IIT KHARAGPUR

32 BIT ALU DESIGN

COMPUTER ORGANIZATION AND ARCHITECTURE LAB

SABYASACHEE BARUAH 12CS30029

NISHKARSH SHASTRI 12CS10034

As part of Computer Organization and Architecture Lab work.

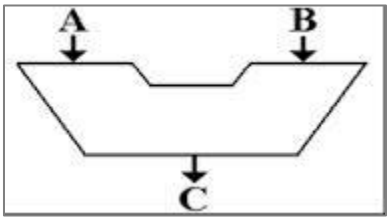
TABLE OF CONTENTS

1. Theory
2. ALU
3. Arithmetic Unit
4. Logical Unit
5. Shift Unit
6. Extensibility
7. Conclusion

THEORY

1.1 Definition

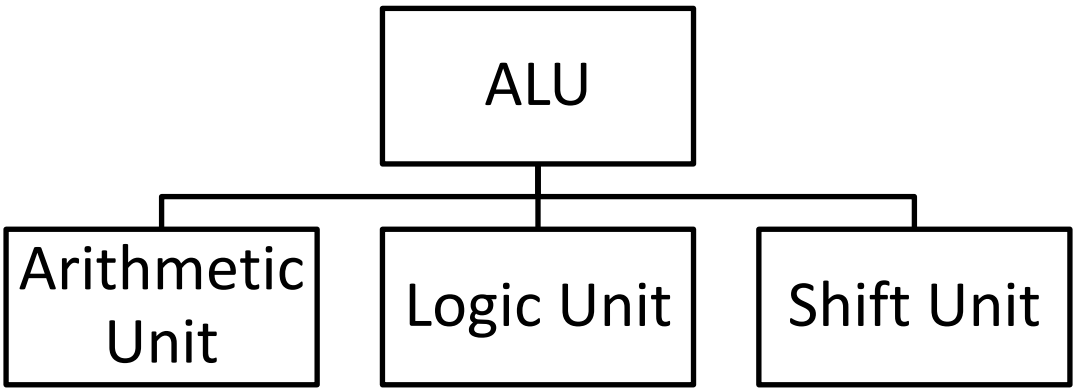
ALU stands for arithmetic logic unit and has the following circuit symbol.



It takes two bus inputs as operands and depending upon the control signals outputs a bus result. It is used to perform integer arithmetic and logic. It can also generate a set of conditions to output to a status register like overflow, divide-by-zero, less, greater etc. All processors contain at least one ALU.

1.2 Parts of ALU

As the name suggests the ALU consists of an arithmetic part and a logic part. Besides in our experiment we have also included a shift unit that left shifts or right shifts the input operand by one bit.



1.3 Instruction Set

The instruction Set we are implementing contains the following instructions:

