

# Datapath Design

Book of John P. Hayes

pp: 252-261

# Arithmetic Logic Unit

- ✓ The various circuits used to execute data processing instructions are usually combined into a single circuit called an arithmetic logic unit (ALU).
- ✓ Two types:
  1. Combinational ALU
  2. Sequential ALU

# A Simple Combinational ALU

- ✓ It combines the functions of a two's complement adder-subtractor with those of a circuit that generates word-based logic functions of the form  $f(X, Y)$ .
- ✓ The minterms of  $f(x_i, y_i)$  are

$$m_3 = x_i y_i \quad m_2 = x_i \bar{y}_i \quad m_1 = \bar{x}_i y_i \quad m_0 = \bar{x}_i \bar{y}_i$$

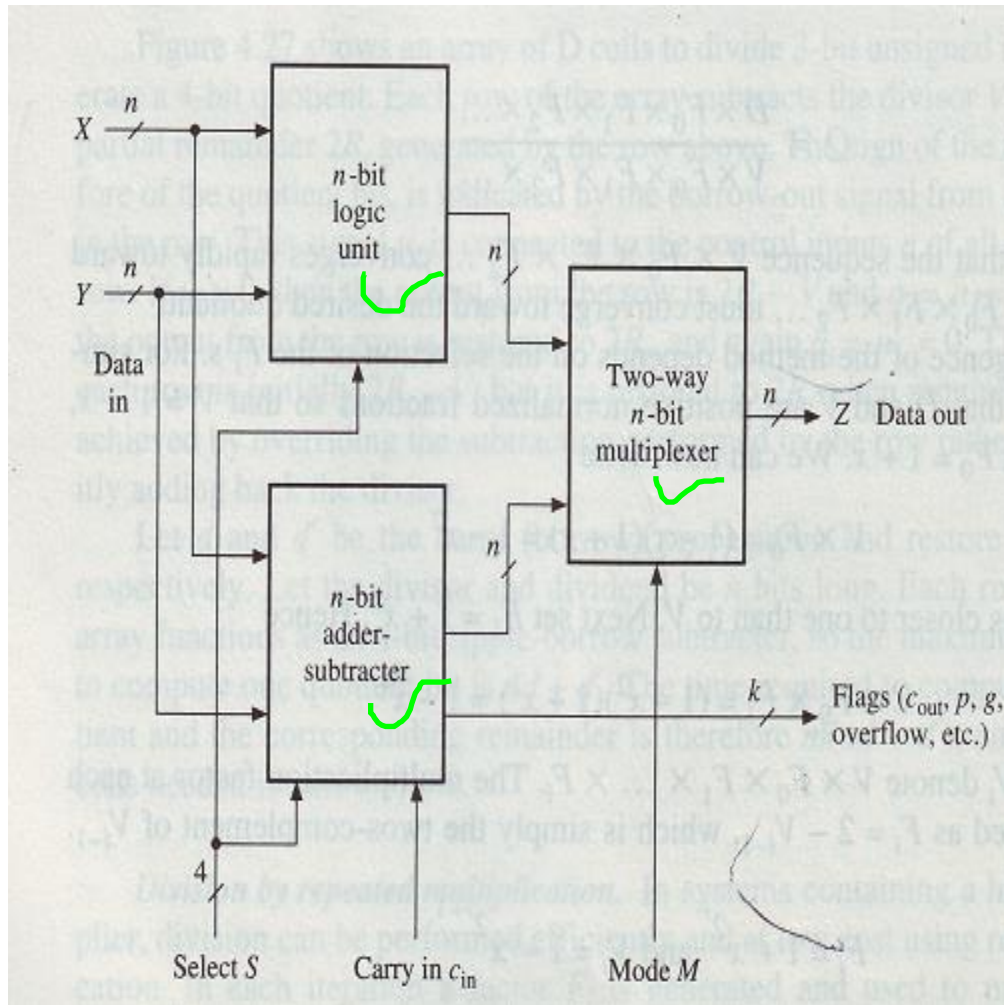
- ✓ We can construct the sum-of-product expressions

$$\begin{aligned} f(x_i, y_i) &= m_3 S_3 + m_2 S_2 + m_1 S_1 + m_0 S_0 \\ &= x_i y_i S_3 + x_i \bar{y}_i S_2 + \bar{x}_i y_i S_1 + \bar{x}_i \bar{y}_i S_0 \end{aligned}$$

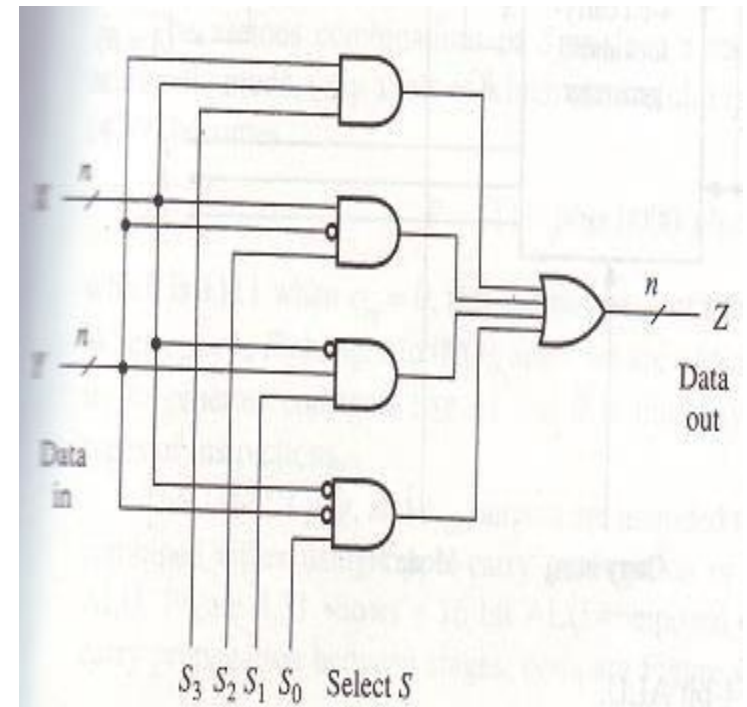
- ✓ Now we can write with n-bit words X and Y

$$f(X, Y) = X Y S_3 + X \bar{Y} S_2 + \bar{X} Y S_1 + \bar{X} \bar{Y} S_0$$

- ✓ S can also be used to select up to 16 different arithmetic operations.
- ✓ It is more expensive and slower in operation. The complete 4-bit ALU can therefore be expected to contain more than 100 gates and have depth 9 or so.



