32 Bit ALU Design

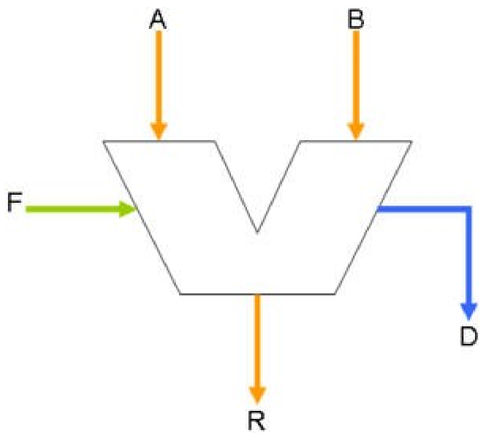
Computer Organization and Architecture Lab

Batch 41

12CS10008	ASEEM PATNI
12CS10012	ASHUTOSH BAHETI

1 Introduction

Arithmetic Logic Unit (ALU) is a critical component of a microprocessor and is the core component of central processing unit. The logic circuitry in this units is entirely combinational (i.e. consists of gates with no feedback and no flip-flops).The ALU is an extremely versatile and useful device since, it makes available, in single package, facility for performing many different logical and arithmetic operations.



Parts of ALU and Instruction Set:

- Arithmetic Unit: ADD, ADDI, INC, SUB, SUBI, DEC
- Shift Unit: SLA, SRA, SRL
- Logical Unit: AND, OR, XOR, NOT

2 Approach

There could be many different approaches in the design of ALU. All these approach may have different hardware requirements and different complexity and flexibility.

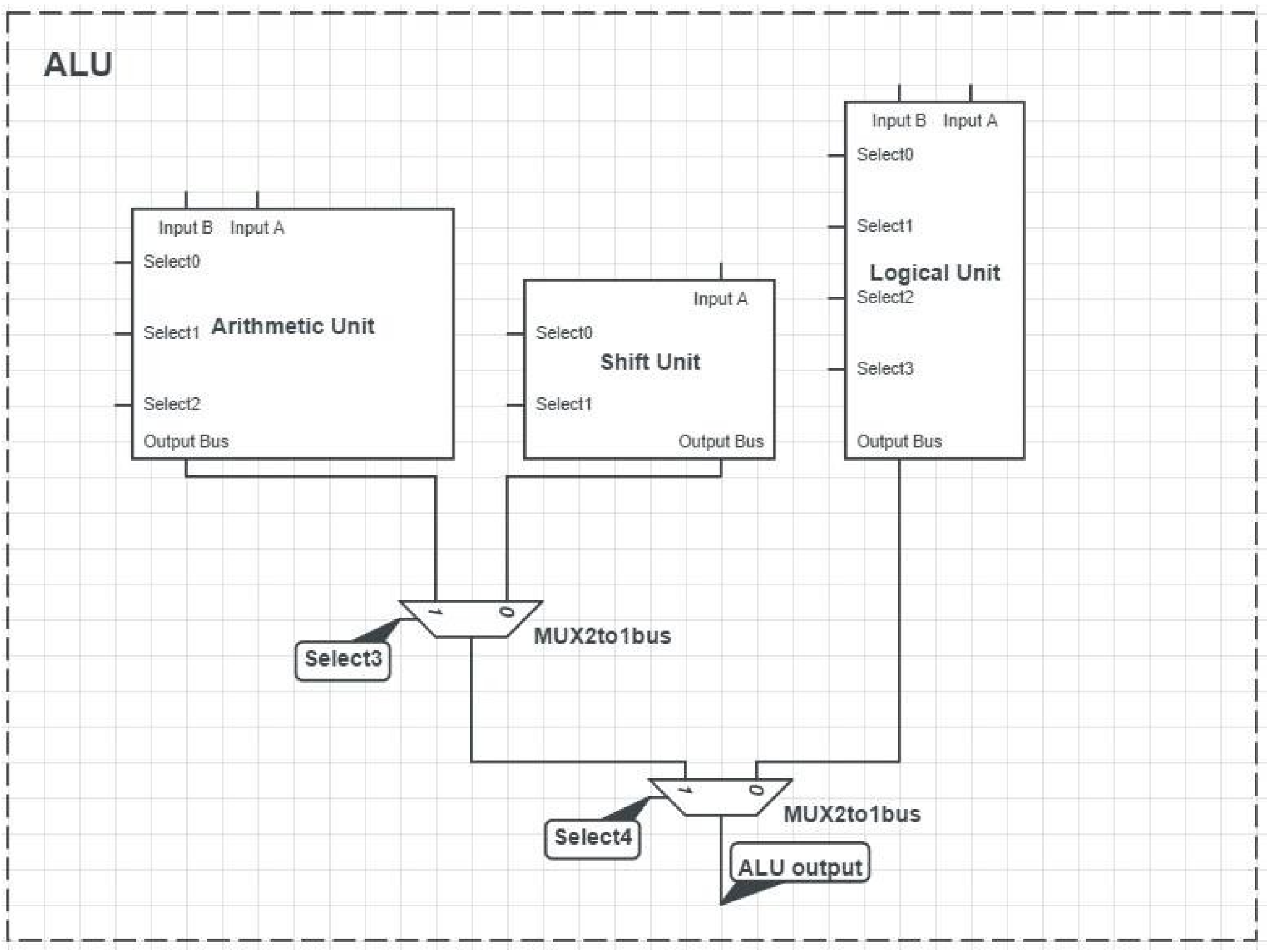
- One approach could be to first design a 1-bit ALU with alll the required operations, and then connect multiple such ALU units to make a 32-bit ALU. This approach is easy to understand and can be used to generate any bit ALU, but remains limited in terms of flexibility. Complex operations will be harder to implement like this.
- Other approach could be to design different units as different modules and then select between them using some select lines. This approach is easier to implement and scale and is flexible too.

