5. The Role of Binary Numbers in Computer Architecture

Binary Number system is a base 2 which uses only two digits 0 and 1. In this system we represent sequence of these two digits as $(1001)_2$.

Computers use the binary system because it directly corresponds to the two states of electronic circuits:

- 0 represents low voltage (Off state)
- 1 represents high voltage (On state)

In digital circuits, these states od 0s and 1s are managed using transistors, where:

- 1 indicates the presence of an electrical current (**high state**).
- 0 indicates the absence of an electrical current (**low state**).

In computing, data is represented in units of 8 bits, also known as a byte. A byte can store values that range from 0 to 255 (in decimal), allowing it to represent characters, numbers, and symbols in various encoding standards, such as ASCII (American Standard Code for Information Interchange) like:

Binary	**ASCII(characters)**
1000001	A
1111010	z

This binary encoding enables computers to store and process text, numerical values, and symbols efficiently.

Thus, in computer architecture, the binary number system is fundamental for representing characters, performing data operations, and encoding information across different computational processes.

Printing ASCII control characters

BINARY	DECIMAL	HEXADECIMAL	SYMBOL
010 0000	32	20	SPACE
010 0001	33	21	!
010 0010	34	22	"
010 0011	35	23	#
010 0100	36	24	$
010 0101	37	25	%
010 0110	38	26	&
010 0111	39	27	'
010 1000	40	28	(
010 1001	41	29	)
010 1010	42	2A	*
010 1011	43	2B	+
010 1100	44	2C	,
010 1101	45	2D	-
010 1110	46	2E	.
010 1111	47	2F	/
011 0000	48	30	0
011 0001	49	31	1
011 0010	50	32	2
011 0011	51	33	3
011 0100	52	34	4
011 0101	53	35	5
011 0110	54	36	6
011 0111	55	37	7
011 1000	56	38	8
011 1001	57	39	9
011 1010	58	3A	:
011 1011	59	3B	;
011 1100	60	3C	<
011 1101	61	3D	=
011 1110	62	3E	>
011 1111	63	3F	?
100 0000	64	40	@
100 0001	65	41	A
100 0010	66	42	B
100 0011	67	43	C
100 0100	68	44	D
100 0101	69	45	E
100 0110	70	46	F
100 0111	71	47	G
100 1000	72	48	H
100 1001	73	49	I
100 1010	74	4A	J
100 1011	75	4B	K
100 1100	76	4C	L
100 1101	77	4D	M
100 1110	78	4E	N
100 1111	79	4F	O

BINARY	DECIMAL	HEXADECIMAL	SYMBOL
101 0000	80	50	P
101 0001	81	51	Q
101 0010	82	52	R
101 0011	83	53	S
101 0100	84	54	T
101 0101	85	55	U
101 0110	86	56	V
101 0111	87	57	W
101 1000	88	58	X
101 1001	89	59	Y
101 1010	90	5A	Z
101 1011	91	5B	[
101 1100	92	5C	\
101 1101	93	5D	]
101 1110	94	5E	^
101 1111	95	5F	_
110 0000	96	60	`
110 0001	97	61	a
110 0010	98	62	b
110 0011	99	63	c
110 0100	100	64	d
110 0101	101	65	e
110 0110	102	66	f
110 0111	103	67	g
110 1000	104	68	h
110 1001	105	69	i
110 1010	106	6A	j
110 1011	107	6B	k
110 1100	108	6C	l
110 1101	109	6D	m
110 1110	110	6E	n
110 1111	111	6F	o
111 0000	112	70	p
111 0001	113	71	q
111 0010	114	72	r
111 0011	115	73	s
111 0100	116	74	t
111 0101	117	75	u
111 0110	118	76	v
111 0111	119	77	w
111 1000	120	78	x
111 1001	121	79	y
111 1010	122	7A	z
111 1011	123	7B	{
111 1100	124	7C	|
111 1101	125	7D	}
111 1110	126	7E	~
111 1111	127	7F	DEL

The Role of binary in computer architectures

Binary can be used in representing and storing data, performing calculations, retrieving data and controlling and organizing tasks. Since there is only two state 0 as low voltage “off” and 1 as high voltage “on” state, we can use this feature of binary to perform calculations using components like transiters and circuits.