A basic  $n$ -bit combinational ALU



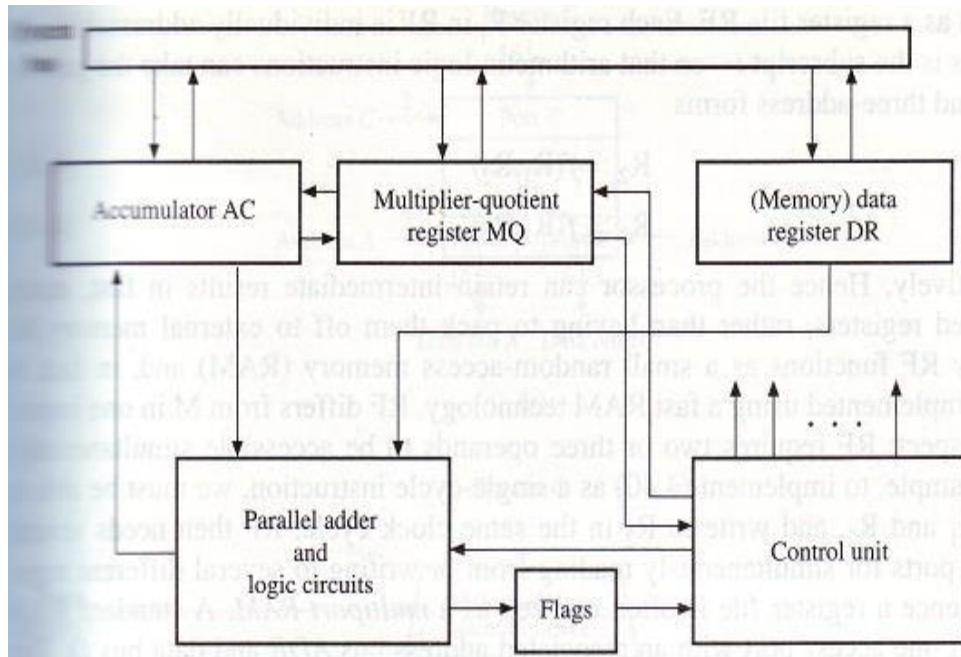
A  $n$ -bit logic unit

# Why Sequential ALU?

---

- ✓ Combinational multiplier and dividers are costly in terms of hardware.
- ✓ The number of gates in the multiply-divide logic is also greater.

# Sequential ALU



The role of these registers



Addition	$AC := AC + DR$
Subtraction	$AC := AC - DR$
Multiplication	$AC.MQ := DR \times M$
Division	$AC.MQ := MQ / DR$
AND	$AC := AC \text{ and } DR$
OR	$AC := AC \text{ or } DR$
EXCLUSIVE-OR	$AC := AC \text{ xor } DR$
NOT	$AC := \text{not}(AC)$

# Register Files

---

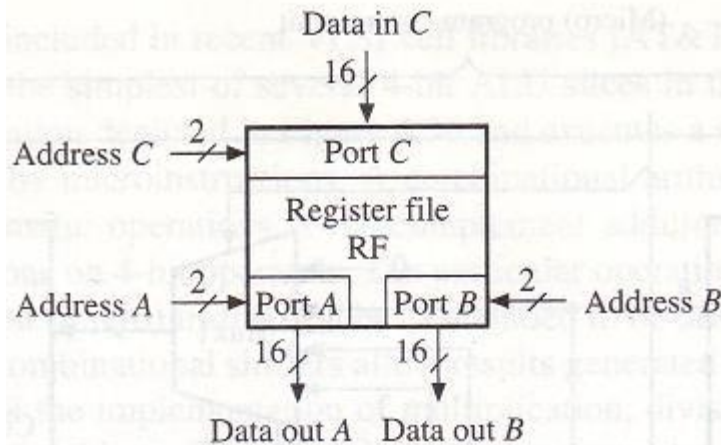
- ✓ Modern CPUs retain special registers like MQ for multiplication and division but AC and DR are replaced by a set of general purpose registers  $R_0:R_{m-1}$ , known as register file RF.
- ✓ Processor can retain intermediate results in fast, easily accessed registers.
- ✓ Each register  $R_i$  in RF is individually addressable- the address is the subscript i.
- ✓ The arithmetic-logic instruction can take the following 2 or 3 address forms
$$R_2 := f(R_1, R_2)$$
$$R_3 := f(R_1, R_2)$$

# Register Files

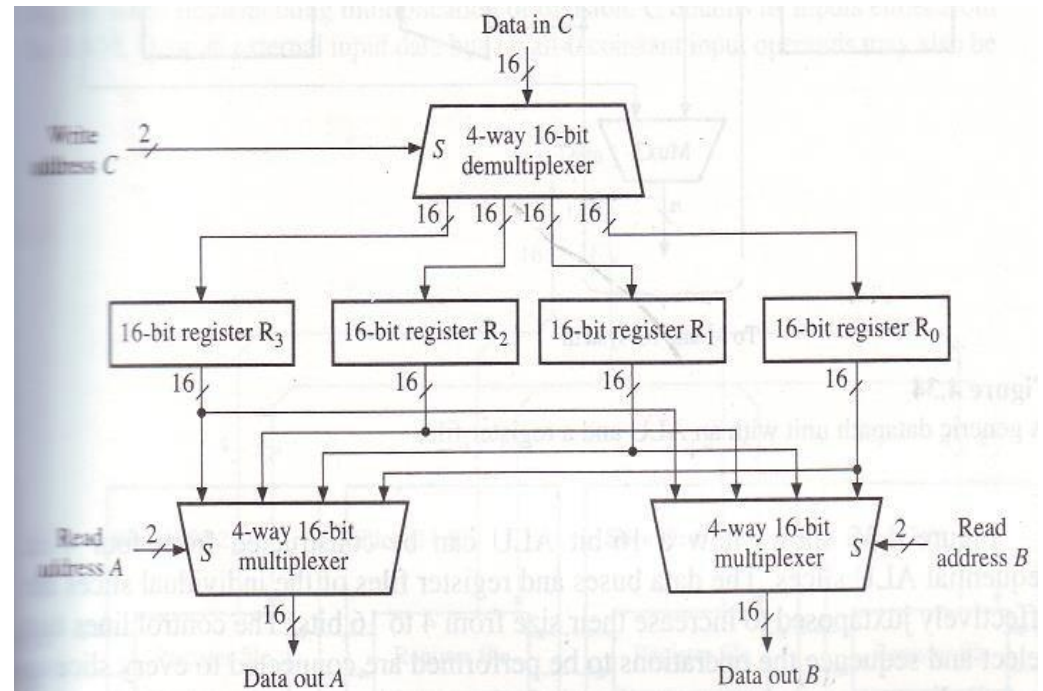
- ✓ RF requires two or three operands to be accessible simultaneously. 
- ✓ RF needs several access ports for simultaneously reading from or writing to several different registers. So, RF is often known as multiport RAM. 
- ✓ Building multiport RF requires a set of registers of the appropriate size and several multiplexer and demultiplexers.