1. Arithmetic: ADD, SUB
2. Logical :AND,OR,XOR,NOT
3. Shift :SRA,SRL,SLA

ALU

2.1 Ports

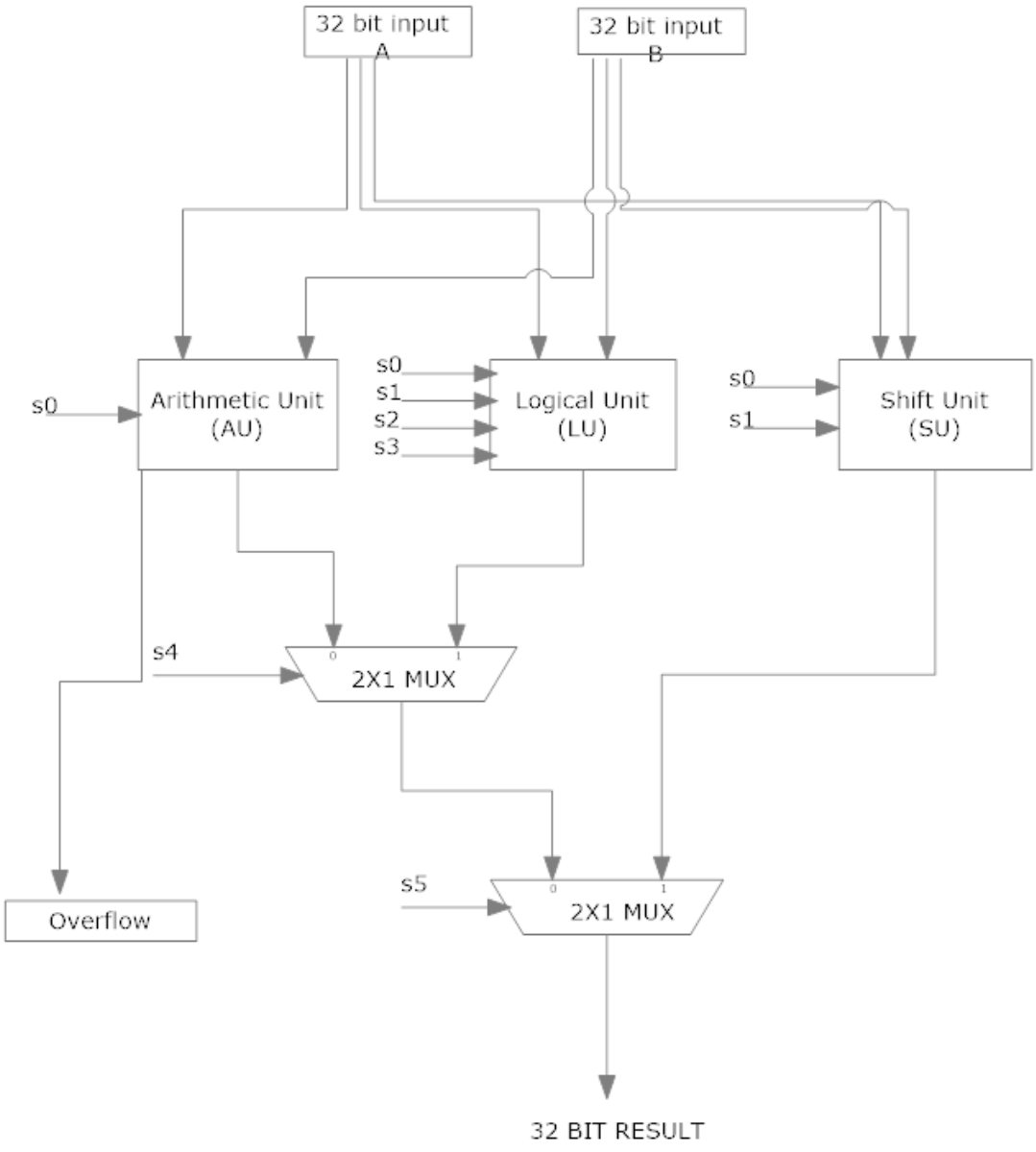
Input : 32-bit A, 32-bit B
Output : 32-bit R
Control : 6-bit Ctrl
Status Signals : 1-bit overflow

2.2 State Table

Number	Ctrl[5]	Ctrl[4]	Ctrl[3]	Ctrl[2]	Ctrl[1]	Ctrl[0]	Operation
1	0	0	0	0	0	0	ADD
2	0	0	0	0	0	1	SUB
3	0	1	1	0	0	0	AND
4	0	1	1	1	1	0	OR
5	0	1	0	1	1	0	XOR
6	0	1	0	0	1	1	NOT
7	1	0	0	0	0	0	SLA
8	1	0	0	0	0	1	SRA
9	1	0	0	0	1	0	SRL

2.3 Diagram (Block level diagram)

ALU BLOCK DIAGRAM



ARITHMETIC UNIT

3.1 Input : 32-bit A, 32-bit B
Output : 32-bit R
Status Signals : 1-bit overflow
Control : 1-bit Control Signal

3.2 Functions

We are implementing integer addition and subtraction functions. We are generating an overflow bit that is set to one if an overflow occurs. Overflow bit = Carry-in of MSB xor Carry-out of MSB. The overflow exception occurs if the result cannot be represented in 32 bits.

