Arithmetic Logic Unit:

i) 10 Instructions : A & B are 32 Bit Binary Numbers

④ → Arithmetic	③ → Logic	③ → Shifting
ADD ⇒ A+B } Binary	XOR → A ^ B	SLL → Shift left logic
SUB ⇒ A-B	AND → A & B	SRL → Shift Right logic
SLT ⇒ Set less than	OR → A | B	SRA → Shift Right Arithmetic
SLTU ⇒ Set less than Unsigned		

ii) SLTU ⇒ Set Less Than Unsigned

A<B ⇒ Output = 1

A>B ⇒ Output = 0

If we do subtraction A<B MSB ⇒ 1 (-ve number)

A>B ⇒ A-B ⇒ +ve → MSB = 0

SLTU ⇒ MSB of Subtraction Operation

Signed ↑ 2's Complement

A ⇒ 100 (+ve) ⇒ 4

B ⇒ 010 (+ve) ⇒ 2

A-B ⇒ A+~B+1 ⇒ 100 + 101 + 1

⇒ 1001 + 1 (Carry)

⇒ 010 ⇒ 2

4-2 = 2 → unsigned Subtraction

A+~B+1

Result ⇒ MSB 32nd Bit. 0 → +ve number, 1 → -ve number

iii) SLT logic ⇒ Set less than for signed numbers

A ⇒ Operand1 B ⇒ Operand2

MSB of these numbers are sign Bit

Operand1[31]	Operand2[31]	Result	
0 → 31 ↑ Bit ↑ 32nd Bit	0	0 ⇒ 0,0	Both are +ve (SLTU)
0	1 ⇒ 0,1	A-B ⇒ +ve-(-ve) ⇒ +ve	0
1	0 ⇒ 1,0	A-B ⇒ -ve-(+ve) ⇒ -ve	1
1	1 ⇒ 1,1	A-B ⇒ (-ve)-(-ve) ⇒ +ve-ve	B-A ⇒ SLTU

