exclusive-ORs each bit in a source byte or word with the same numbered bit in a destination byte or word.

- **NOT – NOT Destination**

  The NOT instruction inverts each bit (forms the 1's complement) of a byte or word in the specified destination.

Logical Instruction

- **NEG – NEG Destination**

  This instruction replaces the number in a destination with its 2's complement.

- **CMP – CMP Destination, Source**

  This instruction compares a byte / word in the specified source with a byte / word in the specified destination.

- **TEST – TEST Destination, Source**

  This instruction ANDs the byte / word in the specified source with the byte / word in the specified destination.

Enable Interrupt (EI)

Disable Interrupt (DI)

Flag manipulation Instruction

- **STC:**

  It sets the carry flag to 1.
- **CLC:**

  It clears the carry flag to 0.
- **CMC:**

  It complements the carry flag.
- **STD:**

  It sets the direction flag to 1. If it is set, string bytes are accessed from higher memory address to lower memory address.
- **CLD:**

  It clears the direction flag to 0. If it is reset, the string bytes are accessed from lower memory address to higher memory address.

Shift Instructions

- **Logical Shift Left**
- **Logical Shift Right**
- **Arithmetic Shift Right**
- **Arithmetic Shift Left**
- **Rotate Right**
- **Rotate Left**
- **Rotate right through Carry**
- **Rotate left through Carry**

Program Control Instructions

- Branch
- Call
- Return
- Jump

Dr. Aakanksha Sharaff
Department of Computer Science and Engg.
NIT Raipur Chhattisgarh

Question on Instruction Format

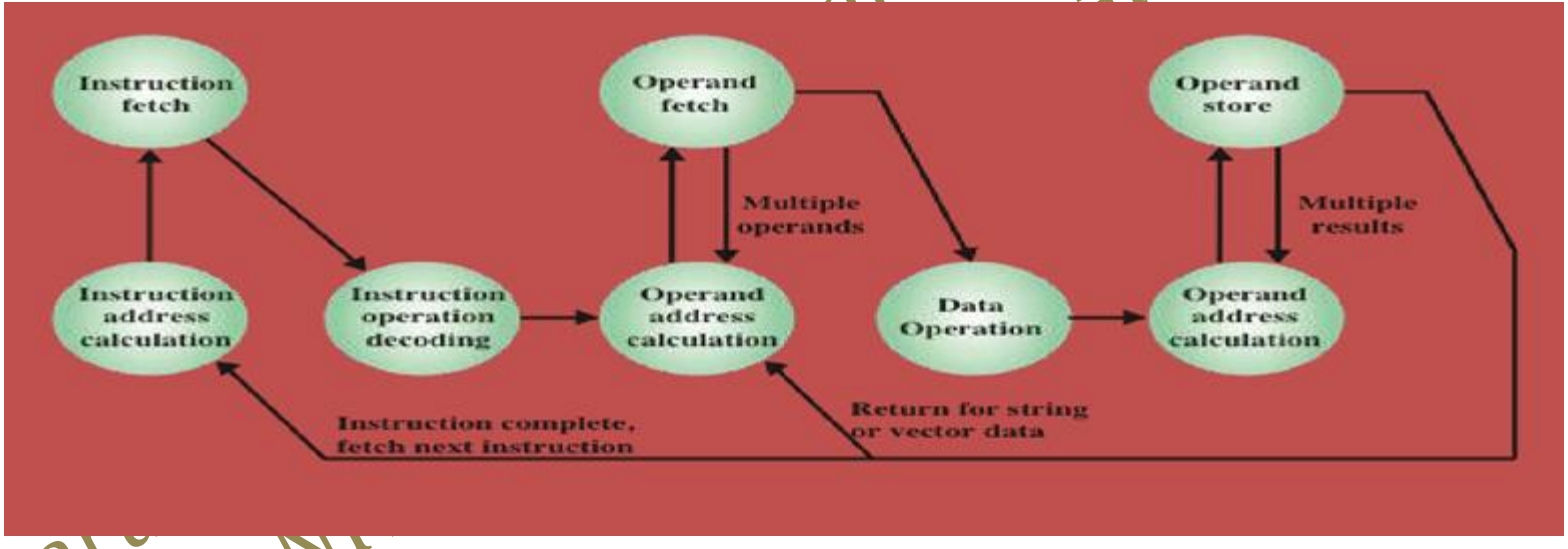
A machine has 24 bit instruction format. It has 32 registers and each of which is 32 bit long. It needs to support 49 instructions. Each instruction has two register operands and one immediate operand. If the immediate operand is signed integer, the minimum value of immediate operand is _____?

a. -32
b. -64
c. -128
d. -256

Instruction Format

- An instruction format or instruction code is a group of bits used to perform a particular operation on the data stored in computer.
- Processor fetches an instruction from memory and decodes the bits to execute the instruction.
- Different computer may have their own instruction set .
- The operation of the processor is determined by the instructions it executes, referred to as machine instructions or computer instructions
- The collection of different instructions that the processor can execute is referred to as the processor's instruction set
- Each instruction must contain the information required by the processor for execution

Instruction Cycle State Diagram



Source and result operands can be in one of four areas:

1) Main memory or virtual memory

- As with next instruction references, the main or virtual memory address must be supplied.

2) I/O devices

- The instruction must specify the I/O module and device for the operation. If memory-mapped I/O is used, this is just another main or virtual memory address.

3) Processor register

- A processor contains one or more registers that may be referenced by machine instructions.
- If more than one register exists each register is assigned a unique name or number and the instruction must contain the number of the desired register

4) Immediate

- The value of the operand is contained in a field in the instruction being executed

Types of Operand

- Addresses: immediate, direct, indirect, stack

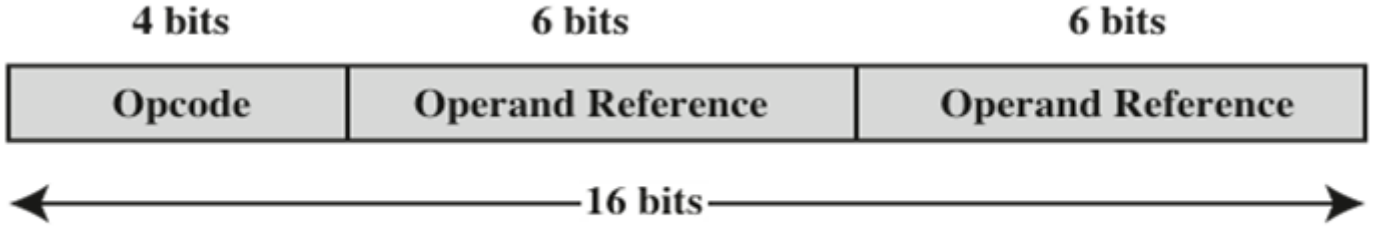
- Numbers: integer or fixed point (binary, twos complement), floating point (sign, exponent), (packed) decimal
  (246 = 0000 0010 0100 0110)

- Characters: ASCII (128 printable and control characters + bit for error detection)

- Logical Data: bits or flags, e.g., Boolean 0 and 1

Instruction Representation

- Within the computer each instruction is represented by a sequence of bits
- The instruction is divided into fields, corresponding to the constituent elements of the instruction



Instruction Types

- A computer should have a set of instructions that allows the user to formulate any data processing task.. Any program written in a high- level language must be translated into machine language to be executed, so we can categorize instruction types as follows:

- Data processing: Arithmetic instructions provide computational capabilities for processing numeric data

- Data storage: Movement of data into or out of register and or memory locations

- Data movement :I /O instructions are needed to transfer programs and data into memory and the results of computations back out to the user

- Control: test instruction test the value of a data word or the status of a computation

- Branch instruction used to branch to a different set of instructions depending on the decision made

Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set
- Instruction Length
  - Affected by and affects:
  - Memory size
  - Memory organization - addressing
  - Bus structure, e.g., width
  - CPU complexity
  - CPU speed
  - Trade off between powerful instruction repertoire and saving space

Allocation of Bits

- Number of addressing modes: implicit or additional bits specifying it
- Number of operands
- Register (faster, limited size and number, 32) versus memory
- Number of register sets, e.g., data and address (shorter addresses)
- Address range
- Address granularity (e.g., by byte)
- Number of Addresses