# Register Files with Three Access Ports

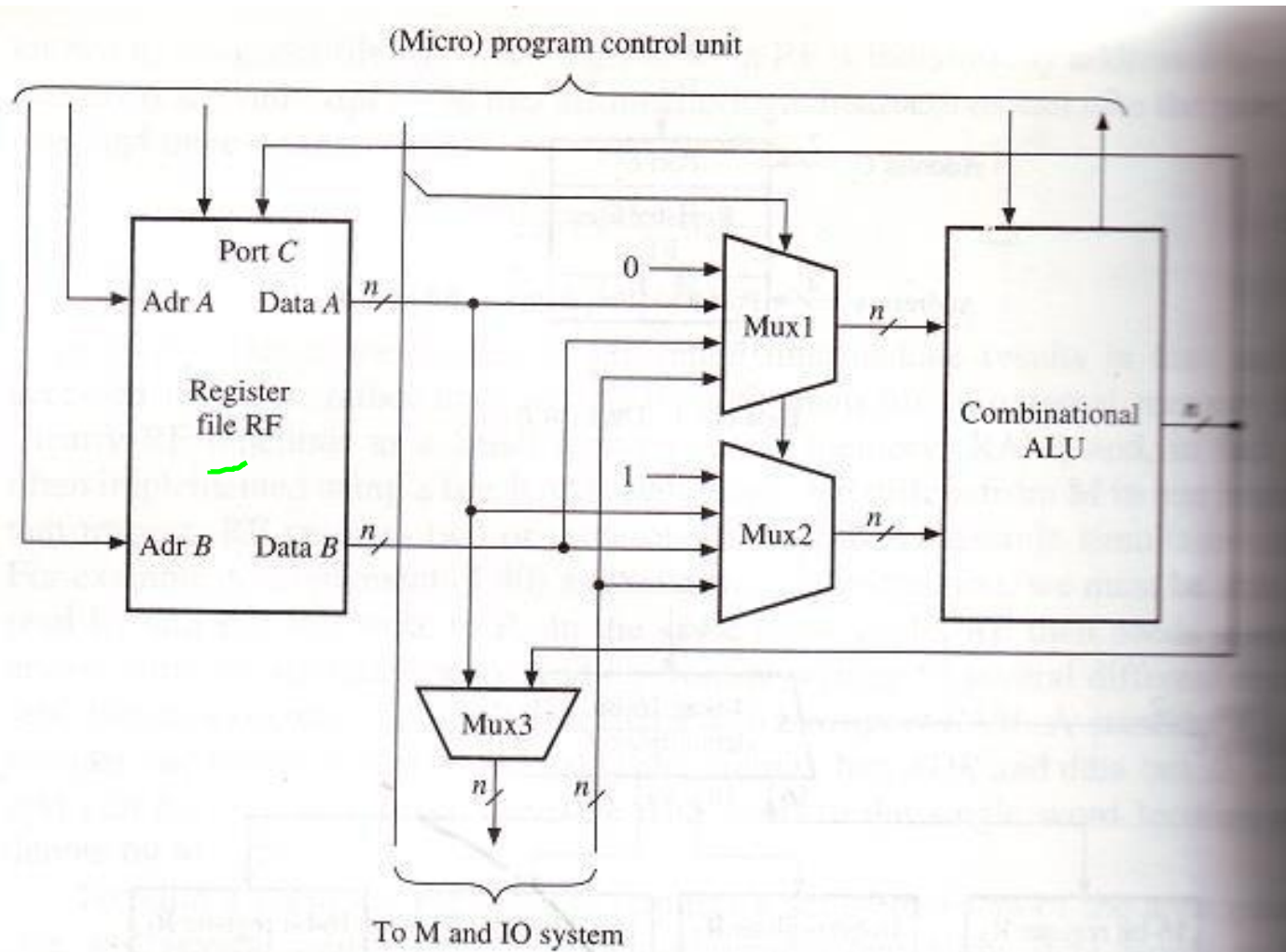


(a) Symbol



(b) Logic Diagram

# A Generic Datapath Unit with an ALU and a Register File



# ALU Expansion

✓ Generally an entire sequential ALU for fixed-point  $m$ -bit numbers is manufactured on a single chip. ALU can easily be designed for expansion to handle operands of size  $n=km$  in two ways:

