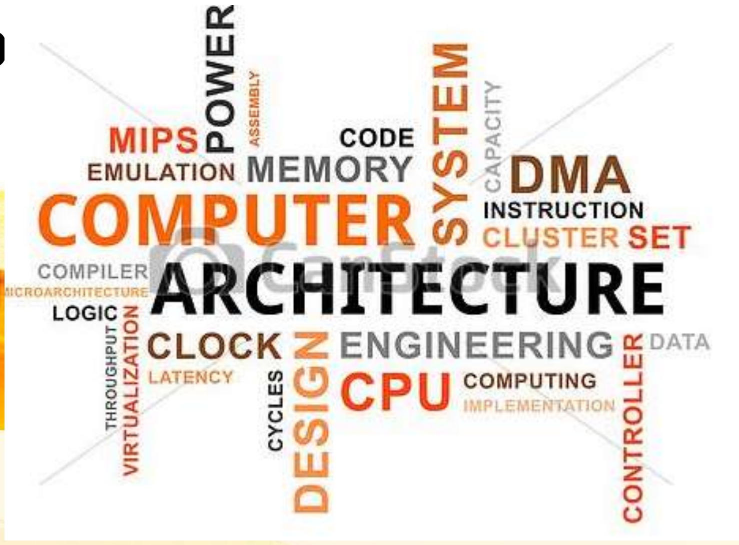
UNIT II
ARITHMETIC OPERATIONS

Addition and subtraction of signed numbers – Design of fast adders – Multiplication of positive numbers - Signed operand multiplication- fast multiplication – Integer division – Floating point numb



Computer Arithmetic’s

- **Arithmetic Instruction** Manipulate data to produce results necessary for the solution of computational problem

- Four Basic Arithmetic Operations are – **Addition, Subtraction, Multiplication and Division**

- An **arithmetic processor** is the part of a processor unit that executes arithmetic operations

- Arithmetic operations can be performed for following data types
  - Fixed point binary data in signed magnitude representation
  - Fixed point binary data in signed 2’s complement representation
  - Floating Point binary data
  - Binary coded decimal data

Addition and Subtraction

Representation of both *positive* and *negative* numbers

- Following 3 representations

Signed magnitude representation
Signed 1's complement representation
Signed 2's complement representation

Example: Represent +9 and -9 in 7 bit-binary number

Only one way to represent + 9 ==> 0 001001

Three different ways to represent - 9:

In signed-magnitude:	1 001001
In signed-1's complement:	1 110110
In signed-2's complement:	1 110111