A-B ⇒ +ve-(-ve) ⇒ +ve+ve = B-A ~ B<A ⇒ 1, B>A ⇒ 0

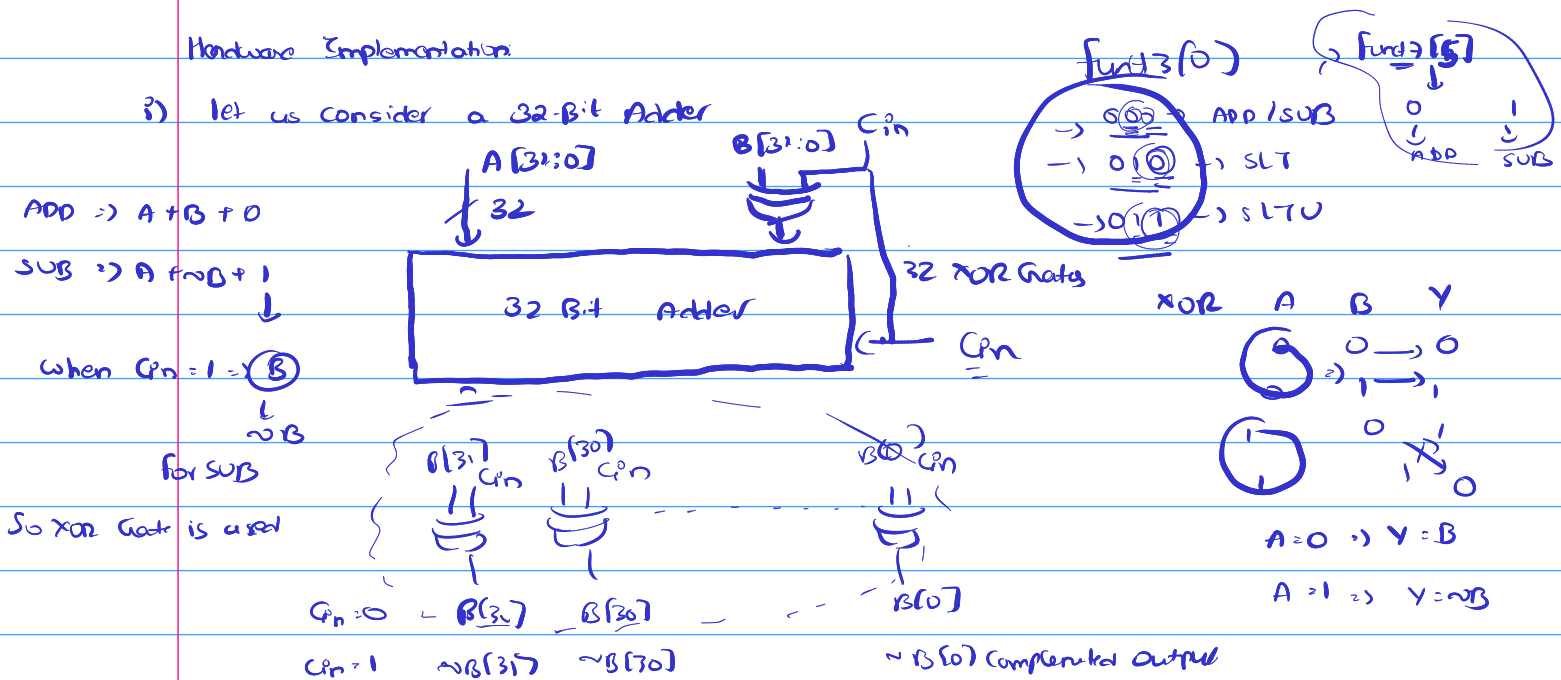
A<B ⇒ 1, A>B ⇒ 0 = NSLTU

A<B ⇒ 1, A>B ⇒ 0

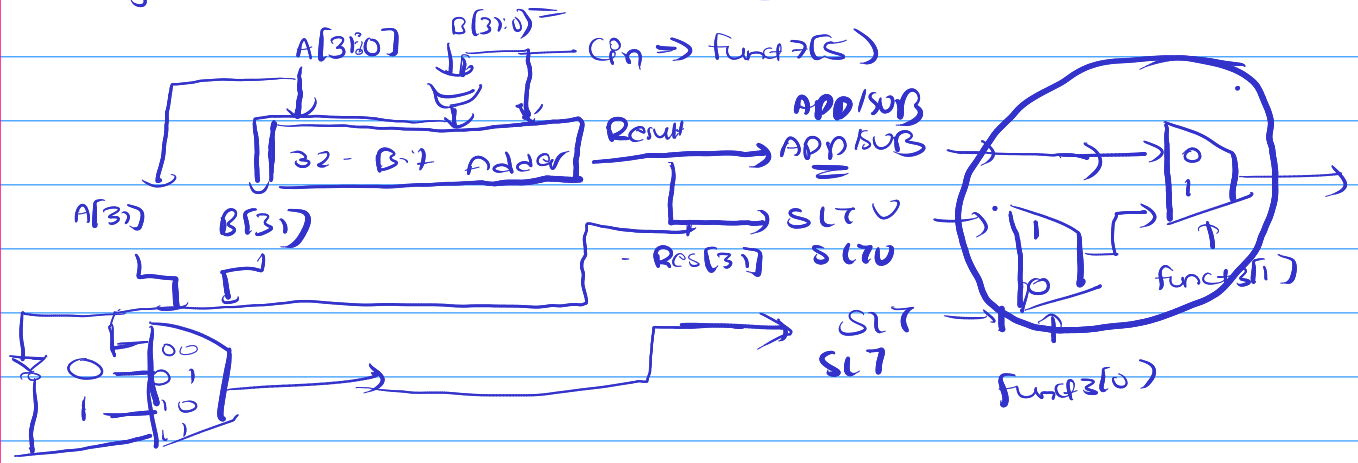
Now, lets implement hardware on these 4 arithmetic instructions