1. Spatial Expansion
2. Temporal Expansion

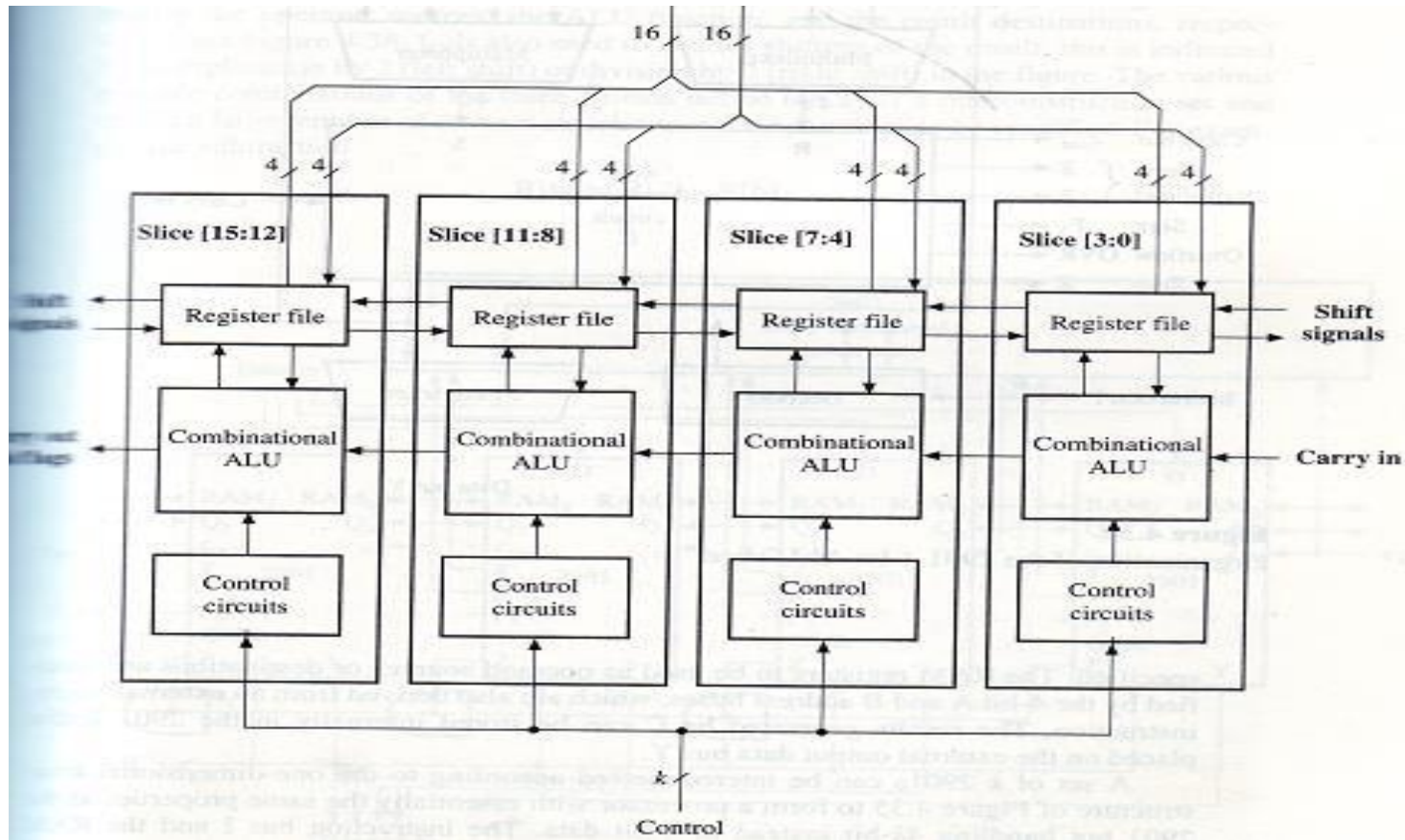
# Spatial Expansion

- ✓ Connect  $k$  copies of the  $m$ -bit ALU in the manner of a ripple-carry adder to form a single ALU capable of processing  $km$  bit words directly.
- ✓ The resulting circuit is called bit sliced. Each component ALU concurrently processes a separate slices of  $m$ -bits from each  $km$  bit operands.

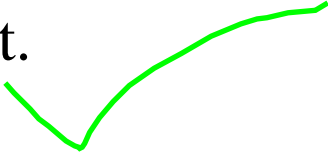
# Temporal Expansion

- ✓ Use one copy of the m-bit ALU chip in a manner of a serial adder.
- ✓ It performs an operation on km-bit words in k consecutive steps.
- ✓ In each step the ALU processes a separate m-bit slice of each operand.

# 16-bit ALU Composed of four 4-bit Slices



# Spatial vs Temporal Expansion

- ✓ The hardware cost of a bit sliced ALU increases directly with  $k$  but the performance measured in cycle per instruction (CPI) remains essentially constant.
  - ✓ The performance decreases directly with  $k$ , but the amount of hardware remains constant.
- 