Addition and Subtraction

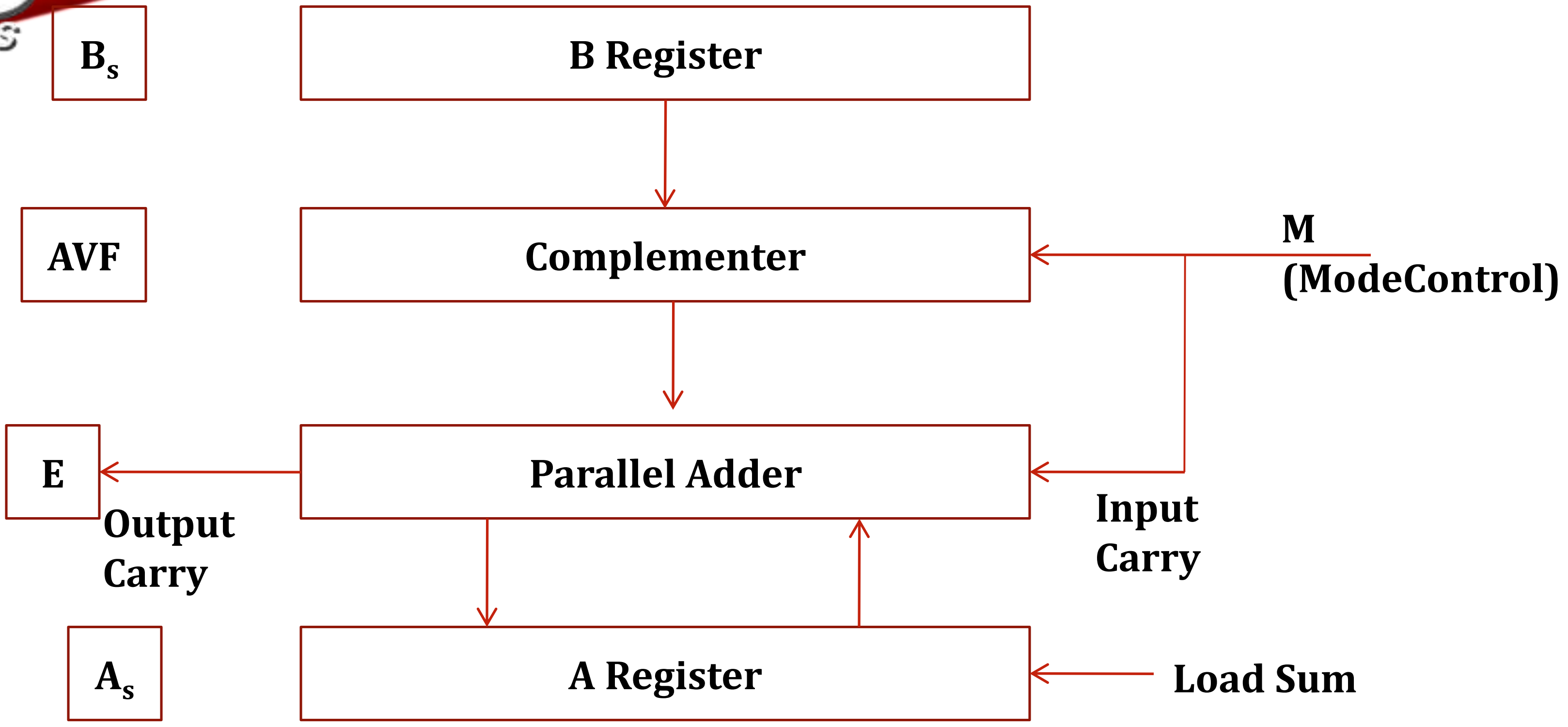
Operation	Add Magnitudes	Subtract Magnitudes		
		A>B	A<B	A=B
(+ A) + (+ B)	+ (A + B)			
(+ A) + (- B)		+ (A - B)	- (B - A)	+ (A - B)
(- A) + (+ B)		- (A - B)	+ (B - A)	+ (A - B)
(- A) + (- B)	- (A + B)			
(+ A) - (+ B)		+ (A - B)	- (B - A)	+ (A - B)
(+ A) - (- B)	+ (A + B)			
(- A) - (+ B)	- (A + B)			
(- A) - (- B)		- (A - B)	+ (B - A)	+ (A - B)

Computer Arithmetic's

Addition (subtraction) Algorithm

- When the sign of A and B are identical (different) , add the magnitudes and attach the sign of A to the result.
- When the signs of A and B are different (identical), compare the magnitudes and subtract the smaller number from the larger.
  - Choose the sign of result to be same as A if A>B
  - or the complement of sign of A if A<B
  - if A=B subtract B from A and make the sign of result positive