Hardware Implementation:

i) let us consider a 32-Bit Adder



for SLT you need to extract MSB(31) => 1 (A<B) 0 A>B

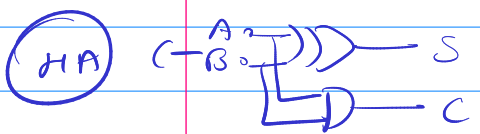


Hardware Implementation for Arithmetic Operations in ALU

2 Bit Adder

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

↓ XOR ↓ AND



3 Bit Adder

A	B	Cin	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

32-Bit Add

When number of 1's is odd output is 1

↳ XOR Gate

A⊕B⊕C = S

From (i) & (ii)

when A=1, B=1 Cin output 1 ← don't care

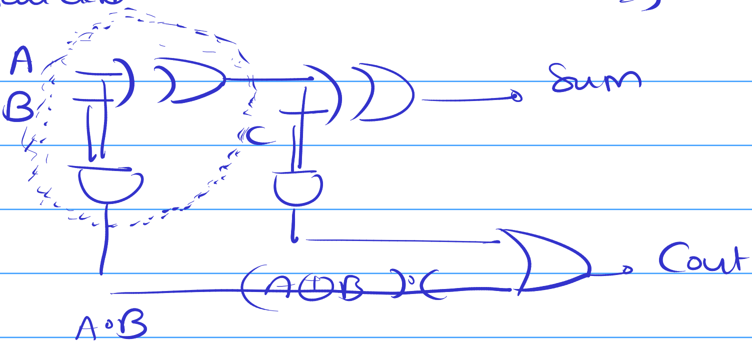
A=B when A & B are 01 & 10 if Cin is 1 output 1 if Cin is 0 output

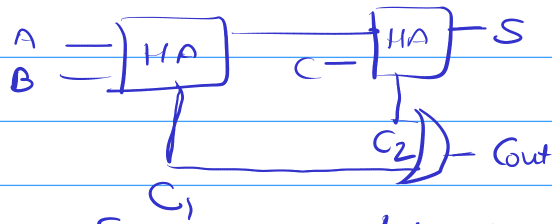
Sum = A⊕B⊕C

Cout = (A·B) + (A⊕B)·C

C = (A⊕B)·C

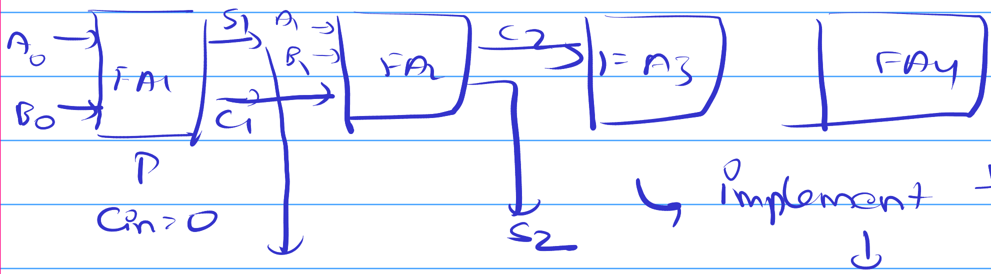
Full adder





Full adder using half adder

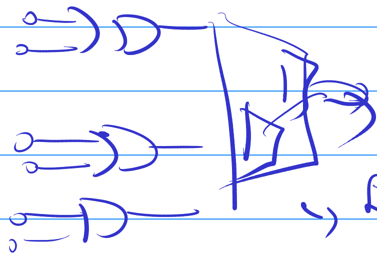
we need to implement a 4 bit adder



↳ implement this for 32 Bits
↓
Ripple Carry Adder Circuit

① 2 4
3 9
———
3

XOR
OR
AND



↳ using ② 2x1 muxes
↳ Logical Operations

Shifting Operations:

Number Range :

A B C D

A0 ——→ a
B0 ——→ b
C0 ——→ c } one-one mapping
D0 ——→ d ↳ One Hot Encoding

0 0 | 00 ——→ a
0 1 | 01 ——→ b
1 0 | 10 ——→ c } Binary
1 1 | 11 ——→ d

n ⇒ $\log_2^n$ ⇒ for rep

3 bit value ⇒ ⑧ (000 - 111) (0-7)
5 Bit value ⇒ ㉜ (00000 - 11111) (0-31)
} → Important Point