# Constructing a Basic Arithmetic Logic Unit

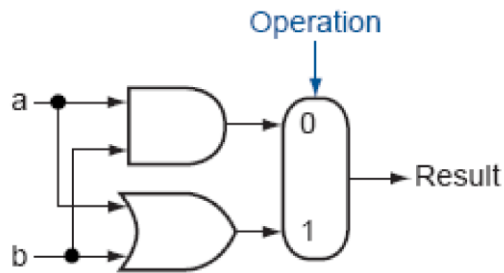


Fig: 1-bit logical unit for AND and OR

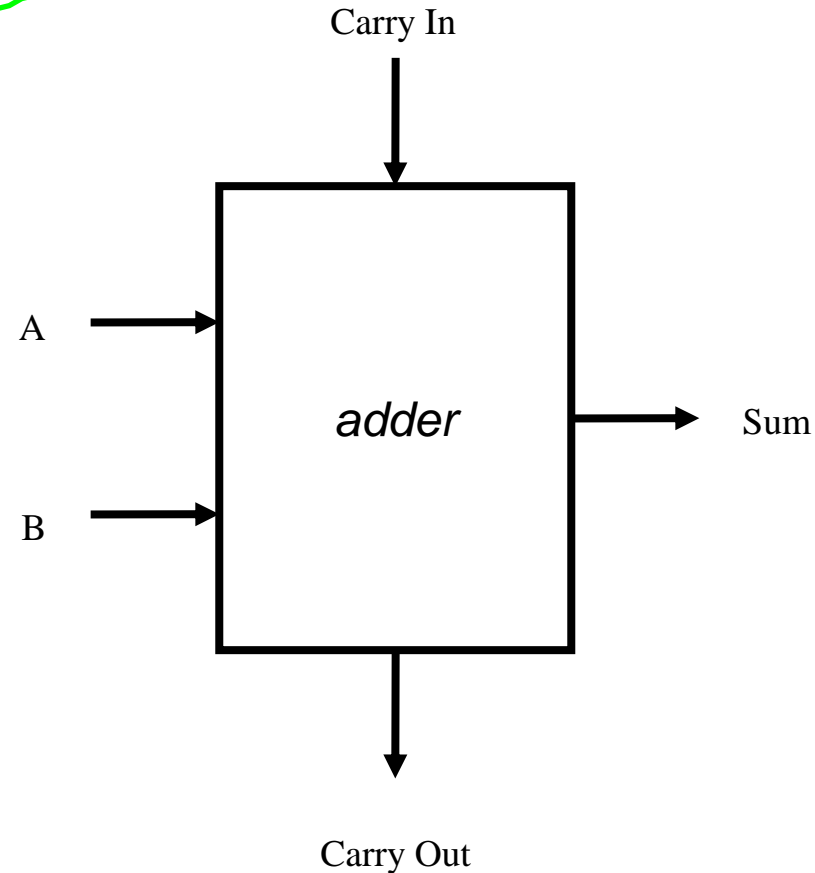
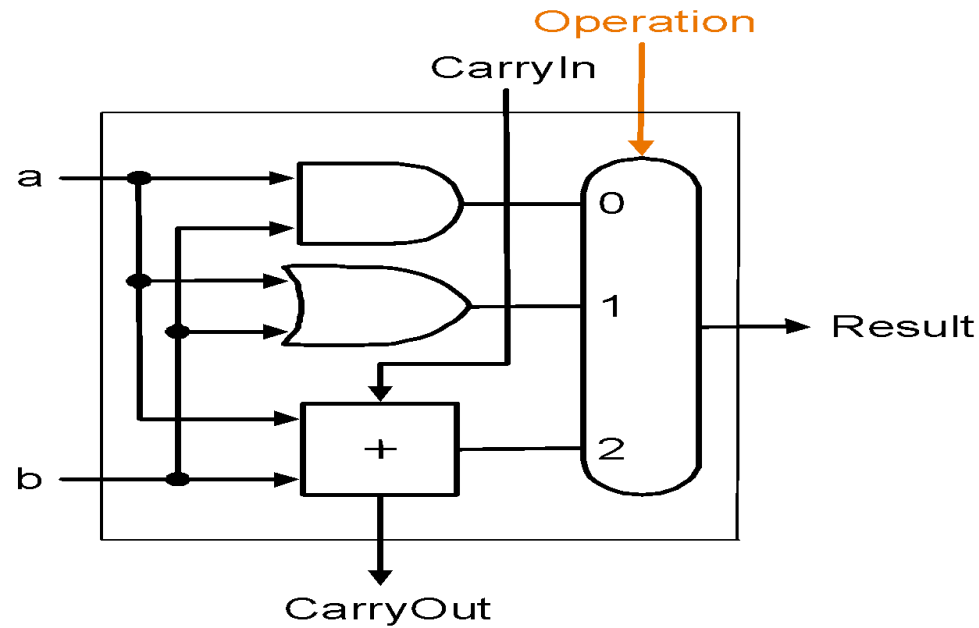