We can also represent signed numbers using representations like sign and magnitude representation, 1’s compliment and 2’s compliment. Of all, Two’s complement is commonly favored in contemporary systems due to its advantages over other methods. It simplifies arithmetic operations and eliminates the issue of having two representations for zero.

Binary bit Operations:

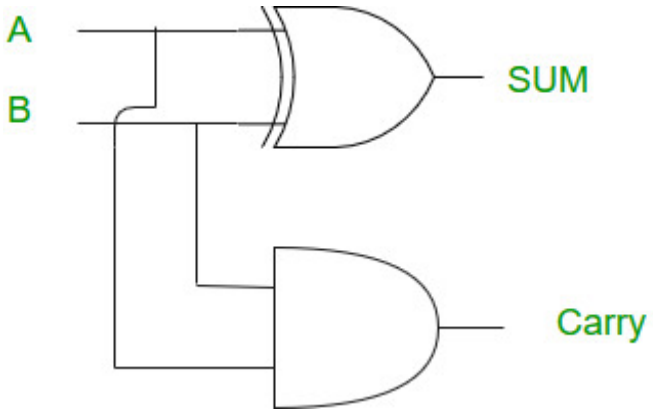
- **Addition:**
  - 1 + 1 = 0, 1 carried over
  - 1 + 0 = 1,
  - 0 + 0 = 0

Example:

```
  1101  (13)
+ 1011  (11)
------------------
1|1000  (24)
```

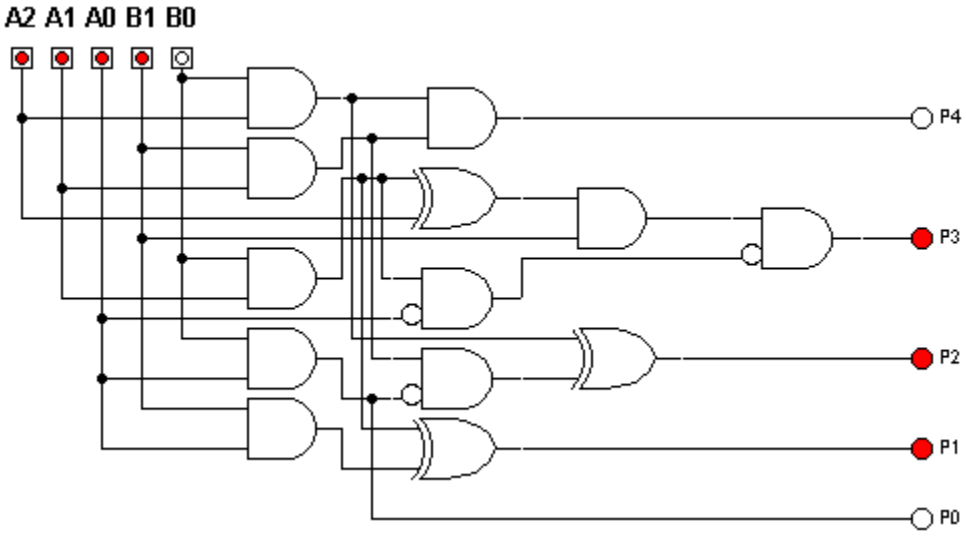
- **Ones column:** 1 + 1 = 10 (write down 0, carry-over 1)
- **Twos column:** 0 + 1 + 1 = 10 (write down 0, carry-over 1)
- **Fours column:** 1 + 0 + 1 = 10 (write down 0, carry-over 1)
- **Eights column:** 1 + 1 + 1 = 11 (write down 1, carry-over 1)
- **Sixteenths column:** 1 (write down 1), exceeds the current bit size.

Addition can be performed in computing with the help of half adders and full adders.