3.3 Subtraction

We are dealing with 2's complement numbers so subtraction is same as addition with the bits of B complemented and carry-in as 1. This is controlled by the control bit which is xor-ed with each of the bits of B and is also the carry-in to LSB.

3.4 Carrylook-ahead Addition

To speed up addition we have used carrylookahead addition instead of ripple-carry method. We have divided the 32 bit input set into two 16-bit input sets. We have passed these two sets to a carrylookahead unit with the carry out of the lower 16 bits being

fed as the carry in to the higher 16 bits. For each bit we have a propagate bit and a generate bit given by

$g = a.b$

$p = a + b$

The 16 bits are divided into 4 nibbles each with their individual carry in bits. Within a nibble the carry to each of the 4 bits are given by

$C[0] = cin$

$C[1] = cin.p[0] + g[0]$

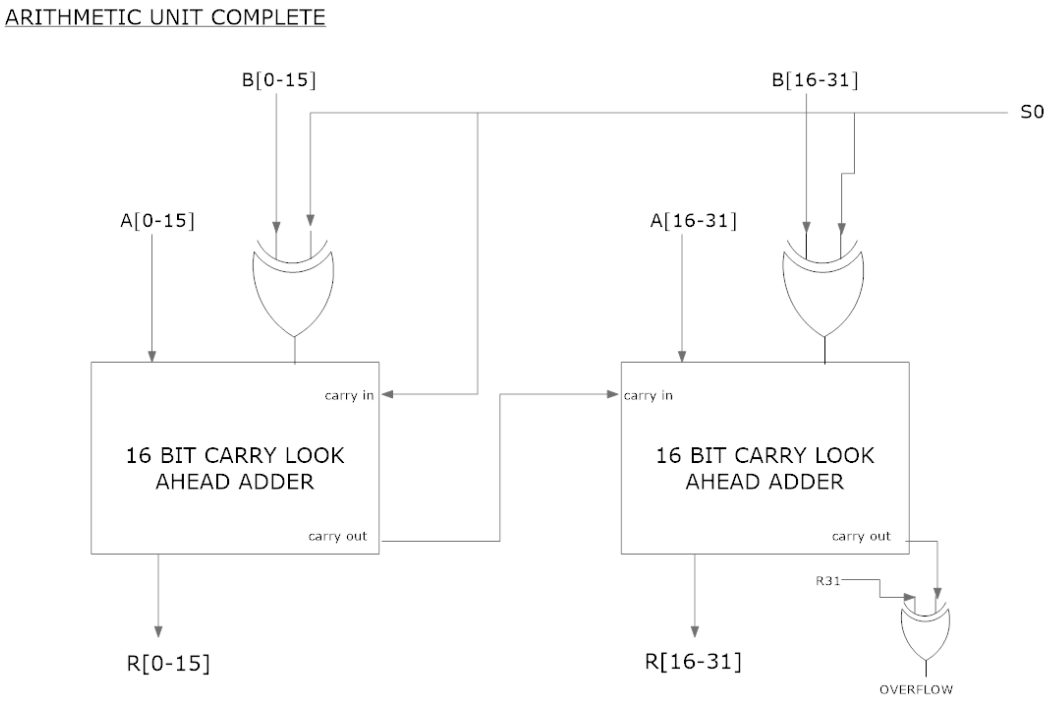
$C[2] = cin.p[0].p[1] + g[0].p[1] + g[1]$