- More addresses
  1 More complex instructions
  2More registers - inter-register operations are quicker 3-Less instructions per program
- Fewer addresses
  1 Less complex instructions
  2More instructions per program, e.g. data movement 3-Faster fetch/execution of instructions

Pentium Instruction Format

- Pentium uses a variable length instruction format.
- The Pentium instruction can be from 2 to 16 bytes long.
- The Instruction is broken down into 6 sections

1-Instruction Prefixes

2-Opcode

3-Mod R/M

4 SIB

5Displacement

Immediate

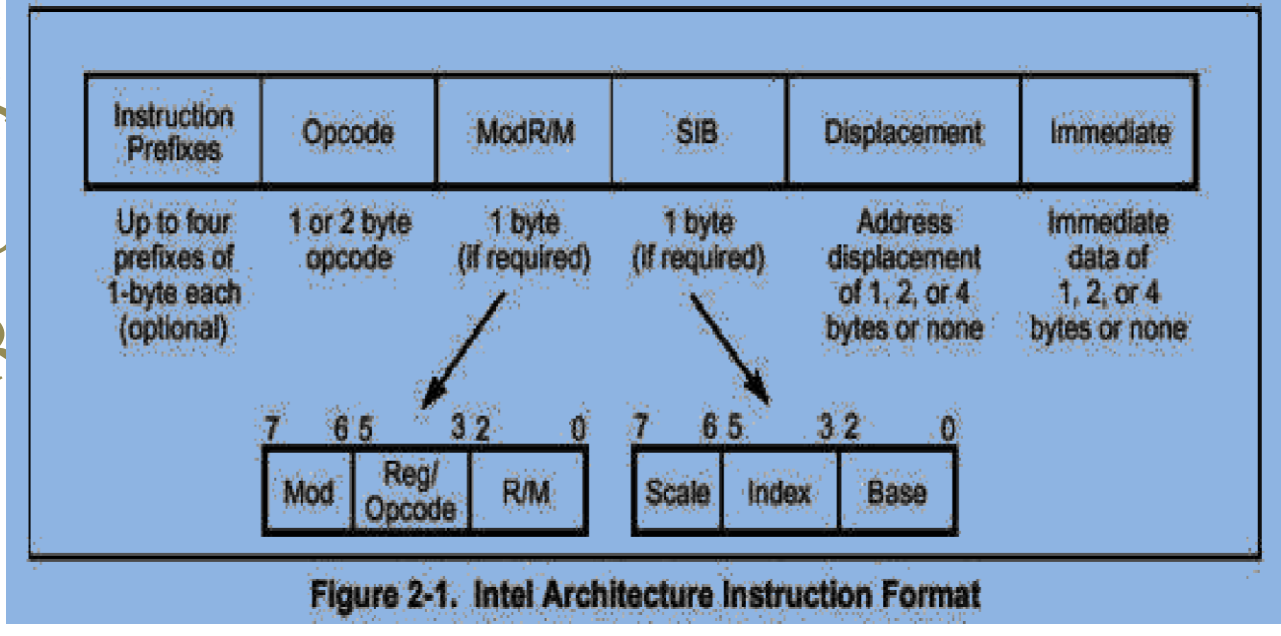
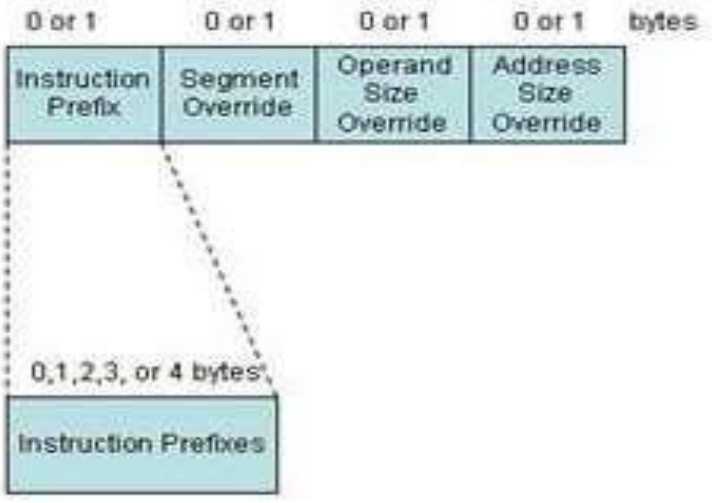


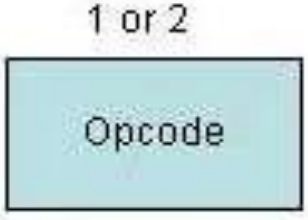
Figure 2-1. Intel Architecture Instruction Format

Instruction Prefixes



- Instruction Prefix:used in multiprocessor enviroments, specifically with strings
- Segment Override: This specifies which segment-register to use, if not the default.
- Operand Size: The operand size can be either 16 or 32 bits, this specifies which is being used.
- Address Size: The Pentium can use a 16 or 32 bit address, this specifies which is being used.

2 .Opcode



- The opcode can be either one or two bytes.
- The opcode can also specify if the data is 16 or 32 bit.
- The opcode specifies which way the data is going (to or from memory).
- The opcode specifies if an immediate value is
  - signed or not.

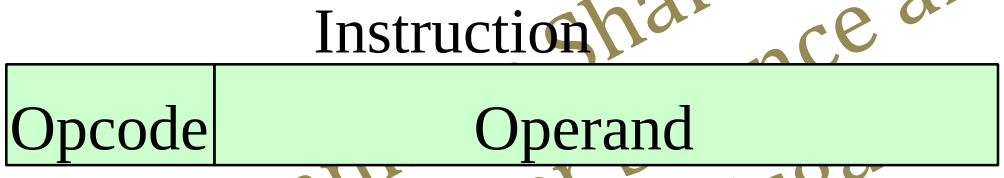
Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement (Indexed)
- Stack

Immediate Addressing

- Operand is part of instruction
- Operand = address field
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range

