Computer Organization
Arithmetic Logic Unit

Department of Computer Science
Missouri University of Science & Technology
hurson@mst.edu

Database System Architecture

Note, this unit will be covered in five lectures. In case you finish it earlier, then you have the following options:

1) Take the early test and start CS3889.module5

2) Study the supplement module (supplement CS3889.module4)

3) Act as a helper to help other students in studying CS3889.module4

Note, options 2 and 3 have extra credits as noted in course outline.

Database System Architecture

Glossary of prerequisite topics

Enforcement of background

Familiar with the topics? →No→ Review CS3889.module4.background

Yes

Take Test

Pass? →No→ Remedial action

Yes

At the end give a test, record the score, and impose remedial action if not successful

Glossary of topics

Current Module

Familiar with the topics? →No→ Take the Module

Yes

Take Test

Pass? →No→

Yes

Options

Study next module?

Lead a group of students in this module (extra credits)?

Study more advanced related topics (extra credits)?

Extra Curricular activities

◆**Arithmetic Logic Unit**

✹It is a functional box designed to perform "basic" arithmetic, logic, and shift operations on the data.

✹Implementation of the basic operations such as logic, program control, and data transfer operations are easier than arithmetic and I/O operations. Therefore, in this section we concentrate on arithmetic operations.

Computer Organization

◆ **Arithmetic Logic Unit**

✹ For a simple machine, the ALU should at least be able to perform operations such as:

- add
- increment
- subtract
- decrement
- ●
- ●
- ●

✹ A simple ALU is basically an adder and some control circuits augmented by special circuits to carry out the logic and shift operations .

Computer Organization

◆ **Arithmetic Logic Unit**

✹ An ALU can be of three types:

- Serial
- Parallel
- Functional (Modular)
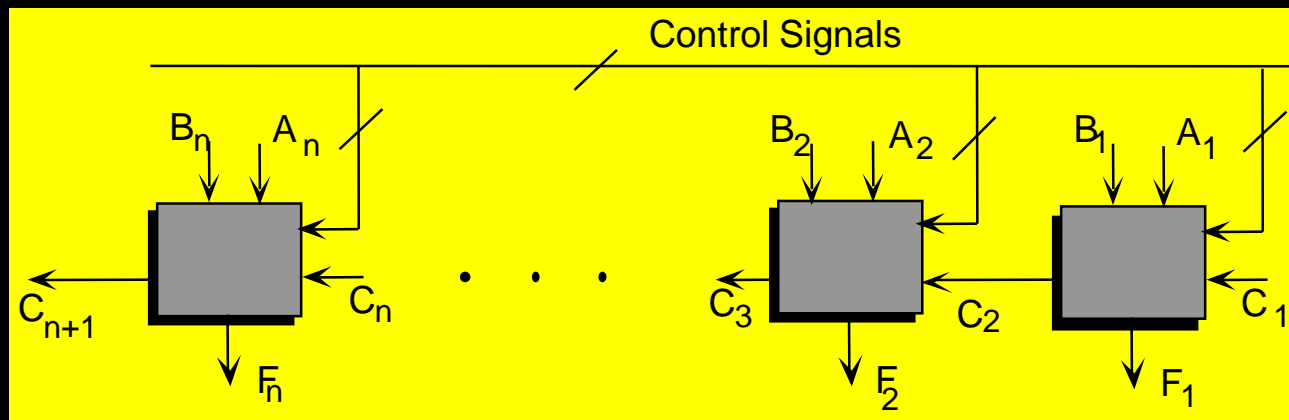
Computer Organization

◆ **Arithmetic Logic Unit**

✸ Similar to the definition of serial and parallel adders, one can define serial and parallel ALUs.

✸ In a serial ALU during each clock pulse one bit of operand(s) participates in the operation.

✸ In a parallel ALU operation on all the bits of operand(s) is initiated simultaneously.

✸ In a simple word a parallel ALU can be looked at as a cascade of identical units forming a one dimensional array of cells.

Computer Organization

◆**Arithmetic Logic Unit –** Parallel ALU

  ✹ ALU operation is determined by the control signals.

  ✹ In a very simple form, the bit-pattern of the control signals is determined by the operation code.