Shifting Operation

32-Bit number (Not Circular shifting) → DSA Padke bakwas mat karo melose

32 Times shift → Output = Zero
⇓
5 Bits

i) SLL: Shift left logic Op1 & Op2 Both are 32 Bit

32 Bit (Op1 << Op2 ⇒) 5 bits are enough rest are not needed
Operand — number of times we need to shift

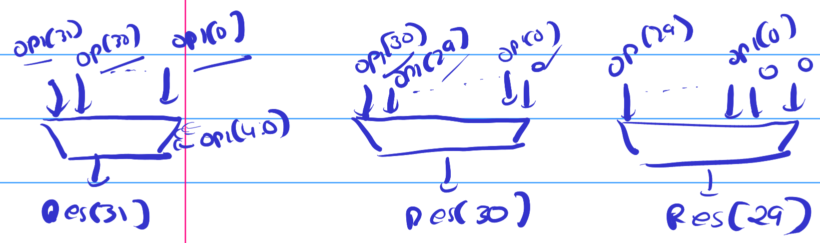
ii) SRL: (Op1) >> (Op2) → 32 Bit 5 Bit (27 Bits are truncated or Don't care)
Op1: 32 Bit, Op2: 5 Bit

iii) SRA: Shift Right arithmetic
Shifting same as the SRL logic but the MSB (Most significant Bit) doesn't change

1011 ⇒ 2 time shift ⇒ SRL 0010 ⇒ SRA 1010
MSB ⇒ 1 → MSB Doesn't change but involves in shifting

How to implement a hardware level logic for this circuit.

i) SLL



→ 32 Multiplexers are Required

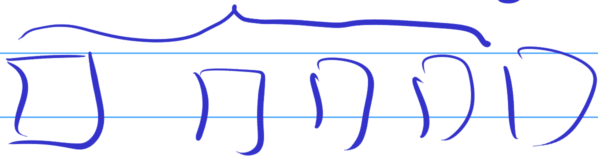
This is Required for single clock cycle execution

32×5 Mux
Select Line
Input

op1[31] .... op1[0] ←
op1[30] .... op1[0] 0 ←
op1(31)

00000
00001

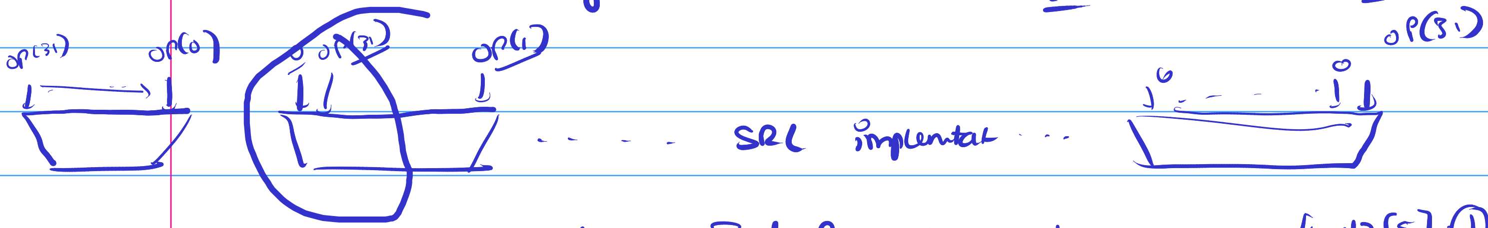
32 Flipflops ⇒ 32 Clock cycles



Short use 32 Bit Shift
↳ it is a sequential circuit
↳ Clock ⇒ flipflops ⇒ Instruction Execute in Clock Cycle

SRL + SRA + SLL -> All in same hardware

SRL & SRA should be together Both have same funct3 funct7[5]



for SRA only MSB doesn't change But rest all remain same funct7[5] 1 ↓ SRA

SRL ← 0 ← Sign Bit

MSB = 0 funct7[5] = 0

same as SRL ↓ SRL

OP	A	B	Y
	0	0	0
	0	1	1
	1	0	1
	1	1	1

A = 0 => B

A = 1 => 1

0 0 1 => 0 0 1 } SRA

1 1 1 => 1 1 1

0 1 1 => 0 0 1 } SRL

1 1 1 => 0 1 1

(MSB == 1) && (funct7[5] == 1)