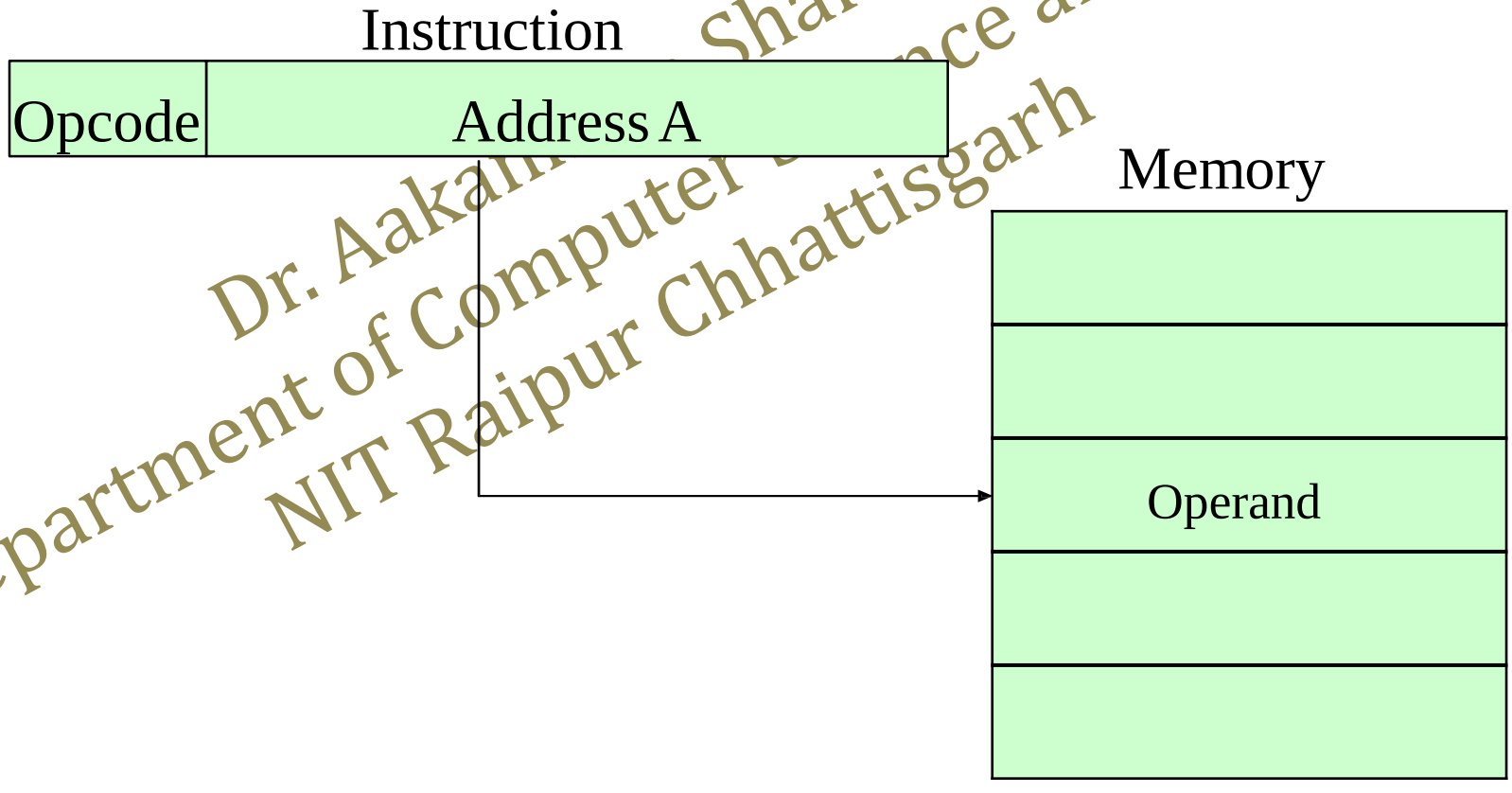
Immediate Addressing Diagram



Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Direct Addressing Diagram



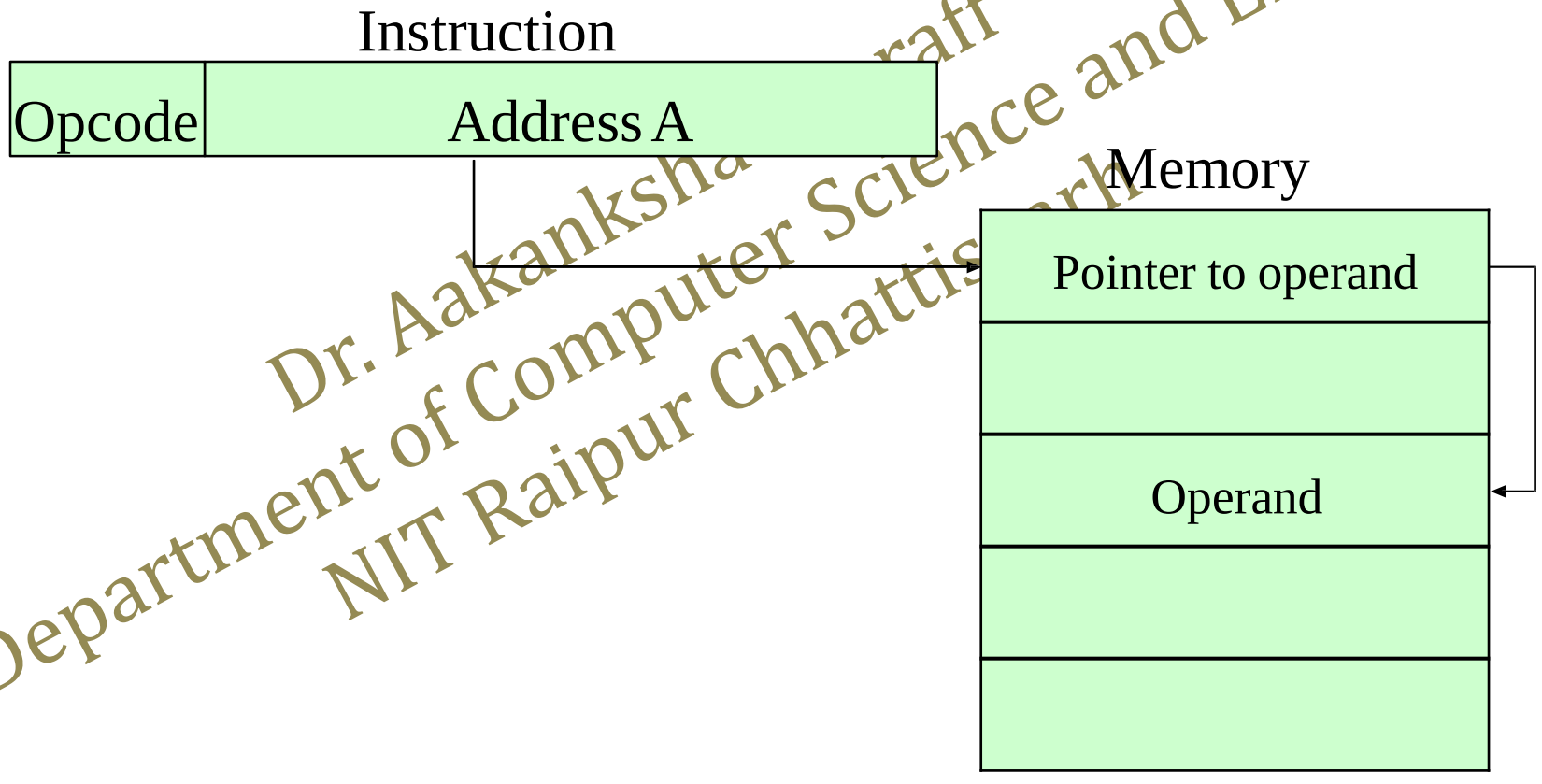
Indirect Addressing (1)

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA = (A)
  - —Look in A, find address (A) and look there for operand
- e.g. ADD (A)
  - —Add contents of cell pointed to by contents of A to accumulator

Indirect Addressing (2)

- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
  - Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower

Indirect Addressing Diagram



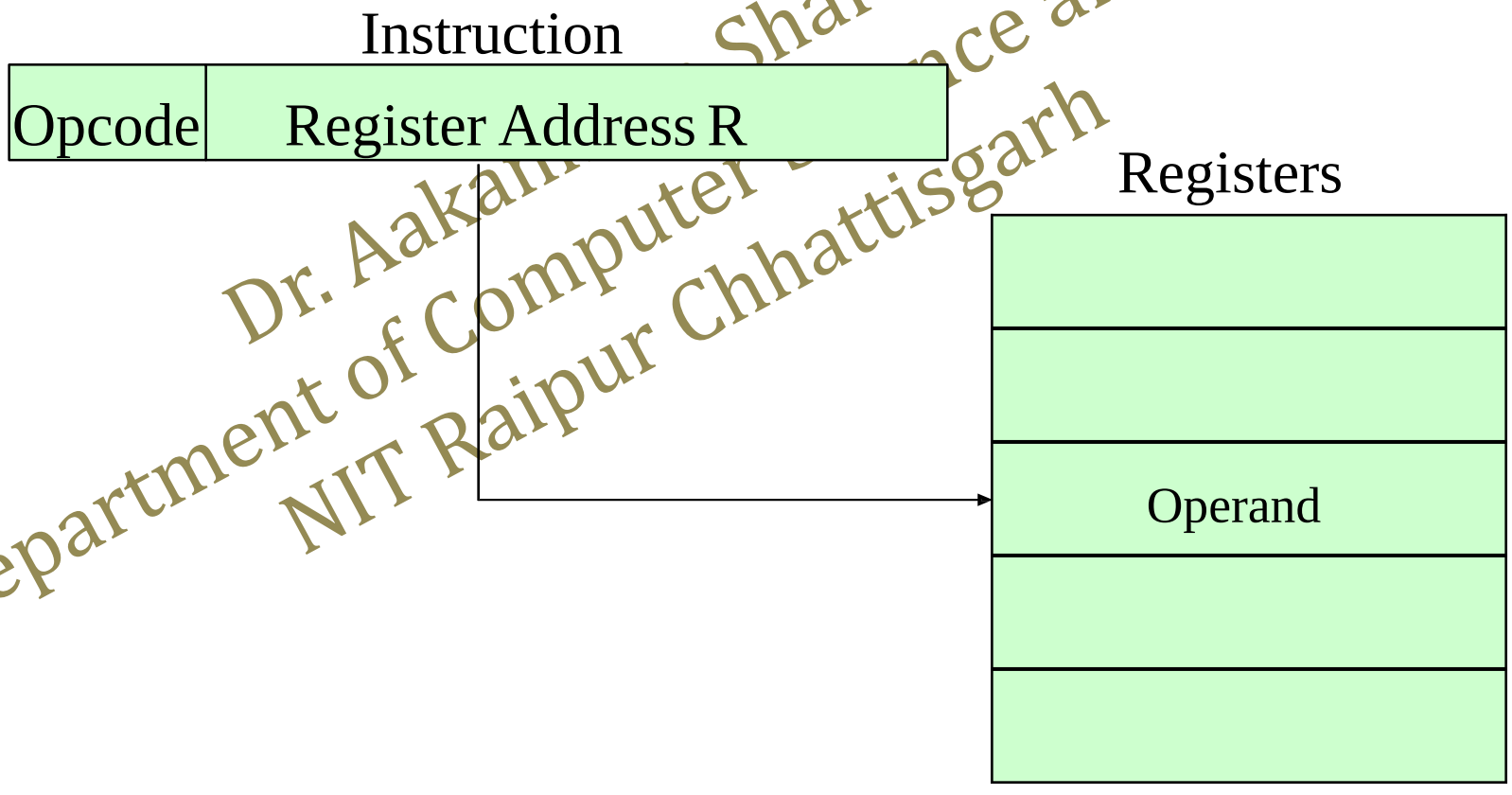
Register Addressing (1)