- **Multiplication:**
  - 1 x 1 = 1,
  - 1 x 0 = 0,
  - 0 x 0 = 0

```
   1 0 1 1.0 1
       1 1 0.1
  ---------------
     1 0 1 1 0 1
   0 0 0 0 0 0
  ---------------
   0 1 0 1 1 0 1  ..... First Intermediate sum
  1 0 1 1 0 1
  ---------------
   1 1 1 0 0 0 0 1 ..... Second Intermediate Sum
 1 0 1 1 0 1
 ---------------
1 0 0 1 0 0 1.0 0 1 ..... Final Sum
```



- **Subtraction**
  1. Find the two's complement of the subtrahend
     - **Invert all bits** (one's complement).
     - **Add 1** to the inverted bits
  2. Add the result to the minuend

3. If there is a carry-out, discard it.
4. If no carry-out occurs, the result is negative (already in two's complement form)

Example: 7 - 5 in Binary in following steps:

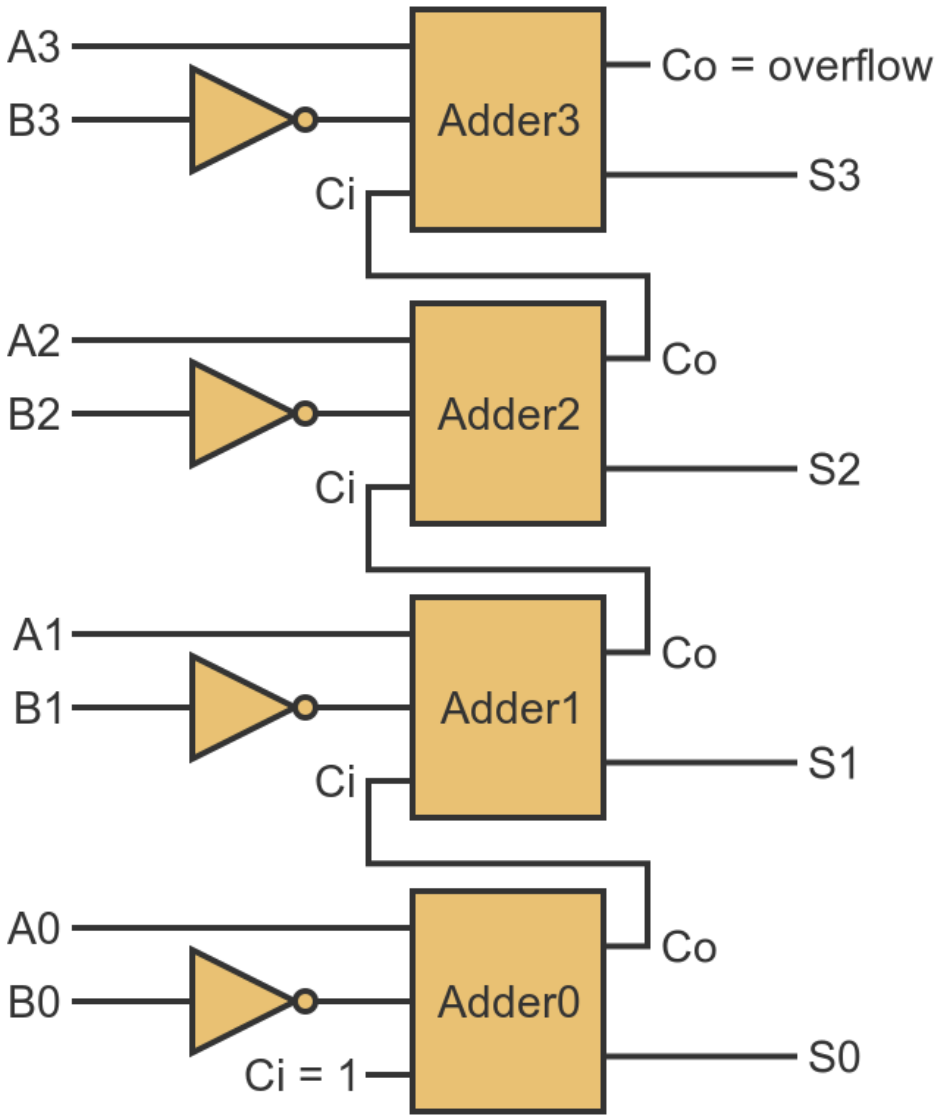
1. Convert Numbers to Binary
   7 → 0111
   5 → 0101
2. Find Two's Complement of 5

   Invert all bits of 5: 0101 → 1010
   Add 1: 1010 + 1 = 1011

3. Add to 7

```
  0111 (7)
+ 1011 (-5 in two's complement)
------------
1|0010 (Ignore carry-out → Final answer: 0010)
```

In digital logic, subtraction is possible through addition.