  ✹ In a parallel ALU, one needs to determine the design of a unit and then replicate it.
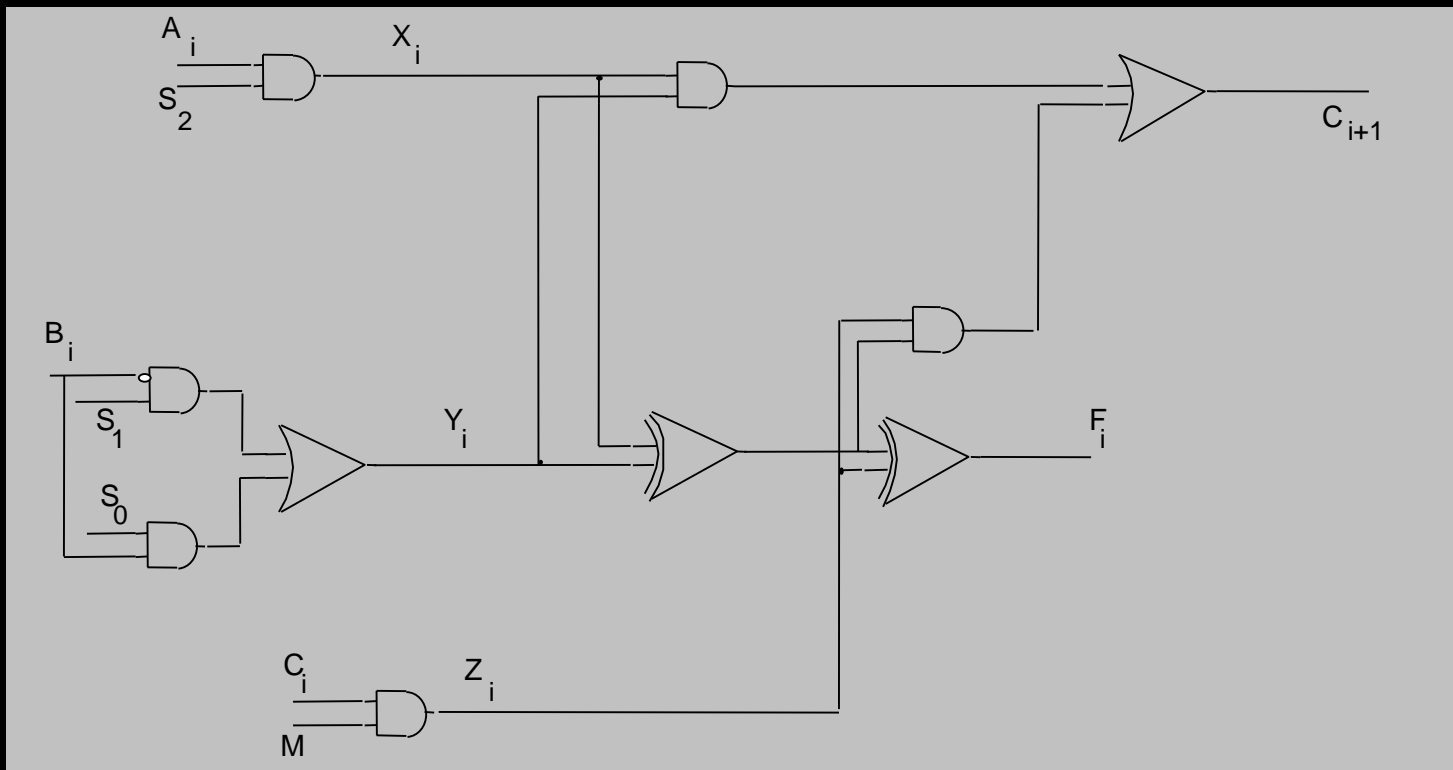
Computer Organization

◆**Arithmetic Logic Unit –** Parallel ALU (Example)

✱ S_2, S_1, S_0, and M are the control signals.

✱ A_i, B_i, and C_i are the operand bits and carry-in, respectively.

✱ F_i and C_{i+1} are the result bit and carry-out, respectively.

◆**Arithmetic Logic Unit –** Parallel ALU (Example)

Computer Organization

◆**Arithmetic Logic Unit –** Parallel ALU (Example)

✹ The function of each stage can be defined as:

$$F_i = X_i \oplus Y_i \oplus Z_i$$

$$C_{i+1} = X_iY_i + (X_i \oplus Y_i)Z_i = X_iY_i + X_iZ_i + Y_iZ_i$$

✹ By appropriate setting of the control signals one can initiate a variety of the operations.

Computer Organization

◆**Arithmetic Logic Unit –** Parallel ALU (Example)

✳ For example:

$$M = 0$$
$$S_2 = 1$$
$$S_1 = 1 \qquad \Rightarrow F \quad \leftarrow \overline{A}$$
$$S_0 = 1$$

$$M = 1$$
$$S_2 = 1$$
$$S_1 = 1 \qquad \Rightarrow F \quad \leftarrow A - 1$$
$$S_0 = 1$$
$$C_1 = 0$$

Computer Organization

◆ Arithmetic Logic Unit

✸ As discussed before a parallel ALU offers a higher speed relative to a serial ALU.