- Operand is held in register named in address filed
- EA = R
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch

Register Addressing (2)

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing
  - N.B. C programming
    - register int a;
- c.f. Direct addressing

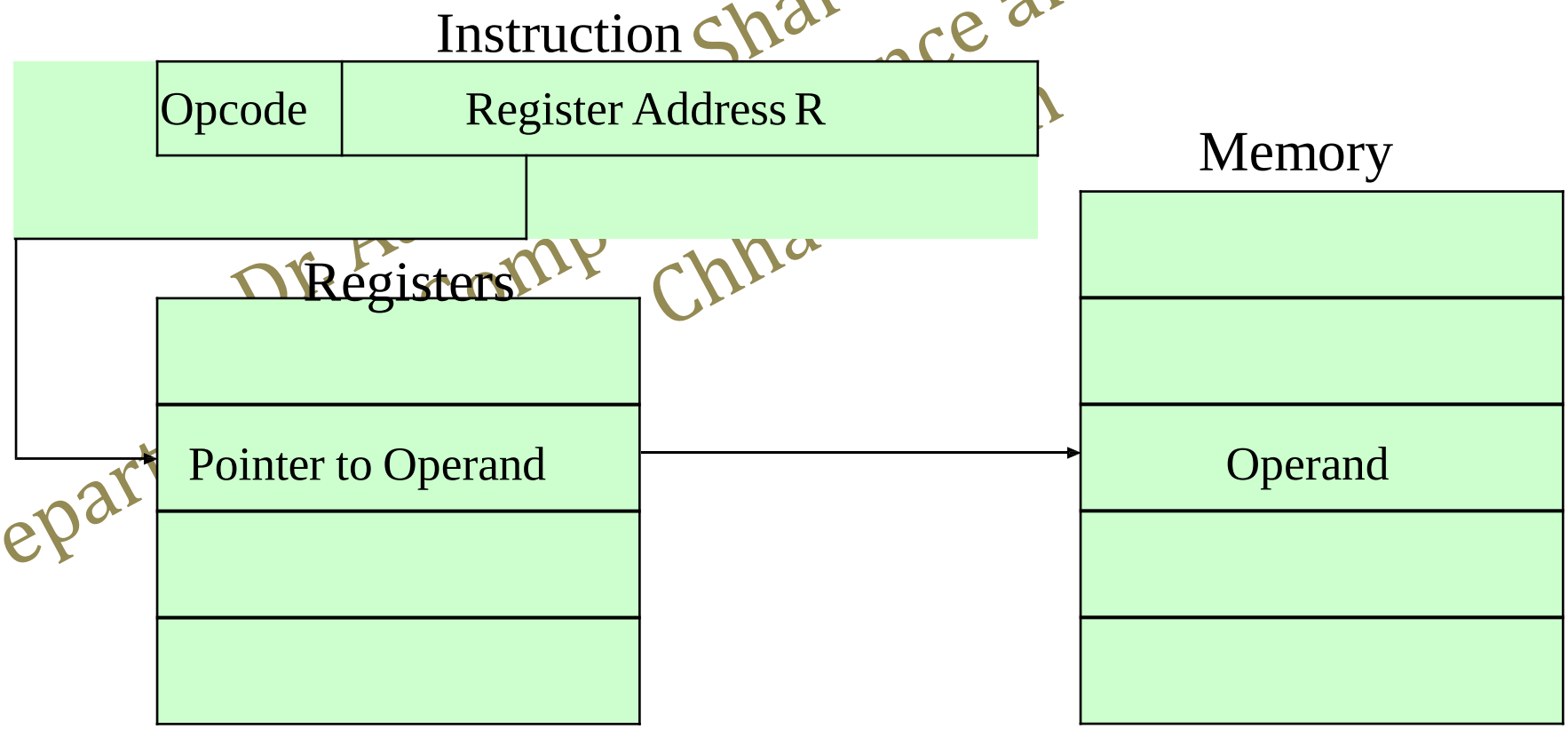
Register Addressing Diagram



Register Indirect Addressing

- C.f. indirect addressing
- EA = (R)
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing