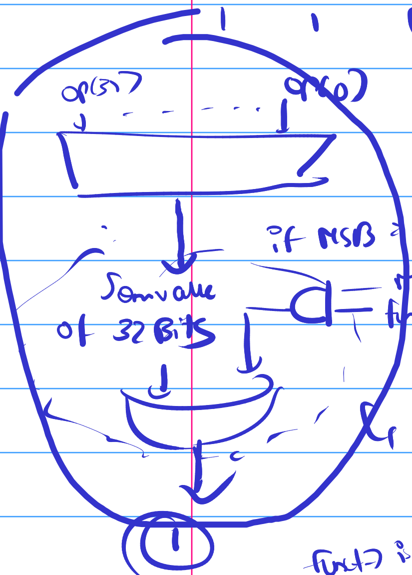
⇓ SRA MSB equals to

Operand -> 32 Bit (MSB = 1')

B ↓ MSB -> OP(31) ↓ Result

Funct7(5)

if it is 1 By dry chuve



if MSB == 1 & funct7(5) == 1

(31 muxes same as SRL)

SRL + SRA

Funct7(5) == 0 AND == 0 OR

funct7(5) == 1 -> MSB = 0 if 0 -> 2nd input

MSB = 1 AND == 0

OR = 1

funct7 is used when same hardware for Both instructions

Cin funct7(5) funct7[5] funct3(1)