3 Arithmetic and Logical Unit

Input:	32-bit A, 32-bit B
Output:	32-bit Out
Control:	5 select lines S0, S1, S2, S3, S4

Select Lines					**Operation**
S4	**S3**	**S2**	**S1**	**S0**	
0	0	0	0	0	ADD
0	0	0	0	1	ADDI
0	0	0	1	0	INC
0	0	1	0	0	SUB
0	0	1	0	1	SUBI
0	0	1	1	0	DEC

Select Lines					Operation
1	1	0	0	0	AND
1	1	1	1	0	OR
1	0	1	1	0	XOR
1	0	1	0	1	NOT
0	1	--	0	--	SLA
0	1	--	1	0	SRA
0	1	--	1	1	SRL

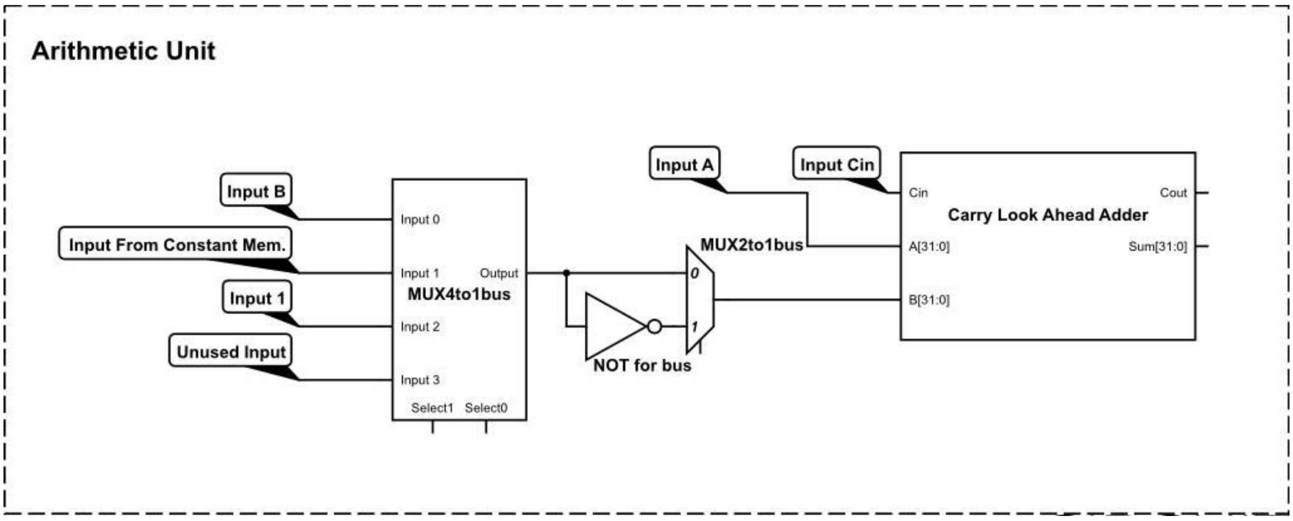


4 Arithmetic Unit

Input:	32-bit A, 32-bit B
Output:	32-bit Out
Control:	3 select lines S0, S1, S2

S2	S1	S0	Operation
0	0	0	ADD
0	0	1	ADDI
0	1	0	INC
0	1	1	---
1	0	0	SUB
1	0	1	SUBI
1	1	0	DEC
1	1	1	---

We use carry-lookahead adders (CLA) to speed up addition. A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits. It can be contrasted with the simpler, but usually slower, ripple carry adder for which the carry bit is calculated alongside the sum bit, and each bit must wait until the previous carry has been calculated to begin calculating its own result and carry bits (see adder for detail on ripple carry adders). The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits.