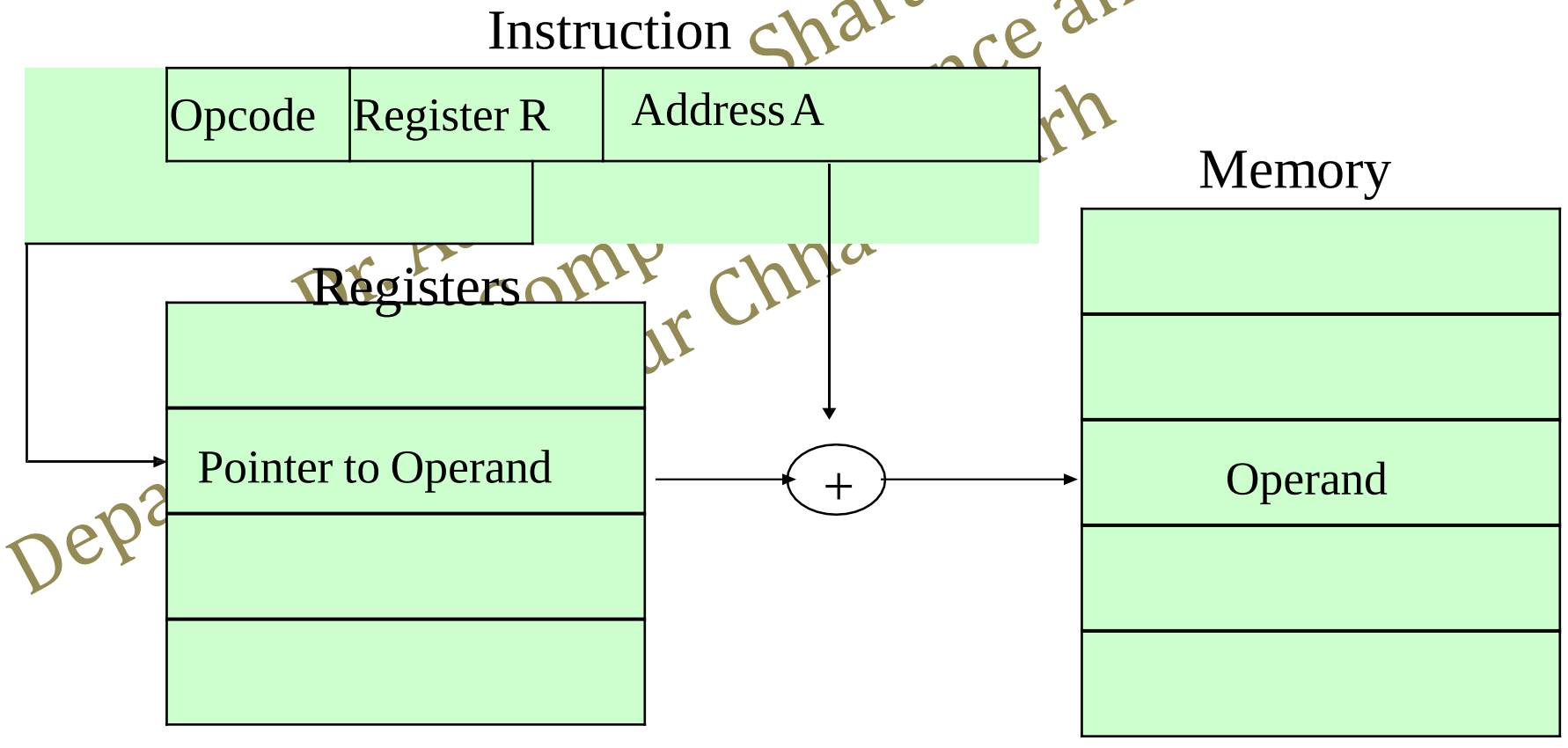
Register Indirect Addressing Diagram



Displacement Addressing

- EA = A + (R)
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa

Displacement Addressing Diagram



Relative Addressing

- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- i.e. get operand from A cells from current location pointed to by PC
- c.f locality of reference & cache usage

Base-Register Addressing

- A holds displacement
- R holds pointer to base address
- R may be explicit or implicit
- e.g. segment registers in 80x86

Indexed Addressing

- A = base
- R = displacement
- EA = A + R
- Good for accessing arrays
  - EA = A + R
  - R++

Combinations

- Postindex
- EA = (A) + (R)

- Preindex
- EA = (A+(R))

- (Draw the diagrams)

Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
  - ADD
    Pop top two items from stack and add

Multiplication Algorithm Hardware and Flowchart

Multiplication

Multiplication (often denoted by x) is the mathematical operation of scaling one number by another. It is a basic arithmetic operation.

Example: 3 x 4= 3+3+3+3=12

5 x 3 ½ =5+5+5+(half of 5)= 17.5

The basic idea of multiplication is repeated addition.

Multiply

❖Paper and pencil example (unsigned):

Multiplicand 1000

Multiplier 1001

1000 **Place a copy**

0000 **Place 0**

0000

Product 1000

01001000

m bits x n bits = m+n bit product

Binary makes it easy:

(0 x multiplicand)

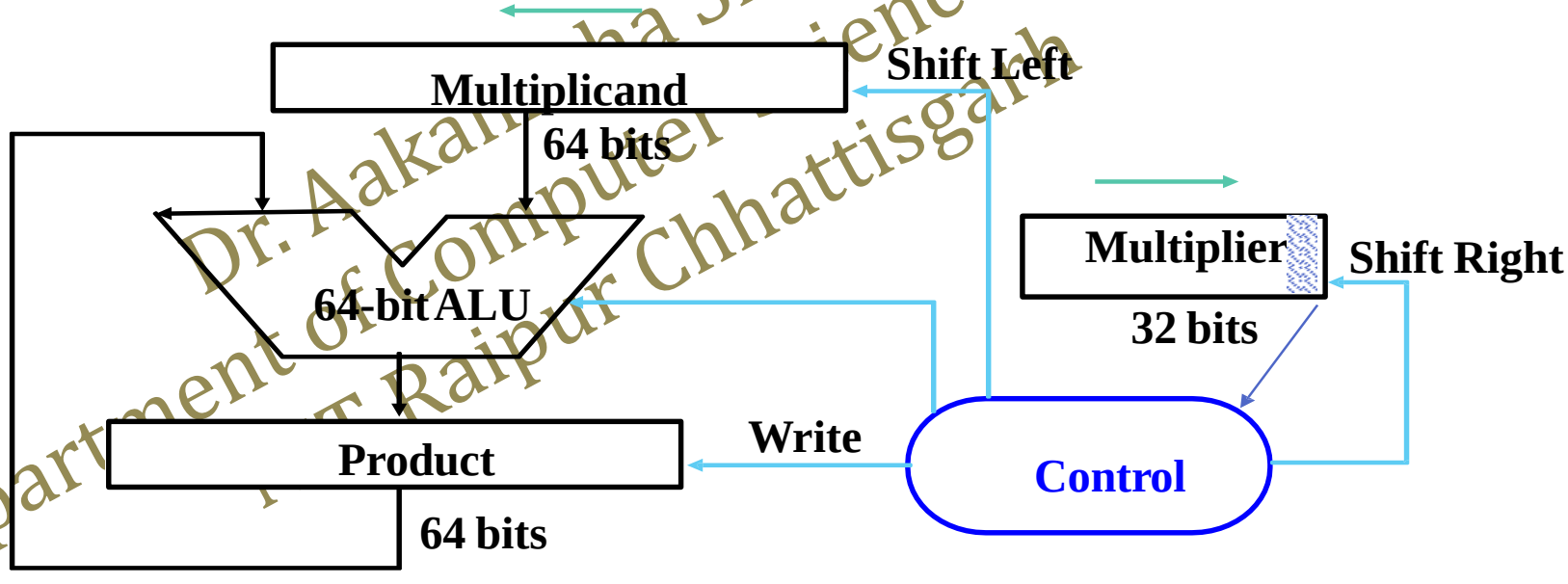
0 => place 0

(1 x multiplicand)