$C[3] = cin.p[0].p[1].p[2] + g[0].p[1].p[2] + g[1].p[2] + g[2]$

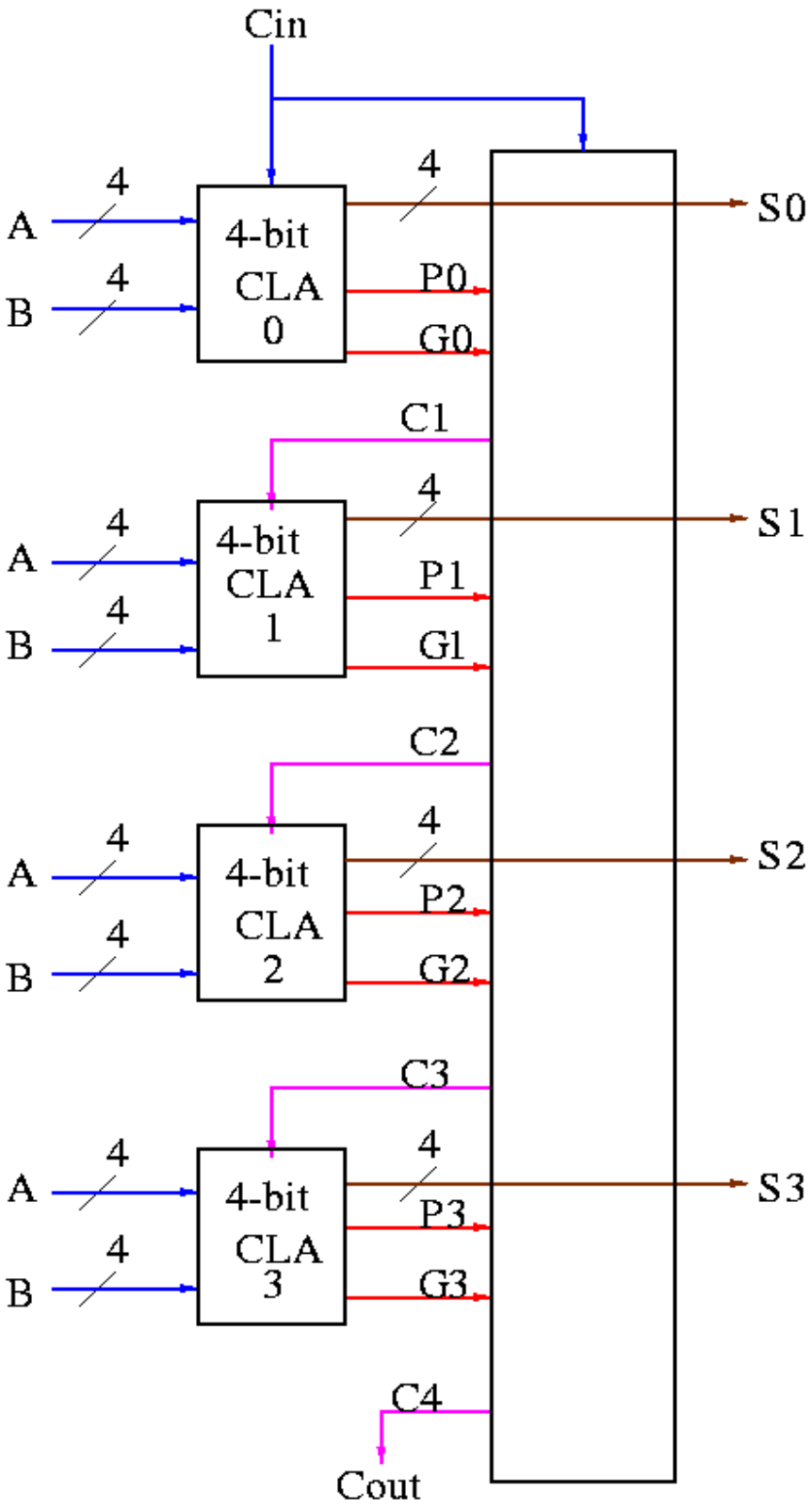
$C[4] = cin.p[0].p[1].p[2].p[3] + g[0].p[1].p[2].p[3] + g[1].p[2].p[3] + g[2].p[3] + g[3]$

where c[i],c[i+1],p[i] and g[i] are the carryin,carryout,propagate and generate bits of the ith bit and cin is the initial carryin. A similar internibble carryunit provides a second level of optimization between the 4 nibbles.

3.5 Diagram



3.6 Carrylookahead Adder



LOGICAL UNIT

4.1 Input :32-bit A, 32-bit B
Output :32-bit R
Control :4-bit Ctrl

4.2 Functions

We are implementing AND, OR, XOR and NOT operation. The design is efficient because we are generating all the 4 minterms of the individual bits that means that we can realize any logic function of A and B.