funct3(1) SUB

SLT -> 1

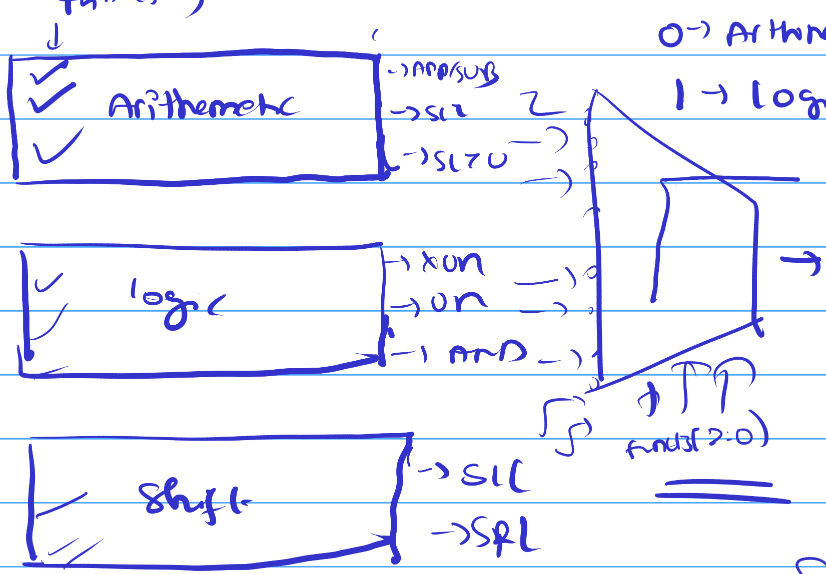
SLTU

XOR

ADD/SUB 001

000 SLT

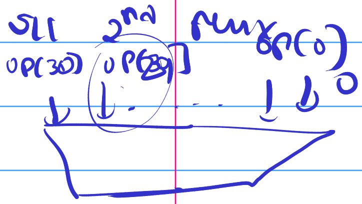
SLTU 010

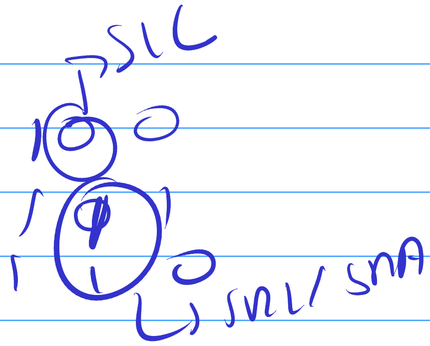


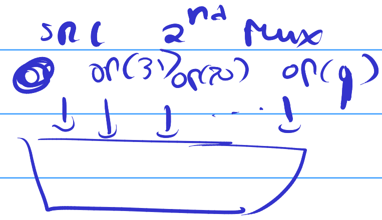
0 -> Arithmetic + SLL

1 -> logic + SRA

logic Enable En b







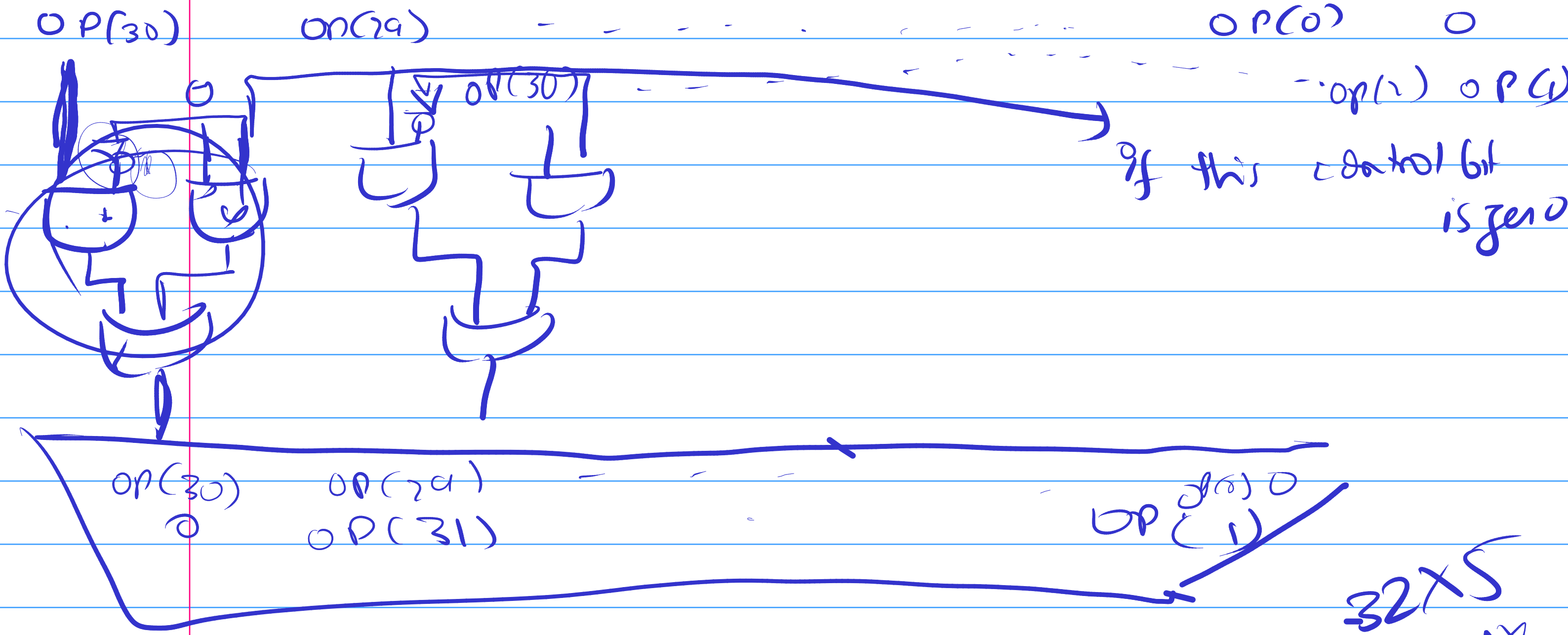
op(31) op(30) ··· op(0) ←

op(30) op(29) ··· op(0) 0

2nd Mux

op(31) op(30) ·· op(0)

0 op(31) ···· op(1)



if this control bit is zero

0 → left

1 → right shift

SLL → 32 Mux

SRL → 32 Mux

} 32×5 Mux

By adding 3 gates You can reduce 32×5 Mux which takes more than 40 logic gates