✸ How can one improve the performance (speed) of ALU further? In other words; is it possible to build (design) an ALU faster than a parallel ALU?
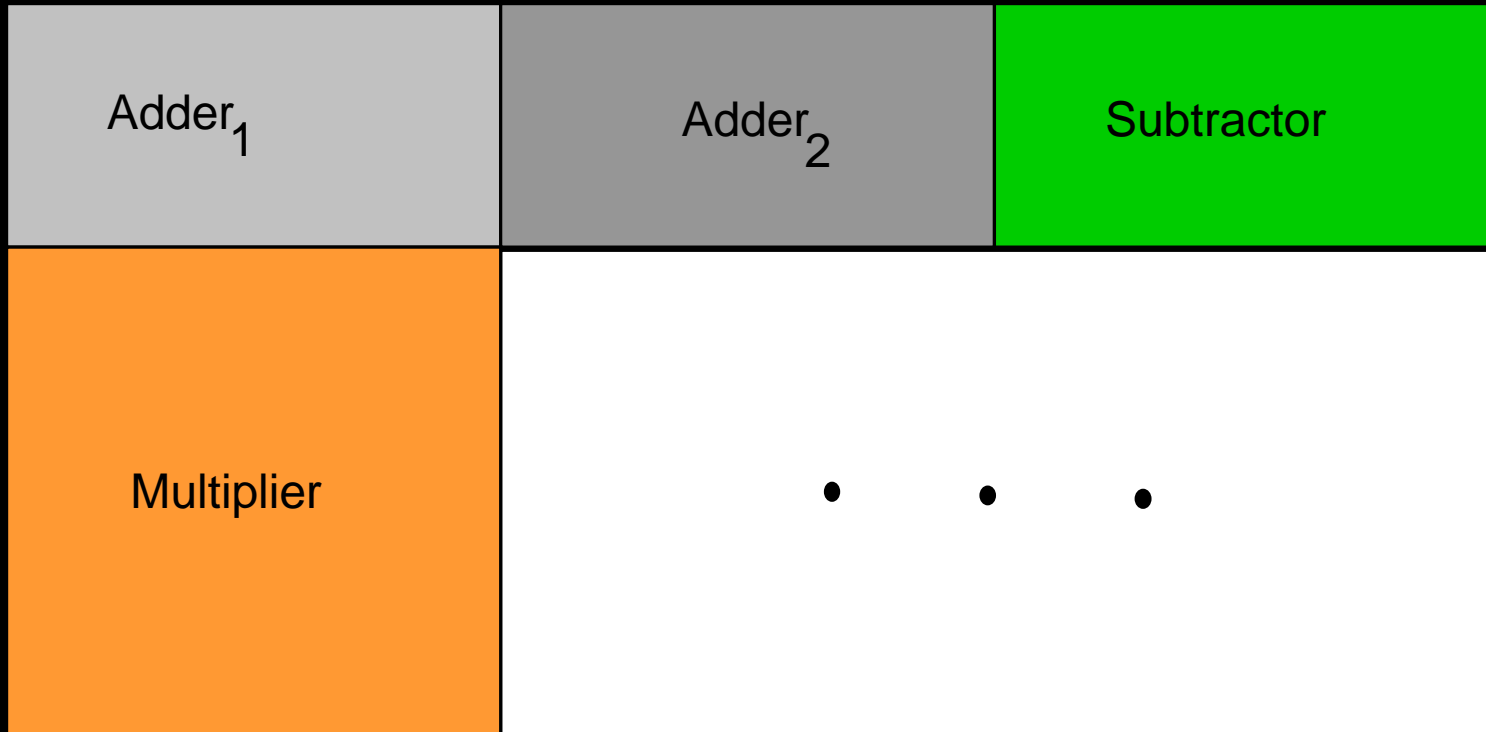
Computer Organization

◆ **Arithmetic Logic Unit** ─ Functional (modular) ALU

✹ In this model, ALU is a collection of independent units each tailored (specialized) for a specific operation. As a result, execution of independent operations can be overlapped.

✹ This approach allows an additional degree of concurrency relative to a parallel ALU, since it allows several operations to be performed on data simultaneously.

✹ However, this speed improvement comes at the expense of extra overhead which is needed to detect data independent operations.

Computer Organization

◆ **Arithmetic Logic Unit –** Functional (modular) ALU

Adder$_1$	Adder$_2$	Subtractor
Multiplier	• • •	

Computer Organization

◆**Arithmetic Logic Unit** – Question

✹ Is it possible to improve the performance of an ALU further?

✹ Naturally, we can improve performance (physical speed) by taking advantage of the advances in technology.

✹ How can we improve the logical speed of the ALU further?

✹ In a functional ALU, is it possible to devise algorithms which allow one to improve the performance of the basic operations?

✹ If this is a valid direction, then the question of "How to design a fast ALU?" will change to "How to design a fast adder, a fast multiplier, ...?"

Computer Organization

◆**Arithmetic Logic Unit –**   Arithmetic Algorithm

✹ The goal is to develop a set of algorithms which allow the execution of basic arithmetic operations on data values presented in different formats and notations (i.e., fixed point, floating point, signed numbers).

✹ In general fixed point operations are easier and faster than floating point operations.

✹ Signed numbers could be in signed magnitude or signed complement format.

Computer Organization

◆ **Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (Direct Comparison)

✱ Two binary bits (a, b) are equal iff:

$$x = \left(a \wedge b\right) \vee \left(\overline{a} \wedge \overline{b}\right) = 1$$

✱ Consequently two registers of n-bits are equal iff:

$x = x_{n-1} \wedge x_{n-2} \wedge \ldots \wedge x_0 = 1$ where $x_i = A_i \wedge B_i$ $0 \leq i \leq n-1$

✱ In other words

$$A = B \Rightarrow x = \bigwedge_{i=0}^{n-1} x_i = 1$$

◆**Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (Direct Comparison)

$$A > B \Rightarrow A_{n-1}\overline{B}_{n-1} + x_{n-1}A_{n-2}\overline{B}_{n-2} + \ldots + x_{n-1}x_{n-2} \ldots x_1 A_0 \overline{B}_0 = 1$$

$$A < B \Rightarrow \overline{A}_{n-1}B_{n-1} + x_{n-1}\overline{A}_{n-2}B_{n-2} + \ldots + x_{n-1}x_{n-2} \ldots x_1 \overline{A}_0 B_0 = 1$$

Computer Organization

◆**Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (Direct Comparison)

✸**Example:**

A=	1	1	0	0	1	1	1
B=	1	1	0	1	0	0	0
	x_6	x_5	x_4	x_3	x_2	x_1	x_0
	1	1	1	0	0	0	0

$x_6 \wedge x_5 \wedge x_4 \wedge x_3 \wedge x_2 \wedge x_1 \wedge x_0 = 1 \wedge 1 \wedge 1 \wedge 0 \wedge 0 \wedge 0 \wedge 0 = 0$