- **Division**

There are various ways of achieving division in digital logics, like:

1. Restoring Division Algorithm:
   - Involves repeated subtraction of the divisor from the dividend.
   - Restores the previous value if the result is negative.
   - Utilizes subtractors and control logic in the Arithmetic Logic Unit (ALU).
2. Non-Restoring Division Algorithm:

- Enhances efficiency by eliminating the restore step.
- Adjusts the quotient based on the sign of the remainder.
- Incorporates adders and control units for sign evaluations.

3. Newton–Raphson Division:
   - An iterative method that approximates the reciprocal of the divisor.
   - Multiplies the reciprocal by the dividend to obtain the quotient.
   - Requires multipliers and adders to perform iterative calculations.
4. Goldschmidt Division:
   - Transforms division into a multiplication problem.
   - Simultaneously scales the dividend and divisor to converge the divisor to 1.
   - Uses parallel multipliers to enhance computation speed.

Example, Divide 13 by 3 in Binary
Dividend (13_{10}) = 1101_2
Divisor (3_{10}) = 11_2
Using **restoring division** algorithm in the following steps:

1. **Initialize:**
   Dividend: 1101_2
   Divisor: 11_2
   Quotient: 0000 (initially zero)
   Remainder: 0000 (initially zero)

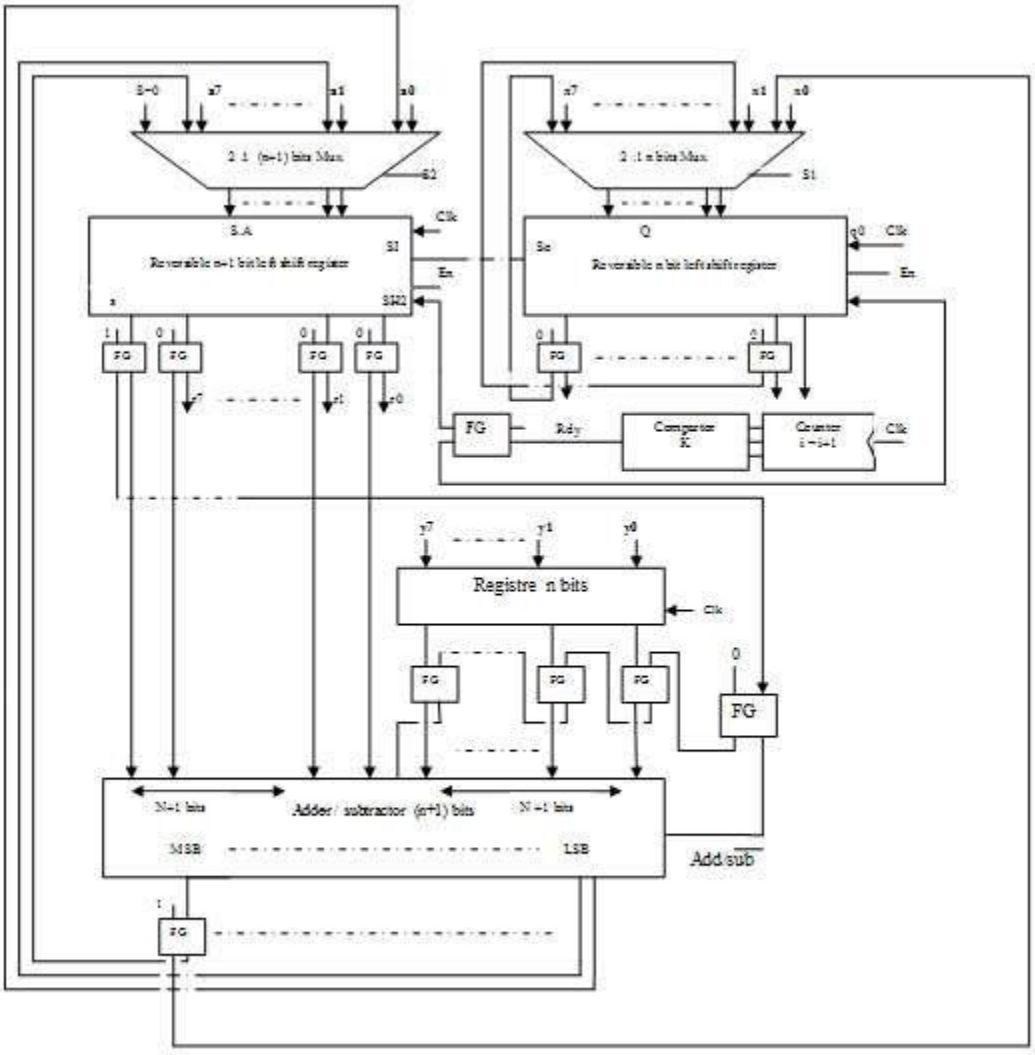
2. **Bring down the first bit of the dividend.**
   Remainder: 110 (first three bits of dividend)
   Perform subtraction: 110 - 11 = 01
   Quotient: 1 (since subtraction was successful, we write 1)