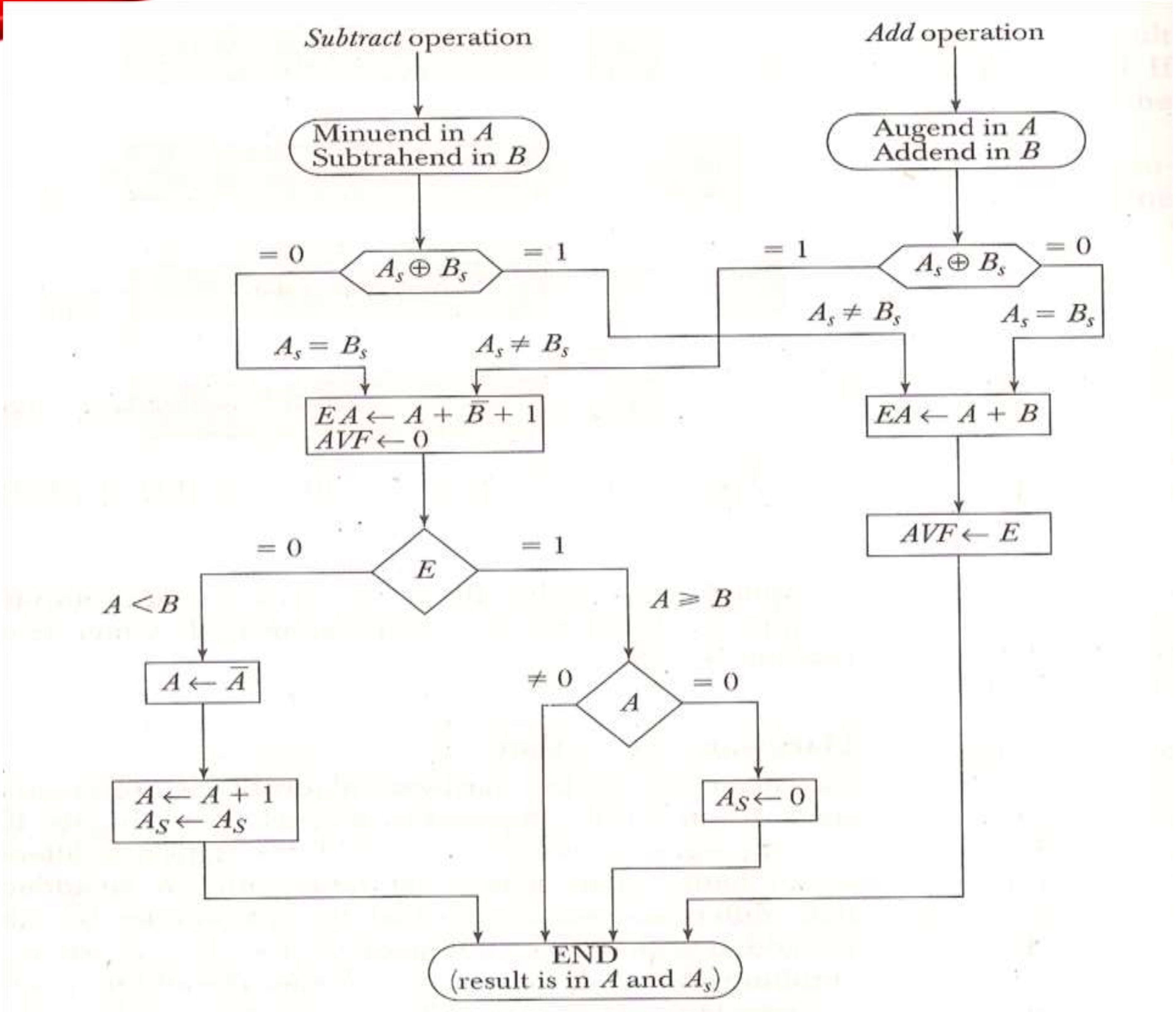
Hardware Implementation



Simple procedure require magnitude comparator, an adder, two subtractor however alternative reveals that using 2’s complement for operation requires only an adder and a complementor

M=0 output = **A+B** **M=1** output = A+B’+1= **A-B**

Flow Chart for Add and Subtract Operation



Signed 2’s Complement Representation

Addition :