$\Rightarrow A \neq B$

Computer Organization

◆ **Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (Direct Comparison)

$$x6 \wedge x5 \wedge x4 \wedge x3 \wedge x2 \wedge x1 \wedge x0 = 1 \wedge 1 \wedge 1 \wedge 0 \wedge 0 \wedge 0 \wedge 0 = 0$$
$$\Rightarrow A \neq B$$

$$A_6\overline{B_6} + x_6A_5\overline{B_5} + x_6x_5A_4\overline{B_4} + x_6x_5x_4A_3\overline{B_3} + x_6x_5x_4x_3A_2\overline{B_2} +$$
$$x_6x_5x_4x_3x_2A_1\overline{B_1} + x_6x_5x_4x_3x_2x_1A_0\overline{B_0} = 0+0+0+0+0+0+0 = 0$$
$$\Rightarrow A \not> B$$

$$B_6\overline{A_6} + x_6B_5\overline{A_5} + x_6x_5B_4\overline{A_4} + x_6x_5x_4B_3\overline{A_3} + x_6x_5x_4x_3B_2\overline{A_2} +$$
$$x_6x_5x_4x_3x_2B_1\overline{A_1} + x_6x_5x_4x_3x_2x_1B_0\overline{A_0} = 0+0+0+1+0+0+0 = 1$$
$$\Rightarrow A < B$$

Computer Organization

◆ **Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (Direct Subtraction)

✸ Assume E is the borrow-out of a parallel subtractor then:

if $A \geq B$ then $E = 0$

if $A < B$ then $E = 1$

if $A = B$ then $E = 0$ and result all 0s

✸ In other words, to check the relative magnitude of two unsigned numbers say, A and B. Perform A-B and then check the borrow-out. The relative magnitude of A and B is determined according to the above relationships.

Computer Organization

◆**Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (2^s Complement Addition)

　✴Assume E is the carry-out of a parallel adder then:

　　if A < B then E = 0

　　if A ≥ B then E = 1

　　if A = B then E = 1 and result all 0s

Computer Organization

◆**Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers

(2^s Complement Addition)

✳ **Proof:**

$$A - B = A + \widetilde{B} = A + 2^n - B = 2^n + A - B$$

✳ if $A \geq B \Rightarrow A - B \geq 0 \Rightarrow 2^n + A - B \geq 2^n \Rightarrow E = 1$

✳ if $A < B \Rightarrow A - B < 0 \Rightarrow 2^n + A - B < 2^n \Rightarrow E = 0$

✳ if $A = B \Rightarrow A - B = 0 \Rightarrow 2^n + A - B = 2^n \Rightarrow E = 1$ and result all 0^s

✳ To determine the relative magnitude of two unsigned numbers, perform $\boxed{A + \widetilde{B}}$ and then check the carry-out.

Computer Organization

◆**Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers

(1ˢ Complement Addition)

✸Assume E is the carry-out of a parallel adder then:

if A ≤ B then E = 0

if A = B then E=0 and result all 1s

if A > B then E =1

✸To determine the relative magnitude of two unsigned numbers, perform $A + \overline{B}$ and then check the carry-out.
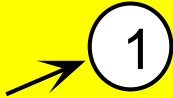
◆**Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (Direct Subtraction)

✴Example

$$
\begin{array}{ll}
A & 1100111 \\
-\ B & \underline{1101000} \\
& 1111111 \quad\quad \Rightarrow E = 1 \Rightarrow A < B \\
& \textcircled{1} \\
\text{Borrow-out}
\end{array}
$$

◆**Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (2^s Complement Addition)

✴Example

$$
\begin{array}{ll}
\text{A} & 1100111 \\
+\ \tilde{\text{B}} & \underline{0011000} \\
& 1111111 \qquad \Rightarrow \text{E} = 0 \quad \Rightarrow \text{A} < \text{B} \\
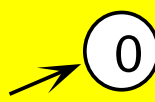\end{array}
$$
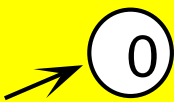
⓪

Carry-out

Computer Organization

◆**Arithmetic Logic Unit** – Comparison of 2-Unsigned Numbers (1^s Complement Addition)

✳ Example

$$
\begin{array}{ll}
\overline{A} & 1100111 \\
+\overline{B} & \underline{0010111} \\
& 1111110 \qquad \Rightarrow E = 0 \quad \Rightarrow A < B \\
& \textcircled{0}
\end{array}
$$

Carry-out

28

Computer Organization

◆Questions

✹Prove the aforementioned formulas for the 1s complement addition operation.

✹How can one determine the relative magnitude of signed numbers?

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (2^s Complement Numbers)

✸Add numbers including the sign bits. If there is a carry-out of the last digit, then disregard it.

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (1^s

Complement Numbers)

✸Add numbers including the sign bits. If there is a carry-out then increment the result by 1 (wrap around the carry).

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✹ Addition (Subtraction):

- When the signs of numbers (say A and B) are identical (different) add the two magnitudes. Sign of the result is the same as A.

- When the signs of A and B are different (identical), compare the magnitudes and subtract the smaller magnitude from the larger one. Sign of the result is the same as the sign of the larger magnitude.

- If result zero then set the sign of the result to 0.

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✳ How to reduce hardware requirements of the proposed algorithm?

- Subtraction can be converted into addition.
- Magnitudes can be compared using an adder.