3. **Bring down the next bit of the dividend.**
   Remainder: 010 (bring down next bit, which is 0)
   Perform subtraction: 010 - 11 (this goes negative)
   Restore the remainder: Return to 110 (previous remainder)
   Quotient: 10 (since subtraction went negative, we write 0)

4. **Bring down the final bit of the dividend.**
   Remainder: 110 (bring down the final bit, which is 1)
   Perform subtraction: 110 - 11 = 01
   Quotient: 101 (successful subtraction, so write 1)

5. **Final Result:**
   Quotient = 101_2 (which is 5 in decimal)

Remainder = 1_2 (which is 1 in decimal)



Binary Arithmetic Implemented in Arithmetic Logic Unit (ALU)

ALU is responsible for performing binary arithmetic and logical operation inside of a CPU. It can process addition, subtraction, multiplication, and division using logic gates, adders, and control circuits.

Major Components of the ALU for Binary Arithmetic:

1. Half Adder & Full Adder (For Addition & Subtraction)
   - The Half Adder performs single-bit binary addition using XOR and AND gates.
   - The Full Adder extends this concept to multi-bit addition by handling carry-over bits.
   - Subtraction is done using Two's Complement, so subtraction circuits often use adders with inverters instead of separate subtractors.

2. Two's Complement Circuit (For Subtraction)
   - The ALU converts subtraction into addition by taking the two's complement of the subtrahend and adding it to the minuend.
   - Uses NOT gates (for bit inversion) and increment circuits (to add 1).

3. Shift-and-Add Multiplication (For Multiplication)
   - Multiplication is performed using bitwise shifting and addition.
   - The ALU checks each bit of the multiplier:
     - If 1, it adds the multiplicand.
     - If 0, it only shifts left (like long multiplication).
   - High-performance processors use Booth's Algorithm to optimize this process.

4. Restoring & Non-Restoring Division (For Division)
   - The ALU uses subtraction-based division algorithms like:
     - Restoring Division (restores remainder if negative).
     - Non-Restoring Division (avoids unnecessary restores for efficiency).
   - More advanced processors use Newton–Raphson and Goldschmidt algorithms for fast division.

5. Logic Gates (For Boolean Operations)
   - The ALU also performs logic operations like AND, OR, XOR, and NOT to manipulate binary values.
   - These operations are used for bitwise operations, comparisons, and decision-making in programs.

How ALU Executes Binary Arithmetic Step-by-Step

1. **Addition**: Uses full adders to sum binary numbers, handling carry propagation.
2. **Subtraction**: Uses two's complement to convert subtraction into addition.
3. **Multiplication**: Uses shift-and-add method, optimized with Booth's Algorithm.
4. **Division**: Uses repeated subtraction or advanced division algorithms.
5. **Logic Operations**: Uses AND, OR, XOR, and NOT gates for bitwise computations.