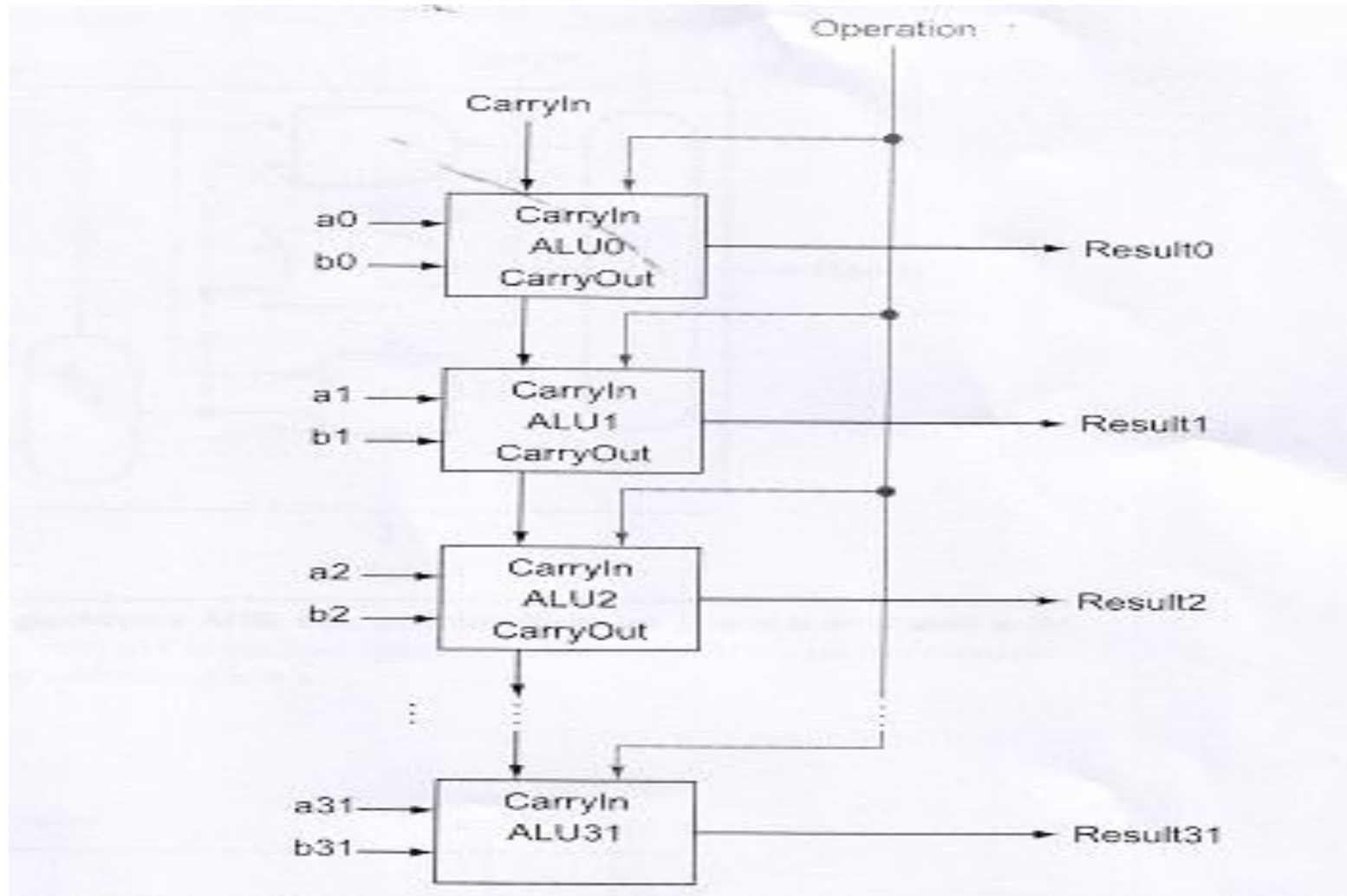
Fig: 1-bit Adder



# A 1-bit ALU that performs AND, OR and addition

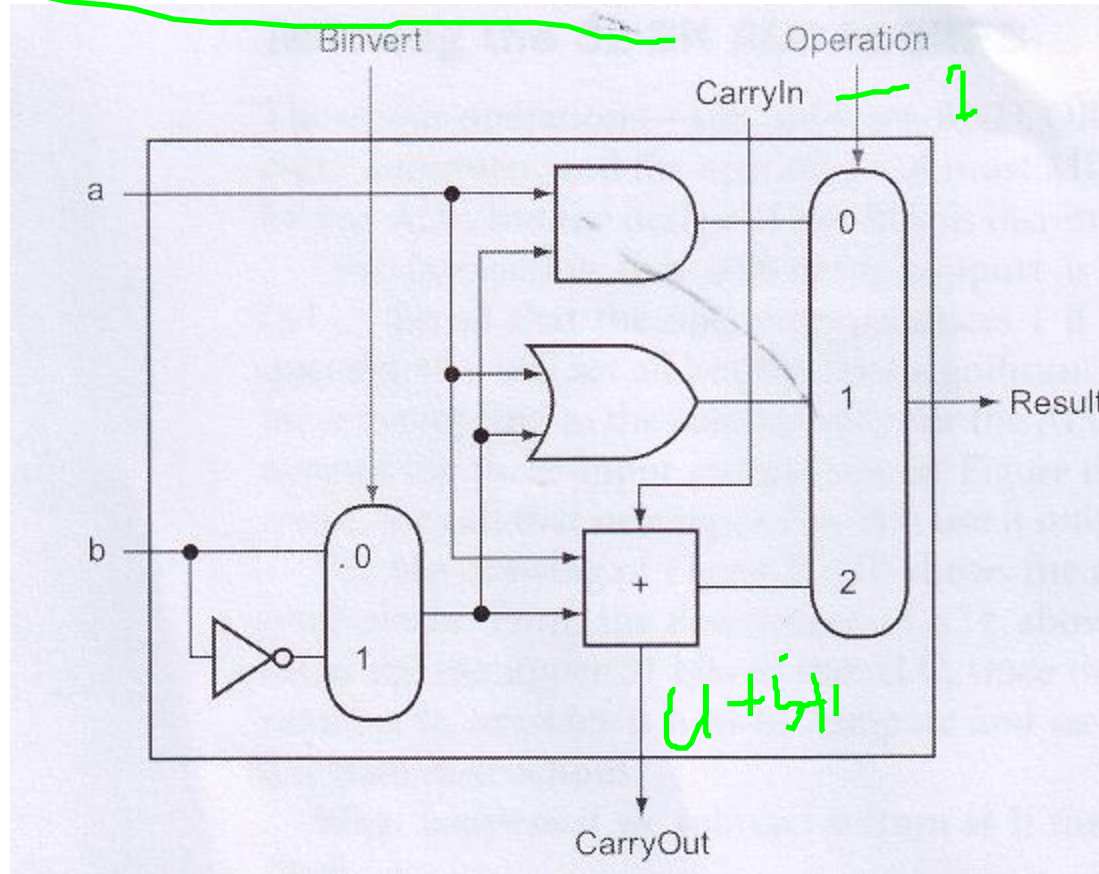


# A 32-bit ALU Constructed from 32 1-bit ALU



# A 1-bit ALU that performs AND, OR and addition on **a** and **b** or **a** and **$\bar{b}$**

504n



$$a + \bar{b} + 1 = a - b \quad [1 \text{ is set in the CarryIn signal of the LSB ALU}]$$