Computer Organization

◆ **Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)
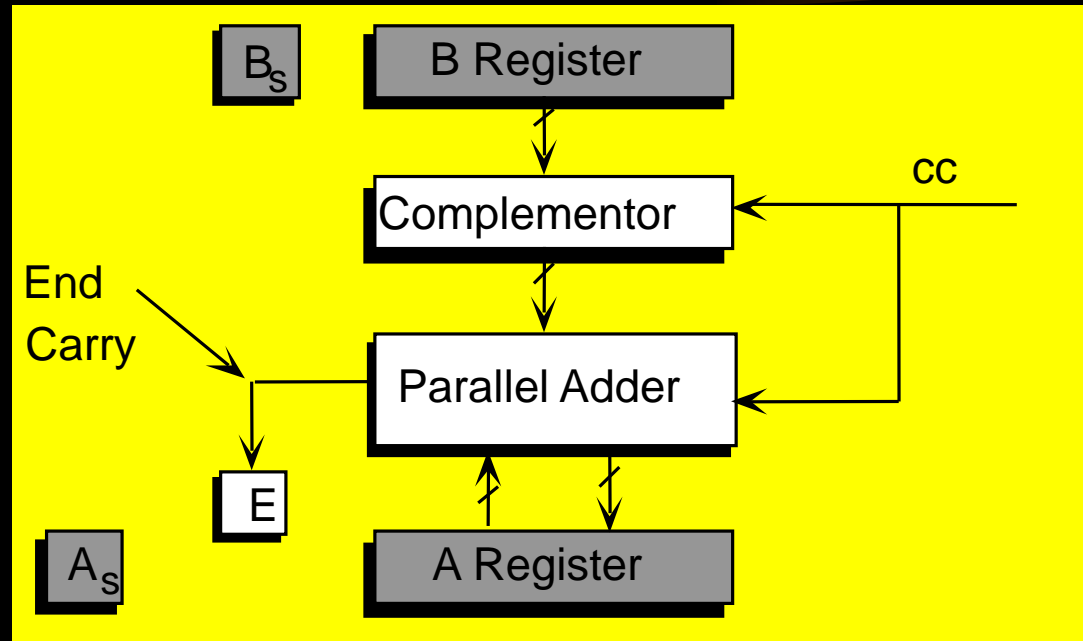
- $cc = 0$ if (addition and $A_s = B_s$) or (subtraction and $A_s \neq B_s$)
- $cc = 1$ if (addition and $A_s \neq B_s$) or (subtraction and $A_s = B_s$)

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✴Example

```
 A (+29)              0011101
               ⇒                    ⇒  Same signs and addition
+B (+14)           +  0001110


                     011101
                     001110              Add the magnitudes.
          E
           ╲ 101011
          0                             E = 0


      A  =  0101011    Final result (+43).
```

◆**Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✸Example

A (-14)	101110	
+B (-7) $\Rightarrow$	+ 100111	$\Rightarrow$ Same signs and addition
	01110	
	00111	Add the magnitudes.
E		
0 ← 10101		E = 0
A = 110101		Final result (-21).

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✸Example

A (-14)		101110	
-B (-7)	$\Rightarrow$	- 100111	$\Rightarrow$ Same signs and subtract

$$
\begin{array}{l}
01110 \\
\underline{11001} \\
\end{array}
$$

E

1 $\leftarrow$ 00111   E = 1, A>B

Add 2^s complete of B to A.

A = 100111   Final result (-7).

Computer Organization

◆ **Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✸ Example

A (+14) $\Rightarrow$ 001110
-B (+20)   - 010100 $\Rightarrow$ Same signs and subtract

01110
01100   Add 2^s complete of B to A.

E
0 ← 11010   E = 0, A<B, take 2^s complement of result

A = 100110   Final result (-6).

◆ **Arithmetic Logic Unit** – Addition Algorithm
(Signed magnitude numbers)

✸ Example

A (-14) $\Rightarrow$ 101110
-B (-14) $\Rightarrow$ - 101110 $\Rightarrow$ Same signs and subtract

                              01110
                              10010   Add 2^s complete of B to A.
              E
              1 ← 00000   E = 1, and result zero

          A = 000000   Final result (+0).

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✸Example

$$
\begin{array}{l}
\text{A} \quad\quad\quad 111011 \\
\text{+B} \quad \Rightarrow \quad + \underline{\,000111\,} \quad\quad \Rightarrow \text{Different signs and addition}
\end{array}
$$

$$
\begin{array}{l}
\quad\quad 11011 \\
\quad\quad \underline{11001} \quad\quad\quad \text{Take } 2^{s} \text{ complement of B and add.} \\
\quad\quad \overset{E}{\,}\\
1 \;\swarrow 10100 \quad\quad\quad E = 1 \text{ and result} \quad\quad \neq 0 \;\Rightarrow A > B
\end{array}
$$

$$
\text{A} \; = 110100 \quad\quad\quad \text{Final result.}
$$

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✸Example

A   010111

+B   101011 $\Rightarrow$ Different signs and addition

10111

10101   Take 2^s complement of B and add.

E

1 ← 01100

E = 1 and result $\neq 0 \Rightarrow A > B$

A = 001100   Final result.

Computer Organization

◆**Arithmetic Logic Unit** – Addition Algorithm (Signed magnitude numbers)

✹Example

A	0 0 0 1 1 1	Same signs and subtraction.
- B	0 0 1 0 1 1	

$$0 0 1 1 1$$
$$1 0 1 0 1$$

Take 2^s complement of B and add.