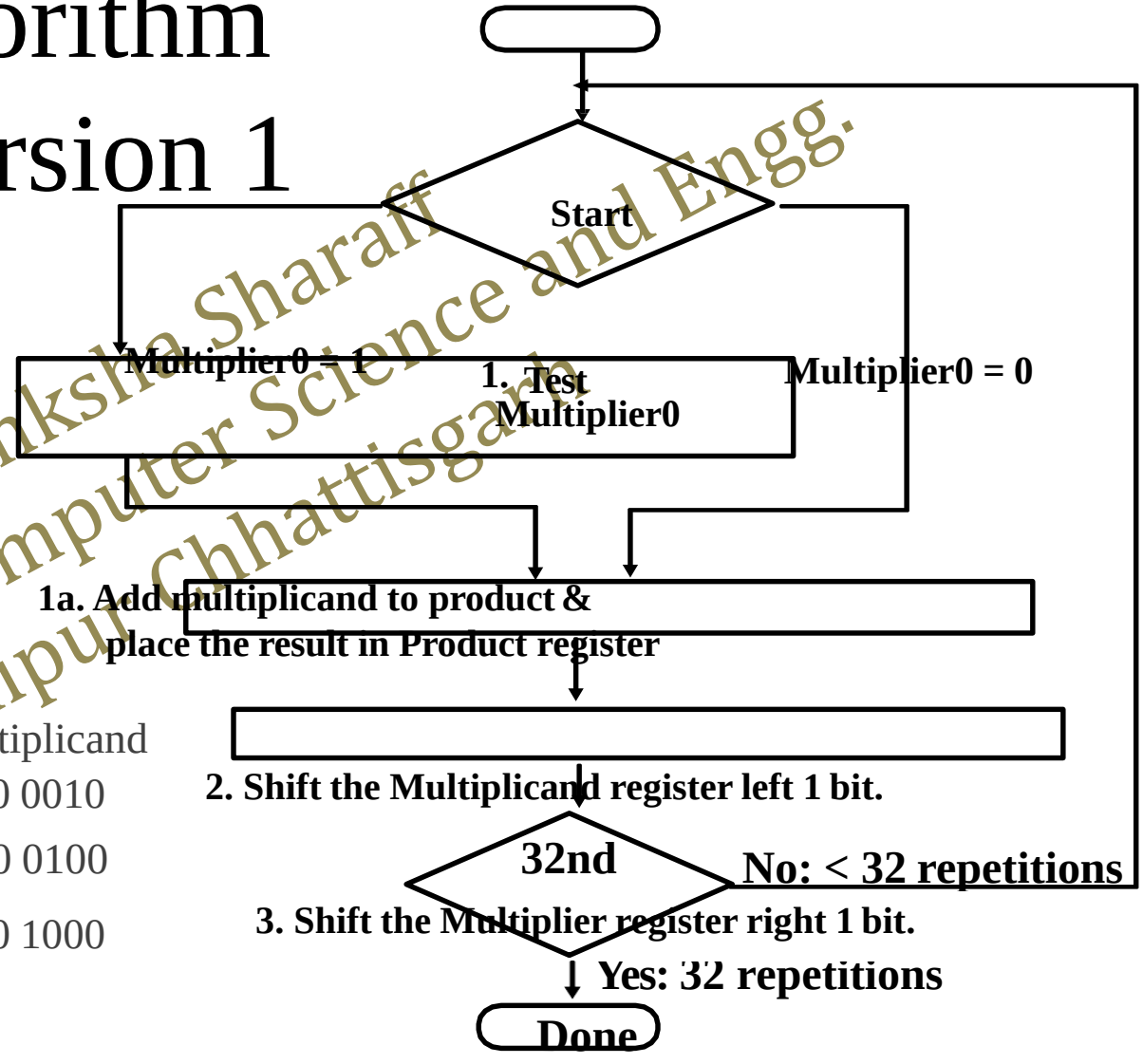
1 => place a copy

Unsigned Shift-add Multiplier

64-bit Multiplicand reg, 64-bit ALU, 64-bit Productreg, 32-bit multiplier reg



Multiply Algorithm Flowchart Version 1

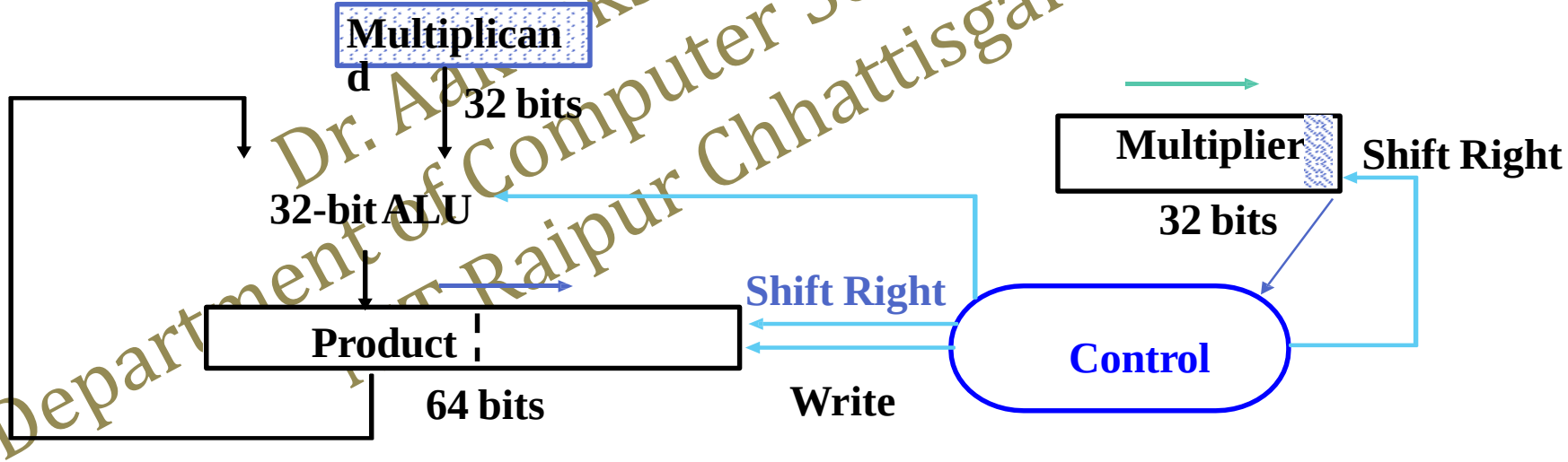


Product	Multiplier	Multiplicand
0000 0000	0011	0000 0010
0000 0010	0001	0000 0100
0000 0110	0000	0000 1000
0000 0110		

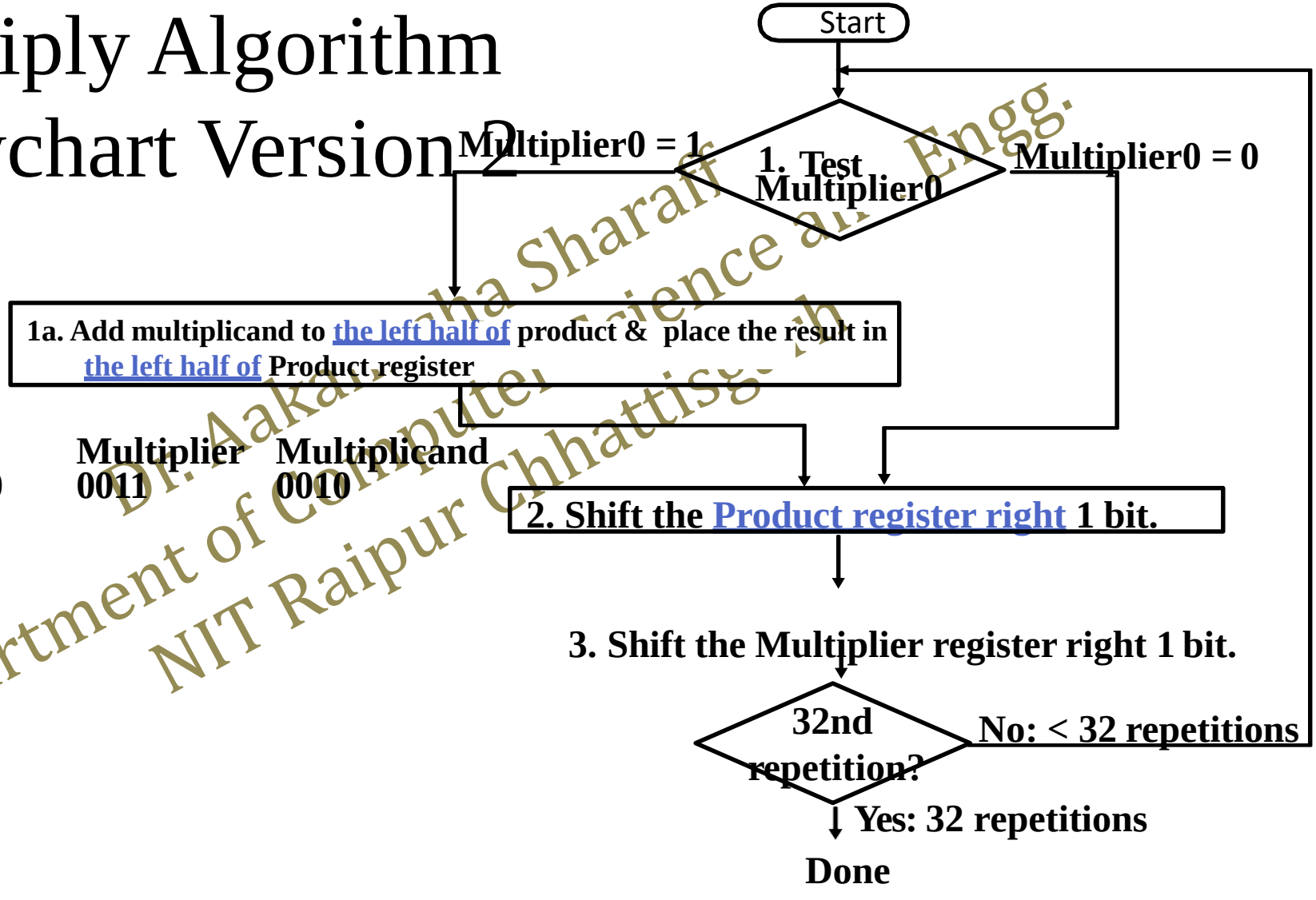
Multiply Hardware Version

Half of the shifted multiplicand register contains 0. Why not shift the product right?