$$a - b = a + (-b) = a + \bar{b} + 1$$

# A 1-bit ALU that performs AND, OR, NOR and Addition

OR

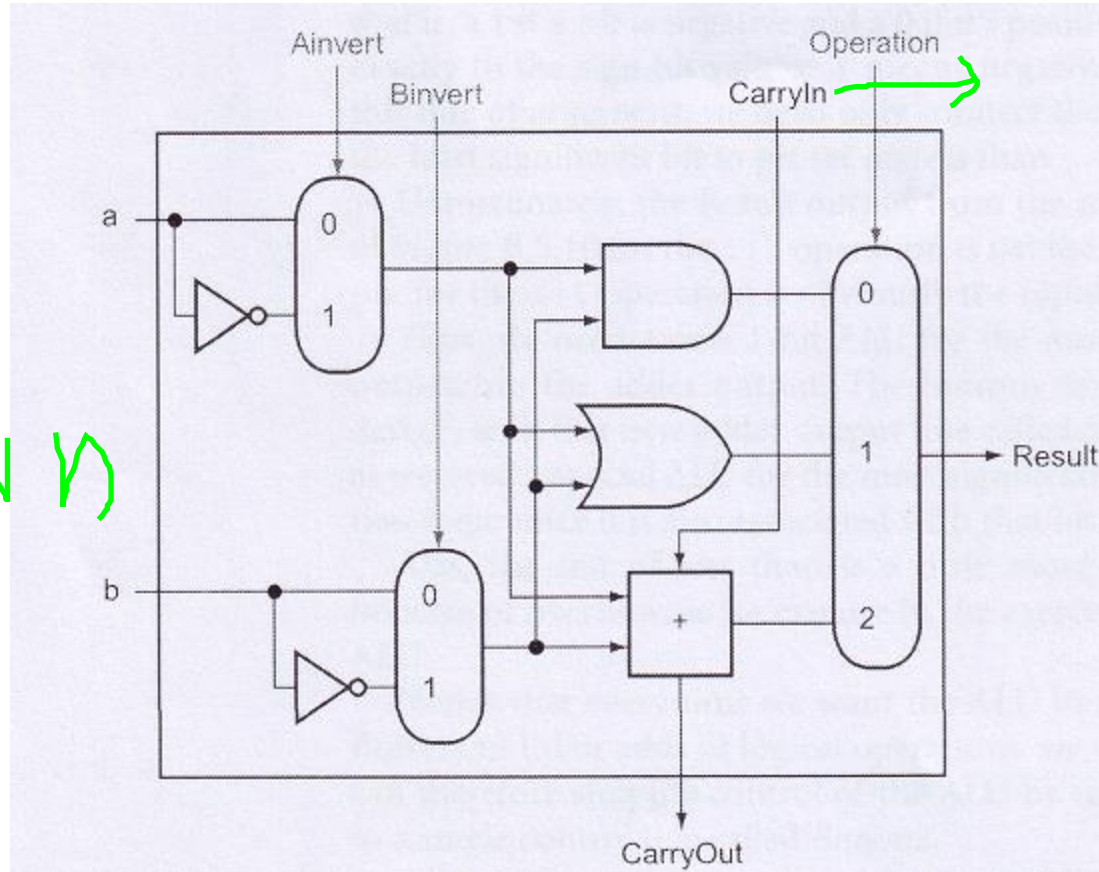
and

NOR

NAND

add

sub



$$\overline{(a+b)} = \overline{a} \cdot \overline{b}$$

# 32 bits ALU for MIPS to Support Slt

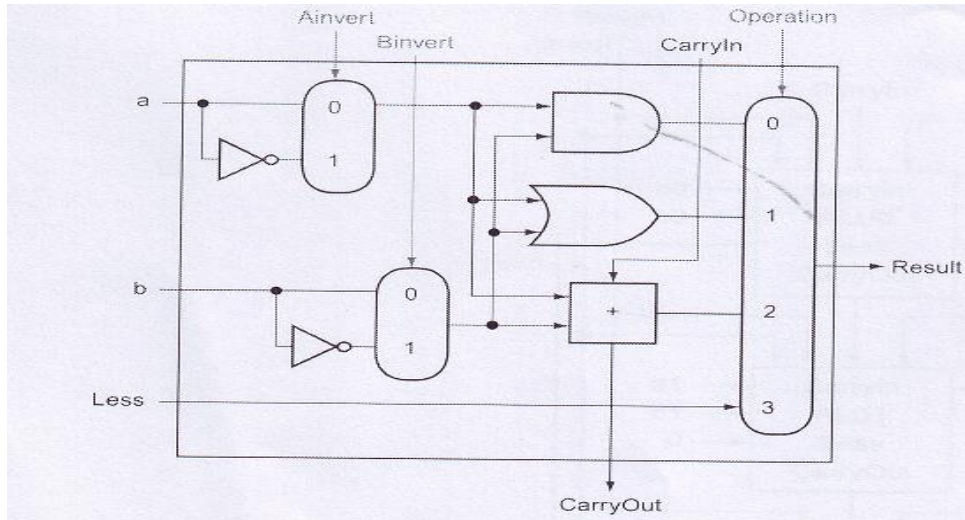


Fig: ALU for Upper 31 bits

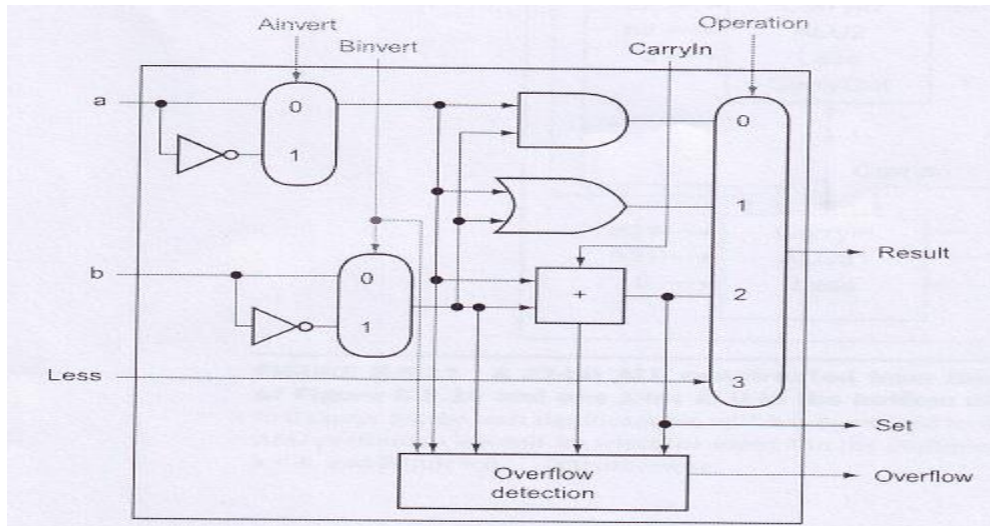
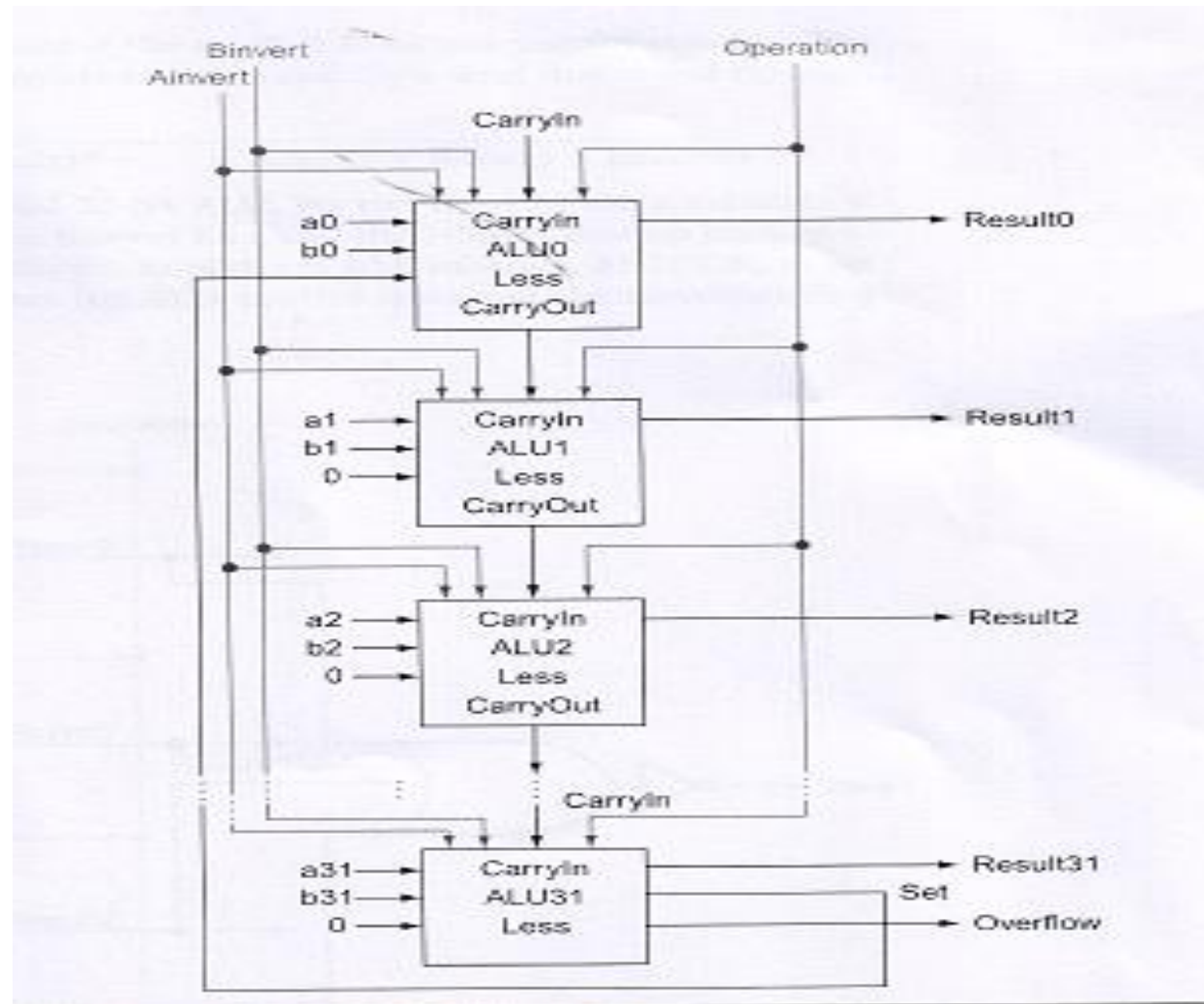