$\overset{E}{\underset{0}{\swarrow}} 1 1 1 0 0$

$E = 0 \Rightarrow B > A$

A = 1 0 0 1 0 0   Final result, take 2^s complement of result.

Computer Organization

◆Questions

✹What is an overflow?

✹How can we detect overflow for:

- Signed magnitude addition?
- 1^s complement addition?
- 2^s complement addition?

Computer Organization

◆**Arithmetic Logic Unit –** Addition Overflow

✸Overflow occurs when two numbers of *n* digits each are added and the sum occupies *n+1* digits.

✸An overflow cannot occur after an addition if one number is positive and the other is negative.

✸In general, overflow can only occur when adding numbers of the same sign or subtracting numbers of different signs.

Computer Organization

◆ **Arithmetic Logic Unit** – Addition Overflow (Signed magnitude numbers)

✴ Detection of overflow in signed magnitude addition is simple. Carry-out of the magnitude bits represents an overflow:
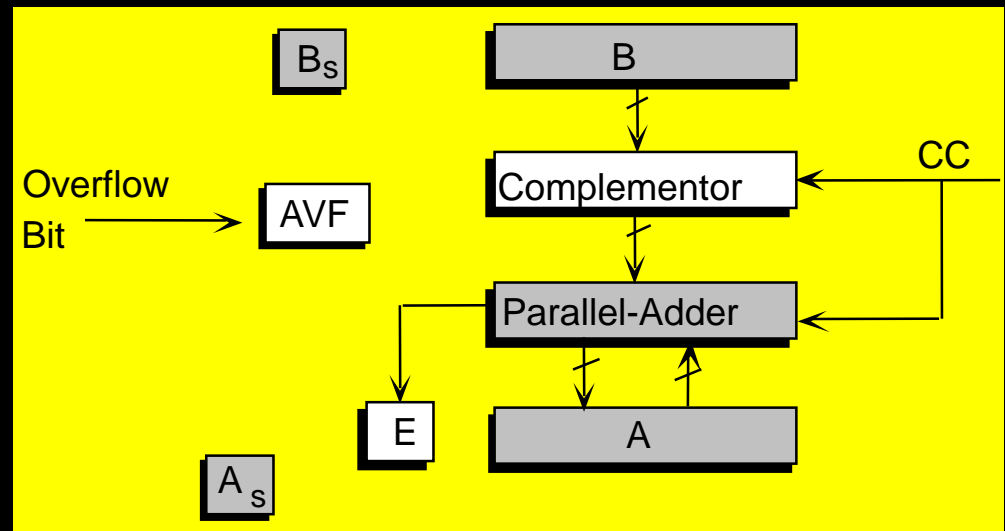
If (cc = 0 and E = 1) then AVF $\Leftarrow$ 1 otherwise AVF $\Leftarrow$ 0

Computer Organization

◆ **Arithmetic Logic Unit** – Addition Overflow (2^s Complement Numbers)

✴ Overflow is occurred when adding numbers of the same sign results in a number of a different sign.
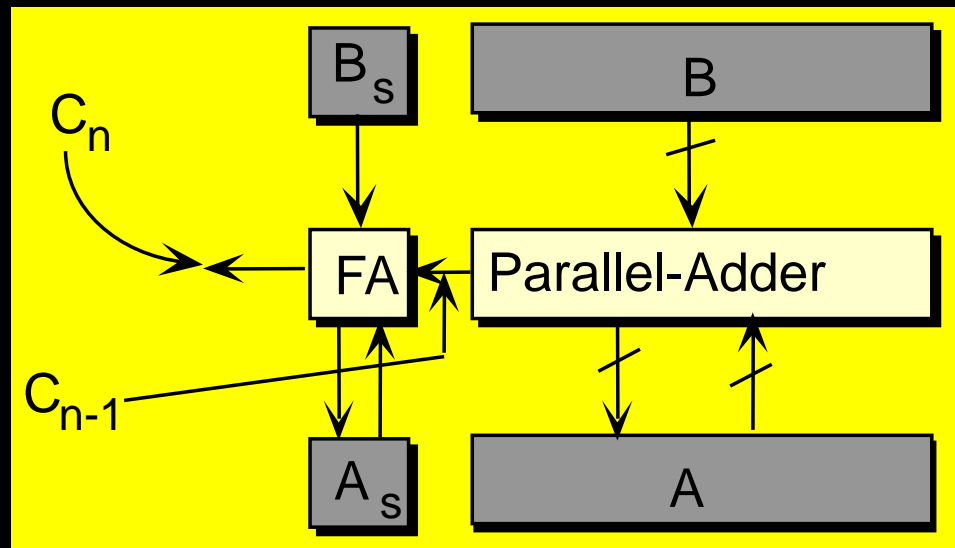
✴ Overflow occurs if $C_n \oplus C_{n-1} = 1$

46

Computer Organization

◆**Arithmetic Logic Unit** – Multiplication Algorithm

✹Multiplication can be performed as a sequence of additions.

✹B * Q $\Rightarrow$ add B, Q times

✹Two general approaches have been adapted for multiplication.

● Software approach

● Hardware approach

Computer Organization

◆**Arithmetic Logic Unit** – Multiplication Algorithm

✹ Software approach

AC $\Leftarrow$ 0;

For i = 1 to Q do;

AC $\Leftarrow$ (AC) + (B);

End;

✹ Time complexity of this algorithm is 0(m) where m is the magnitude of Q.