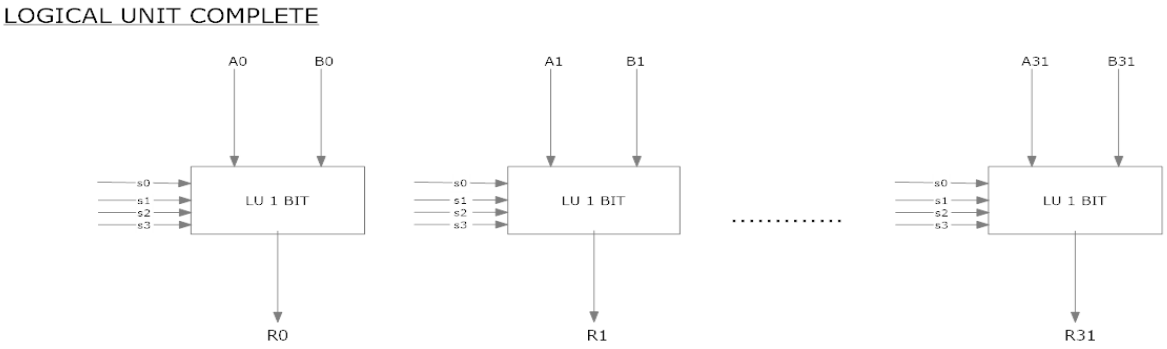
4.3 Explanation

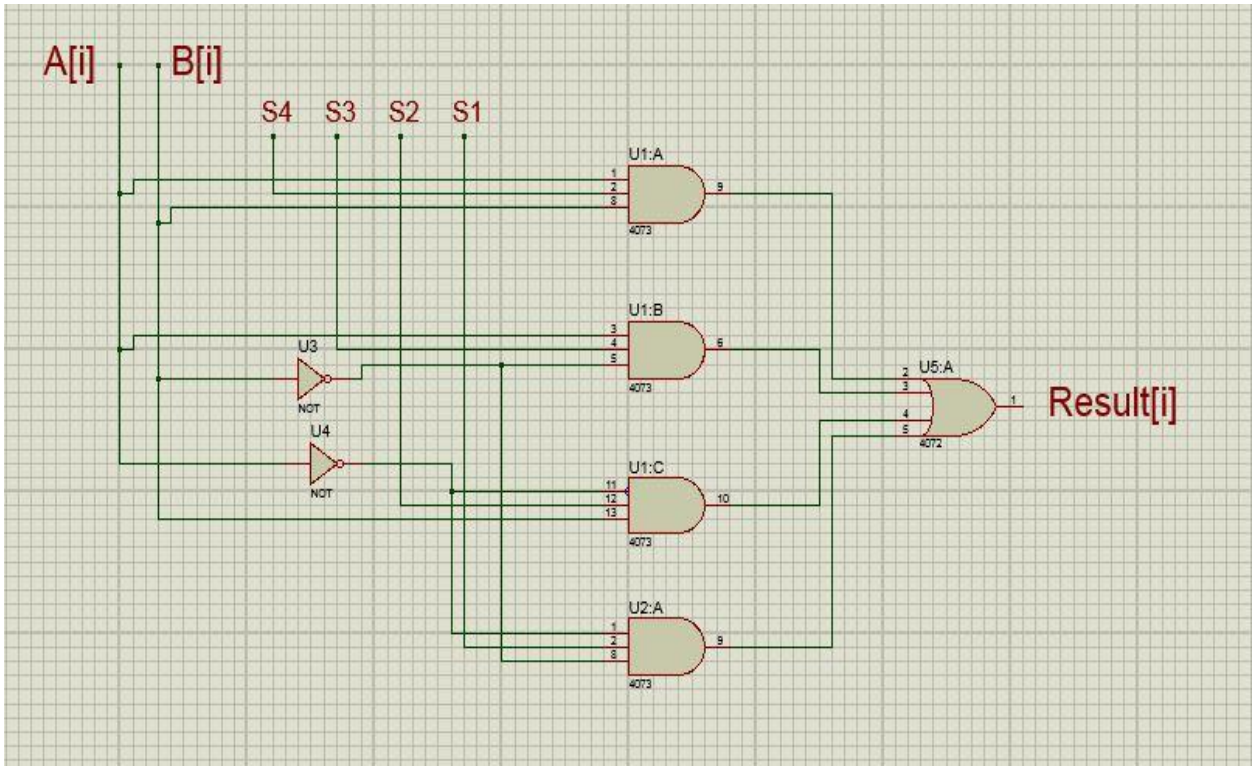
AND, OR, XOR and NOT can be written in terms of the minterms as