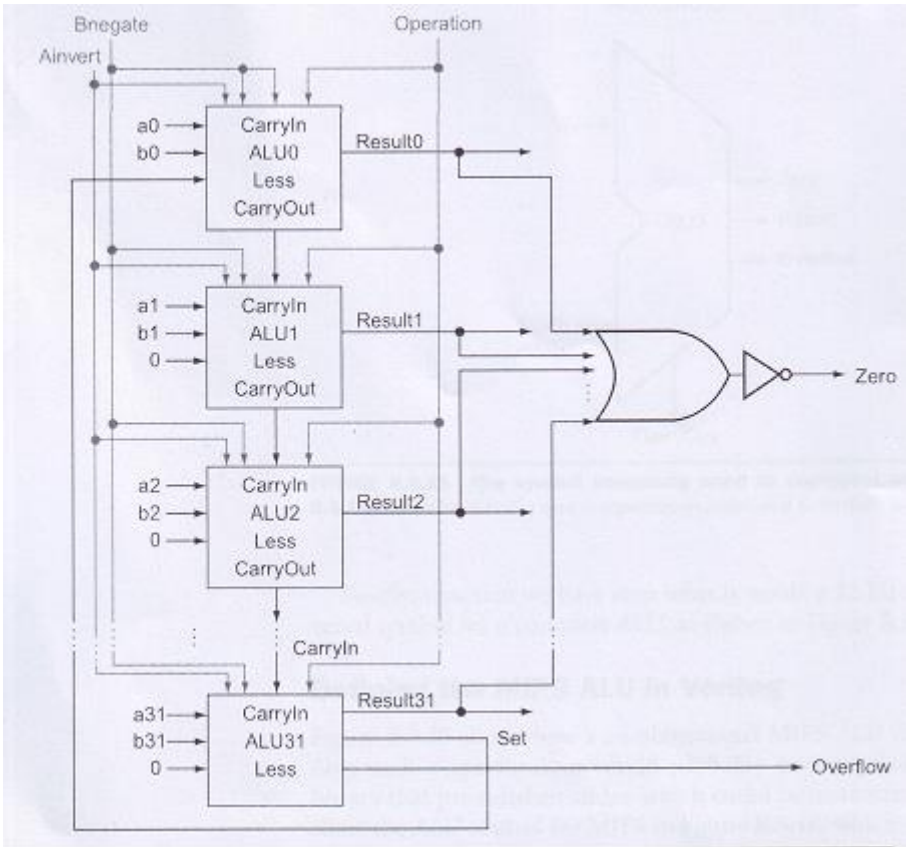
Fig: MSB ALU

# 32 bits ALU for MIPS to Support ALU





# Final 32 bits ALU for MIPS to Zero Checking



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Fig: Control Signal for the ALU

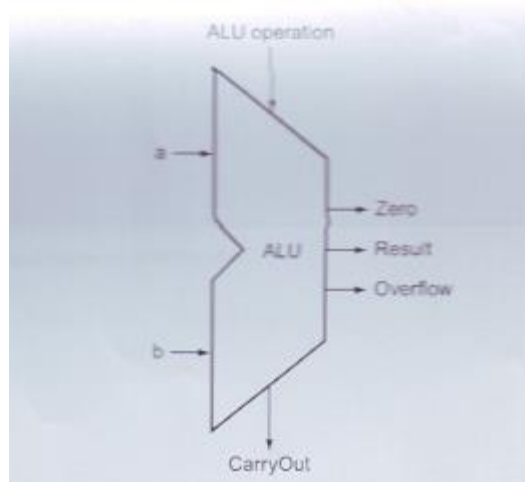


Fig: Symbol Representing the 32 bit ALU