Computer Organization

◆**Arithmetic Logic Unit** – Multiplication Algorithm

✺ Software approach

● Time complexity of the algorithm can be improved if the number of iterations is based on the multiplier length:

$AC \Leftarrow 0$;

For i = 1 to n Do;

   if $Q_1 = 1$ then $C_{out} AC \Leftarrow (AC) + (B)$;

$AC_n...AC_2AC_1Q_n...Q_2Q_1 \Leftarrow C_{out}AC_n...AC_2AC_1Q_n...Q_2$;

End;

Computer Organization

◆ **Arithmetic Logic Unit** – Multiplication Algorithm

✹ Hardware Approach

- The simplest hardware approach is based on the previous algorithm - e.g., add and shift.

  - In each iteration the least-significant bit of multiplier (Q) is checked;

  - If one, then B is added to the accumulator and the contents of accumulator and Q is shifted right one position.

  - If zero, just shift accumulator and Q to the right.

Computer Organization

◆**Arithmetic Logic Unit –** Multiplication Algorithm (Signed magnitude numbers)

✹ Hardware Configuration

● A double register (AC and Q) holds the final result.

● Sign of the result is the EX-OR of the signs of multiplier and multiplicand.
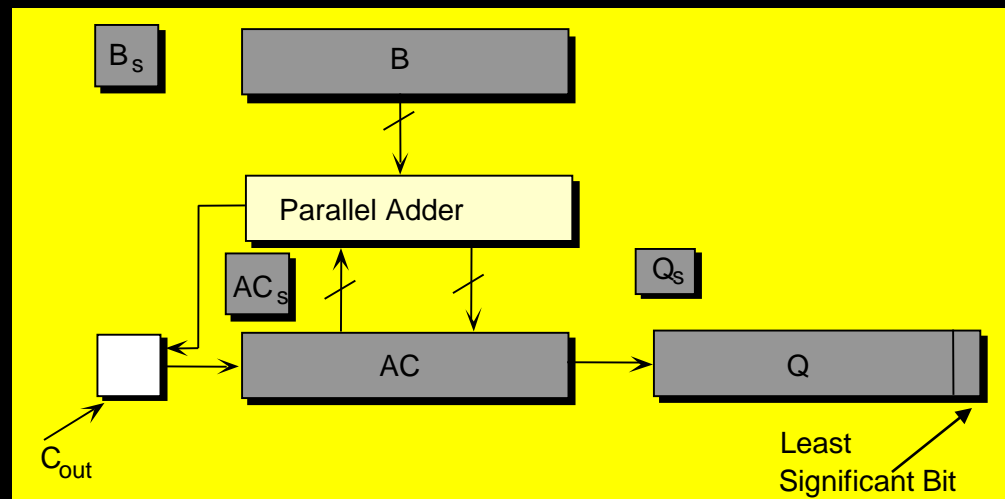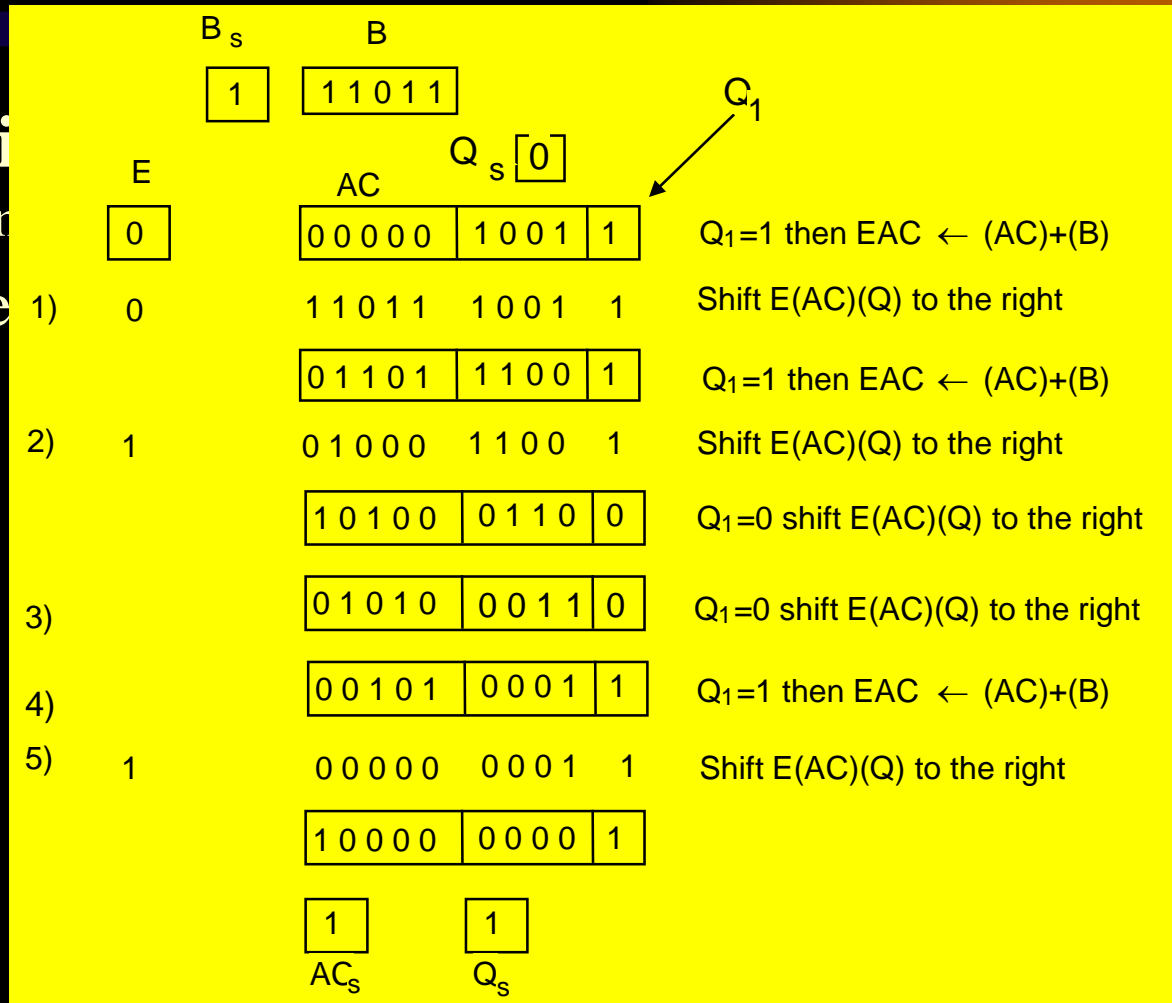
◆**Arithmeti** ... ed