32-bit Multiplicand reg, 32-bit ALU, 64-bit Product reg, 32-bit Multiplier reg

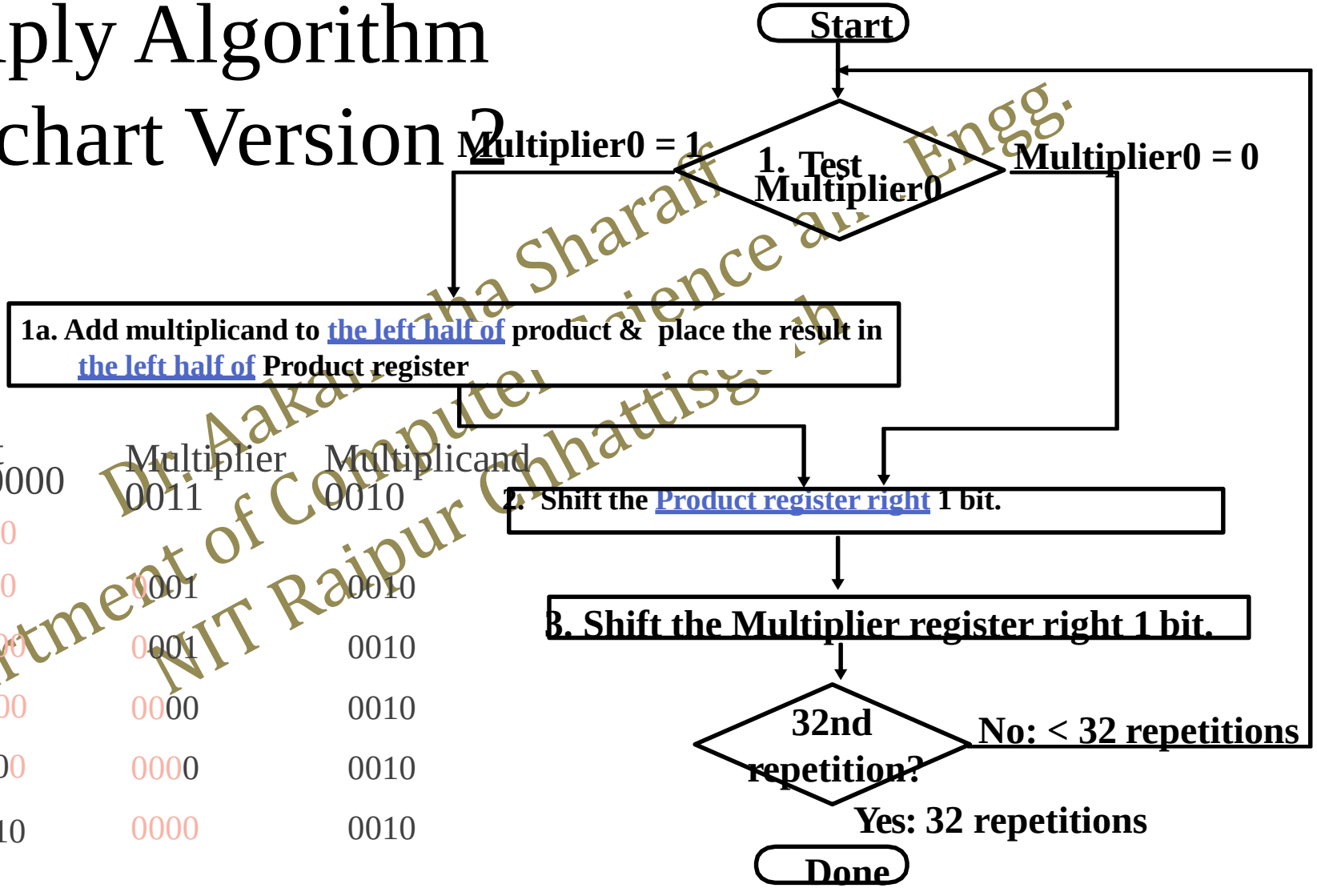


Multiply Algorithm Flowchart Version 2



- **Product** **Multiplier** **Multiplicand**
  0000 0000 **0011** **0010**

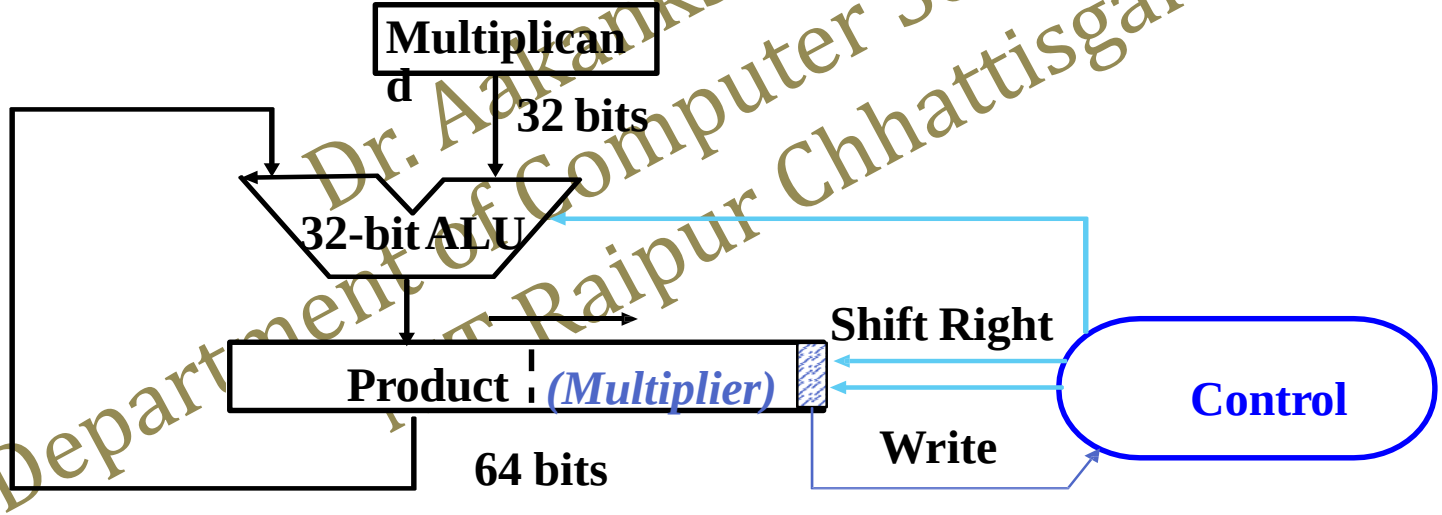
Multiply Algorithm Flowchart Version 2



Product	Multiplier	Multiplicand
0000 0000	0011	0010
0010 0000		
0001 0000	0001	0010
0011 0000	0001	0010
0001 1000	0000	0010
0000 1100	0000	0010
0000 0110	0000	0010