$a.b = m3$
$a + b = m1 + m2 + m3$
$a\ xor\ b = m1 + m2$
$not\ a = m0 + m1$

Where m3,m2,m1 and m0 are the minterms of a and b.

4.4 Diagram




A[i]
B[i]
S4
S3
S2
S1
U1:A
U1:B
U1:C
U2:A
U3
U4
U5:A
NOT
NOT
4073
4073
4073
4073
4072
Result[i]

SHIFT UNIT

5.1 Input :32-bit A
Output :32-bit R
Control :2-bit Ctrl

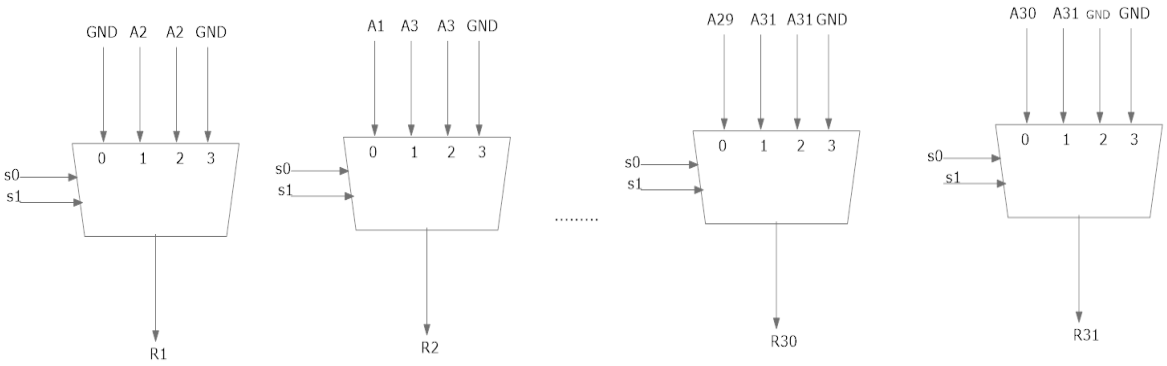
5.2 Functions

We are implementing SRA, SRL and SLA. SLA means shift left arithmetic in where you left shift the number and replace the LSB by 0. SRA means right shifting the number and preserving the sign bit in the MSB. SRL is also right shifting the number but we replace MSB by 0.

5.3 Special case of MSB

Each of the lower 31 shift units are similar but because of the difference of SRA and SRL the MUX used for the MSB has got different inputs namely the 31st bit of A(SRA), 0(SRL) and 30th bit of A(SLA).

5.4 Diagram



Shift Unit - Complete

EXTENSIBLE

We have got 6 control signals which means that we can implement at most 2^6 or 64 different functions of which we are using only 9. So we can easily add extra functioning in our ALU. Specifically we have the provision to select between different B bus inputs in the arithmetic unit that can enable us to choose between adding a register value or an immediate value as in ADDI to operand A. However we have decided to keep this selection outside the ALU domain and concern ourselves only with the arithmetic and logic functions.

Another area where we can extend the function of our ALU is in the generation of status signals. Our instruction set for the entire processor also includes comparison and branch functions that essentially compares the two operands and depending upon the result takes an action. So we can have flags for LESS, ZERO, GREAT, CARRYOUT etc. Again these can be easily implemented as we are generating carryouts for all the 32 bits.

CONCLUSION

We can design the 32-bit ALU in various ways and each have its own tradeoffs between space occupied, hardware, number of functions implemented, complexity etc. We should choose the design that is most suitable for us in our context and keep provisions for addition of more functionality in the future if required.