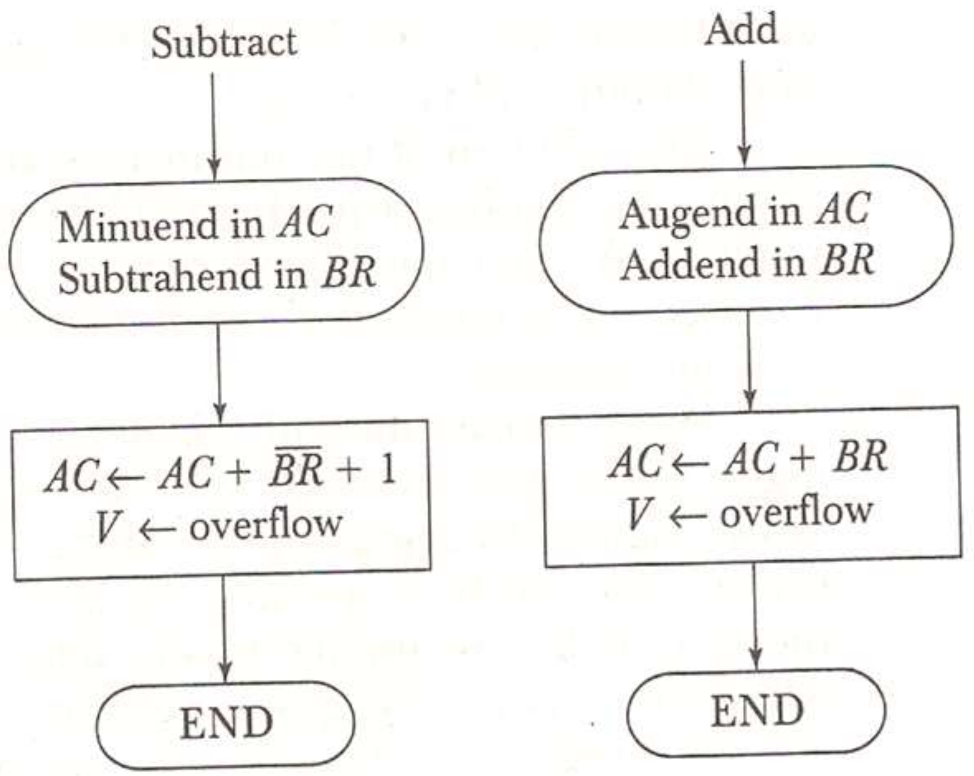
- Addition of two numbers in signed 2’s complement form consists of adding the numbers with the sign bits treated the same as the other bits of the number. Carry out of sign bit is discarded
- Sum is obtained by adding the content of AC and BR (including the sign bit). Overflow bit is set to 1 if EX-OR of last two carries if 1

Subtraction :

- Here Subtraction consists of first taking the 2’s complement of the subtrahend and then adding it to minuend

- Subtraction done by adding the content of AC to 2’s Complement of BR.

Signed 2's Complement Representation



PROBLEM

- Addition and subtraction is the basic operation to be performed by computer

- Digits are added bit by bit from right to left, with carries passed to the next digit to the left.

Example

Adding 6_{10} to 7_{10} in binary

Solution

6	0110
7	0111
13	1101

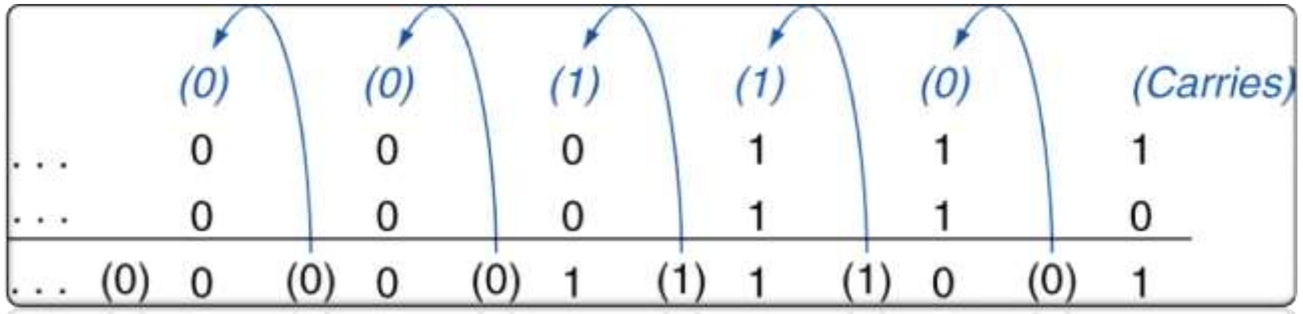
COMPUTER ADDITION

- Can be taken place in 32 bit formats

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_2 = 7_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_2 = 6_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_2 = 13_{10}$



7/10/2017 A.Aruna / CS6303 /ADDITION

Overflow in Addition

Happened in addition and subtraction

- Cannot be represented in available hardware
- Example :
  - First operand – 32 bit word
  - Second operand – 32 bit word
  - Result also must be 32 bit word
  - If exceeds more than 32 bit word causes overflow