5 Logical Unit

Input:	32-bit A, 32-bit B
Output:	32-bit Out
Control:	4 select lines S0, S1, S2, S3

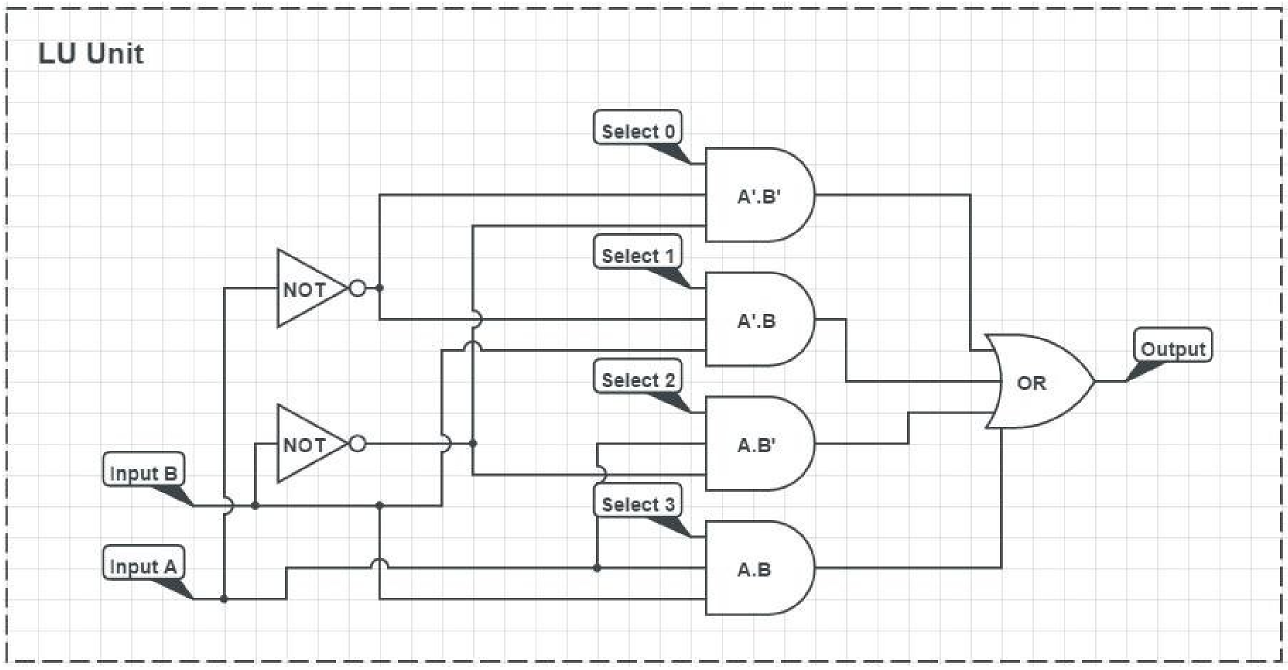
We want to implement AND, OR, XOR and NOT operations. But taking more efficient approach, we generate all the four min-terms so that any logical operation can be easily implemented. Each select line corresponds to a min-term of A and B.

Implementation of the required functions using min-terms:

- a.b = m1
- a+b = m1 + m2 + m3
- a xor b = m2 + m3
- not a = m2 + m4

Select Line	Min-term
S3	M1 = A.B
S2	M2 = B.A'
S1	M3 = A'.B
S0	M4 = A'.B'

Thus, the combination of these select lines can be used to implement any logical function.

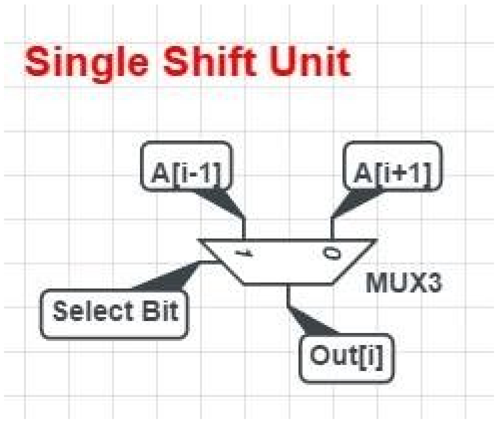


6 Shift Unit

Input:	32-bit A
Output:	32-bit Out
Control:	2 select lines S0, S1

S1	S2	Operation
0	--	SLA
1	0	SRA
1	1	SRL

A 32-bit shift unit consists of array of 32 Single bit shift units one for each output bit.



For ith shift unit:

Input:	A[i+1], A[i-1]
Output:	Out[i]
Select:	1 select line S1 (except when i=31)

When shifting left, the MSB of the output should be 0

if i=0 A[i-1] = 0

When shifting right, we need to consider whether the shift is logical or arithmetic. For this, we assign S0. So, if S0 is 1 (Logical Shift), then the MSB of the output should be 0 and when S0 is 1 (Arithmetic) then the MSB of the output should be the same as MSB of A (sign extension). Therefore, we get:

if i=31 2 select lines required S1, S0
A[i+1] = (S0.0) + (S0'.A[i])

This can be implemented by using another2 to 1 multiplexer with input as 0 and A[31], select line as S0 and output connected as A[i+1] in the MSB shift unit.