Multiply Algorithm Version 3

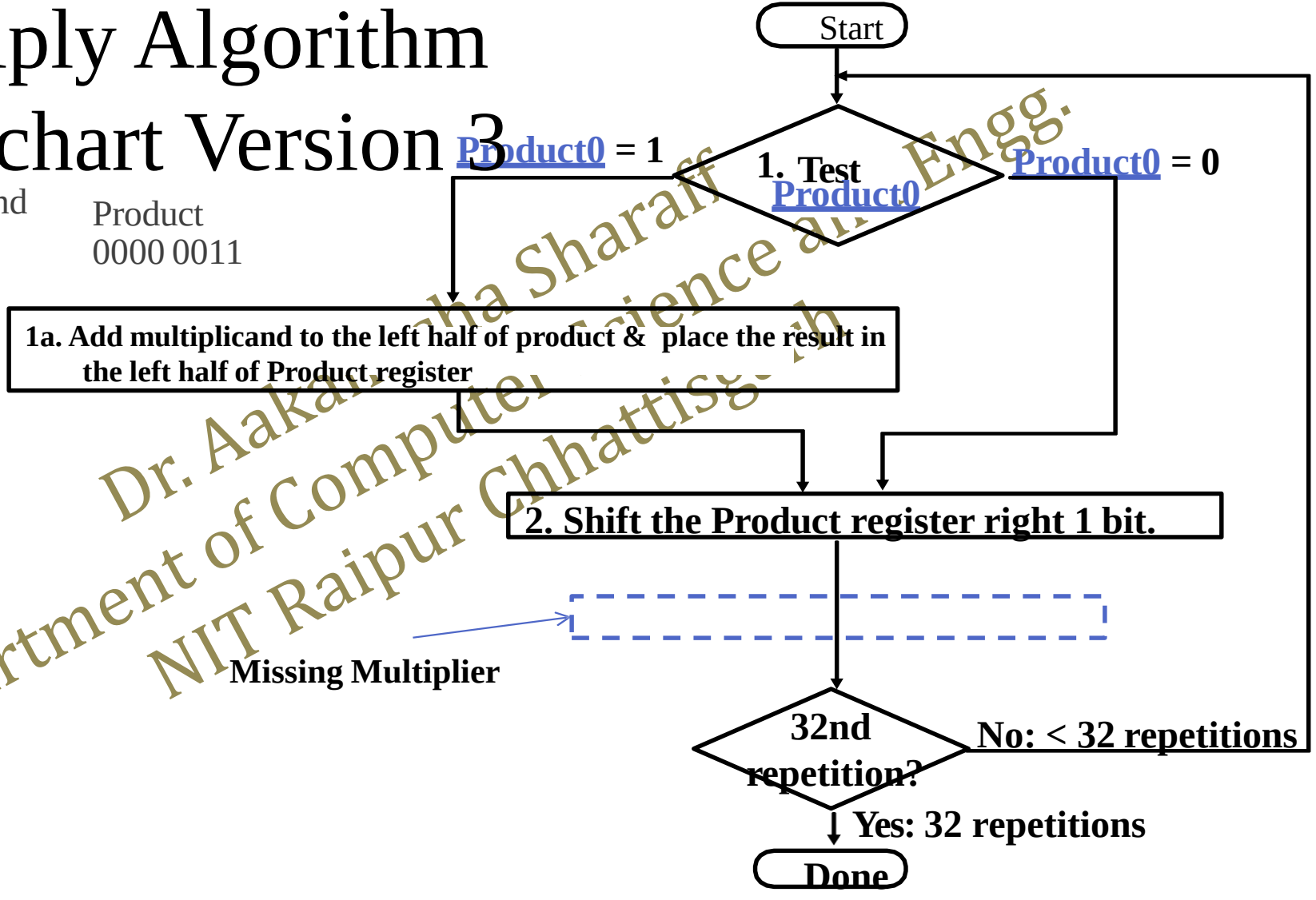
- Combine Multiplier register and Product register
- 32-bit Multiplicand reg, 32 -bit ALU, 64-bit Product reg, (0-bit Multiplier reg)



Multiply Algorithm Flowchart Version 3

Multiplicand
0010

Product
0000 0011



		Mcand: 0010	P: 0000 0011
1	P=P+Mcand	Mcand: 0010	P: 0010 0011
	Shr P	Mcand: 0010	P: 0001 0001
2	P=P+Mcand	Mcand: 0010	P: 0011 0001
	Shr P	Mcand: 0010	P: 0001 1000
3	0=>nop	Mcand: 0010	P: 0001 1000
	Shr P	Mcand: 0010	P: 0000 1100
4	0=>nop	Mcand: 0010	P: 0000 1100
	Shr P	Mcand: 0010	P: 0000 0110

Booth Multiplication Algorithm

- It reduces the number of additions.
- Negative multiplier and positive multiplier are treated same.

Booth Multiplication Algorithm

Booth algorithm gives a procedure for multiplying binary integers in signed –2's complement representation.

Example, 2 x (-4)

0010 * 1100

Step 1: Making the Booth table

1. From the two numbers, pick the number with the smallest difference between a series of consecutive numbers, and make it a multiplier.

i.e., 0010 -- From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change ,so there are two changes on this one

1100 -- From 1 to 1 no change, 1 to 0 one change, 0 to 0 no change, so there is only one change on this one.

Therefore, multiplication of 2 x (– 4), where 2^{ten} (0010^{two}) is the multiplicand and $(-4)^{ten}$ (1100^{two}) is the multiplier.

Booth Multiplication Algorithm

2. Let X = 1100 (multiplier)
   Let Y = 0010 (multiplicand)
   Take the 2's complement of Y and call it –Y
   –Y = 1110
3. Load the X value in the table.
4. Load 0 for X-1 value it should be the previous first least significant bit of X

5.Load 0 in U and V rows which will have the product of X and Y at the end of operation.

6.Make four rows for each cycle; this is because we are multiplying four bits numbers.

U	V	X	X-1	
0000	0000	1100	0	Load the value
				1st cycle
				2nd cycle
				3rd Cycle
				4th Cycle

Booth Multiplication Algorithm

Step 2: Booth Algorithm

Look at the first least significant bits of the multiplier "X", and the previous least significant bits of the multiplier "X - 1".

a.

1	0	Shift only
2	1	Shift only.
3	1	Add Y to U, and shift
4	0	Subtract Y from U, and shift or add (-Y) to U and shift

b. Take U & V together and shift arithmetic right shift which preserves the sign bit of 2's complement number. Thus a positive number remains positive, and a negative number remains negative.

c. Shift X circular right shift because this will prevent us from using two registers for
the X value.

Booth Multiplication Algorithm

Repeat the same steps until the four cycles are completed.

U	V	X	X-1
0000	0000	1100	0
0000	0000	0110	0
0000	**0000**	**0011**	**0**

← Shift only

U	V	X	X-1
0000	0000	1100	0
0000	0000	0110	0
0000	0000	0011	0
1110	**0000**	**0011**	**0**
1111	**0000**	**1001**	**1**

← Add –Y (0000 + 1110 = 1110)

← Shift

Booth Multiplication Algorithm

U	V	X	X-1
0000	0000	*1100*	0
0000	0000	0110	0
0000	0000	0011	0
1110	0000	0011	0
1111	0000	1001	1
1111	**1000**	***1100***	**1**

Shift only

We have finished four cycles, so the answer is shown, in the last rows of U and V which is: 11111000

Now to easily check our calculations, we take the original question, 2 x -4. This becomes -8. -8 is the two's compliment of 8. 8 in eight bit binary is 00001000.

Taking the two's compliment by the method previously described, we get the result 11111000, which is exactly the same as our Booths algorithm answer.

Booth Recoding

1,1	→ 0
1,0	→ +1
0,1	→ −1
0,0	→ 0

$+1000-1$

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$16+0+0+0-1=15$

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

0 1 1 1 1

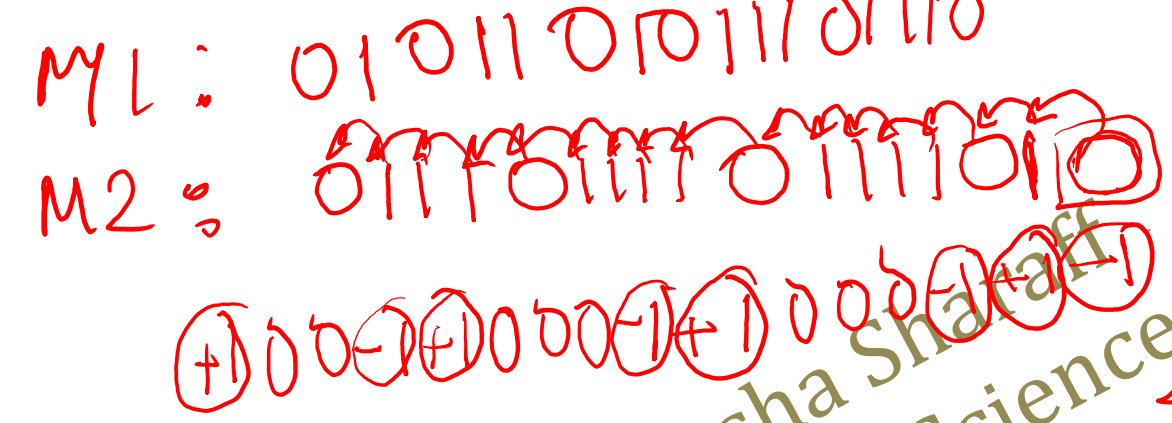
$0+8+4+2+1=15$

→ 1 0 1 1 0 1 1

$\times$ 0 1 1 0 1

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

1 1 0 1 1 0 1 1



Q1: M1: 0 1 1 1 1 0

M2: 0 1 0 0 1 0

4-A, 4-S

Thank you ... ☺