- Adding +ve and –ve operands, no overflow
- Adding two +ve operands
  - Overflow if result sign is 1
- Adding two –ve operands
  - Overflow if result sign is 0
- Example : -10 + 4 = -6.
  - Since the operands fit in 32 bits and the sum is no larger than an operand, the sum must fit in 32 bits as well.
  - No overflow can occur when adding positive and negative operands.
- Adding or subtracting two 32-bit numbers can yield a result that needs 33 bit for fully expressed

- The lack of a 33rd bit means that when overflow occurs, the sign bit is set with the value of the result instead of the proper sign of the result.
- Overflow occurs when adding a 2 positive numbers and the sum will be negative and vice- versa (adding 2 positive and 2 negative numbers)
- Addition can be performed in two ways
  - Unsigned binary Numbers
  - Signed Binary Numbers
- Overflow bit indicated by flag bit

- Positive number – unsigned
- Negative number – signed
- Ordinary,
  - + sign indicative +ve
  - – sign indicate –ve.
- In computer, everything are binary numbers,
  - 0 represents positive number
  - 1 represents Negative numbers

Before processing user must identify , whether the number is signed or unsigned

- Left most bit represent the sign bit

Example

01001 **+9**

11001 **- 9**

Example

- Consider a two 4 bit positive number
- +9 and +8 = 01001 + 01000 = 10001
- Consider a two 8 bit positive number
- +98 and +87

    01001 1000

    01000 0111

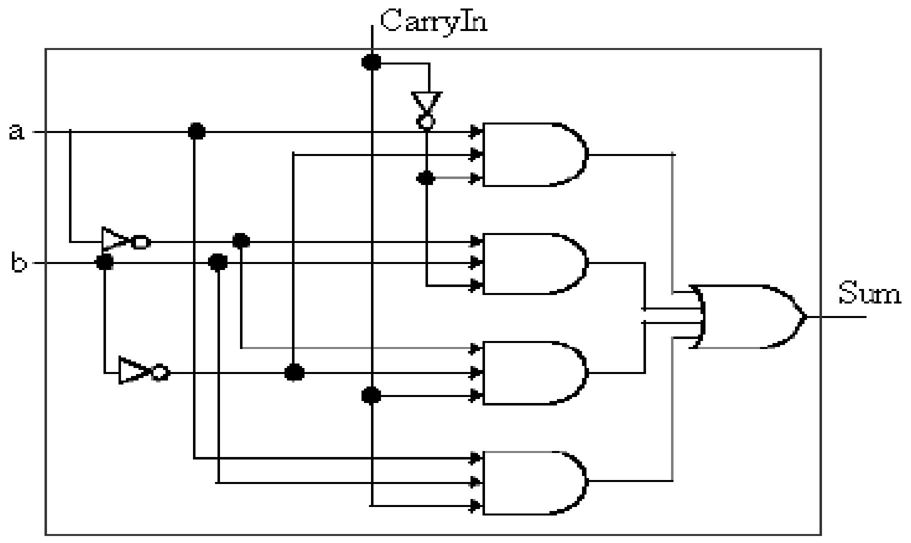
    10001 1111

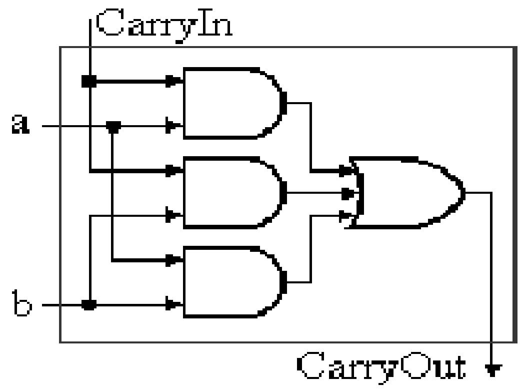
Example

- Consider a two 4 bit Negative number
- -9 and -6 = 11001 + 10110 = 101111
  - 1's complement - to avoid overflow
- Consider a two 8 bit positive number
- -83and -24

```
 11000 0011
 10010 0100
101010 0111
```

Assessment





Carryout = (b.CarryIn)+(a.CarryIn) +(a.b)
Sum = (a.b’.CarryIn’)+ (a’.b.CarryIn’)+ (a’.b’.CarryIn)+ (a.b.CarryIn)

Thank You