magnitude nur

★Example

	B_s	B
	1	1 1 0 1 1

Q_s [0] Q_1

E	AC		
0	0 0 0 0 0	1 0 0 1	1	Q_1=1 then EAC ← (AC)+(B)

1) 0   1 1 0 1 1   1 0 0 1   1   Shift E(AC)(Q) to the right

| | 0 1 1 0 1 | 1 1 0 0 | 1 | Q_1=1 then EAC ← (AC)+(B)

2) 1   0 1 0 0 0   1 1 0 0   1   Shift E(AC)(Q) to the right

| | 1 0 1 0 0 | 0 1 1 0 | 0 | Q_1=0 shift E(AC)(Q) to the right

3) | | 0 1 0 1 0 | 0 0 1 1 | 0 | Q_1=0 shift E(AC)(Q) to the right

4) | | 0 0 1 0 1 | 0 0 0 1 | 1 | Q_1=1 then EAC ← (AC)+(B)

5) 1   0 0 0 0 0   0 0 0 1   1   Shift E(AC)(Q) to the right

| | 1 0 0 0 0 | 0 0 0 0 | 1 |

1	1
AC_s	Q_s

Computer Organization

◆**Arithmetic Logic Unit –** Multiplication Algorithm (2ˢ Complement numbers)

   ✸Pre-Post Complementation

      ●Convert negative numbers to positive numbers.

      ●Apply "add and shift" algorithm.

      ●Convert the result - if necessary.

Computer Organization

◆**Arithmetic   Logic   Unit –** Multiplication Algorithm (2^s Complement numbers)

✹Extension of "add and shift"

- Convert the multiplier to a positive number - i.e., if multiplier is negative then multiply the multiplier and the multiplicand by -1.

- Apply a modified "add and shift" algorithm in which addition and shift operations are done for 2^s complement numbers.

Computer Organization

◆ **Arithmetic Logic Unit** – Multiplication Algorithm (2^s Complement numbers)

✳ Booth's Algorithm

● Instead of checking one bit of the multiplier at a time, check two bits at a time and take proper actions according to the following table:

00	no action, shift right.
01	add multiplicand, shift right.
10	subtract multiplicand, shift right.
11	no action, shift right.

Computer Organization

◆**Arithmetic Logic Unit** – Multiplication Algorithm (2^s Complement numbers)

✹ Booth's Algorithm

- It is suitable for 2^s complement numbers.
- On average, Booth's Algorithm is faster than "add and shift".
- Always, extend the multiplier by a zero on the right.

◆**Arit**

Algori

✦ Bo

Extension

1101101		

0000000	0000111	0

0010011 1st 2 bits are 10   ⇌ subtract (add 2^s complement)

0010011   0̄000111 0   Shift right
0001001 1   0̄00011 1 1st 2 bits are 11   ⇒ shift right
0000100 11   0̄0001 1 1st 2 bits are 11   ⇒ shift right
0000010 011   0000 1 1st 2 bits are 01   ⇒ add
1101101

1101111 011   0̄000 1   Shift right
1110111 1011   0̄00 0 1st 2 bits are 00   ⇒ shift right
1111011 11011   00̄ 0 1st 2 bits are 00   ⇒ shift right
1111101 111011   0̄ 0 1st 2 bits are 00   ⇒ shift right

1111110	1111011	0

B * Q

Computer Organization

◆**Questions**

✶Why is Booth's algorithm faster than "add and shift" algorithm (in general)?

✶Why does Booth's Algorithm work?

Computer Organization

◆**Arithmetic Logic Unit –** Division Algorithm

✹ Similar to multiplication, one can develop a software routine which performs division as a sequence of subtractions. Naturally, such an algorithm is very inefficient and slow.

Computer Organization

◆ **Arithmetic Logic Unit –** Division Algorithm

✸ Similar to the pencil and paper routine, one can develop an algorithm which performs division as a sequence of Compare, Shift, and Subtract operations.

✸ One should note that division in a binary system is much simpler than the division in decimal system, since the quotient digits are either 0 or 1.

Computer Organization

◆**Arithmetic Logic Unit** – Division Algorithm

✶To minimize the hardware requirements for division, we should remember that:

- Comparison can be performed via arithmetic operation(s).

- Subtraction can be performed via complement-addition.

✶In other words; division requires almost the same hardware modules as multiplication does.

◆Hardware configuration for multiplication
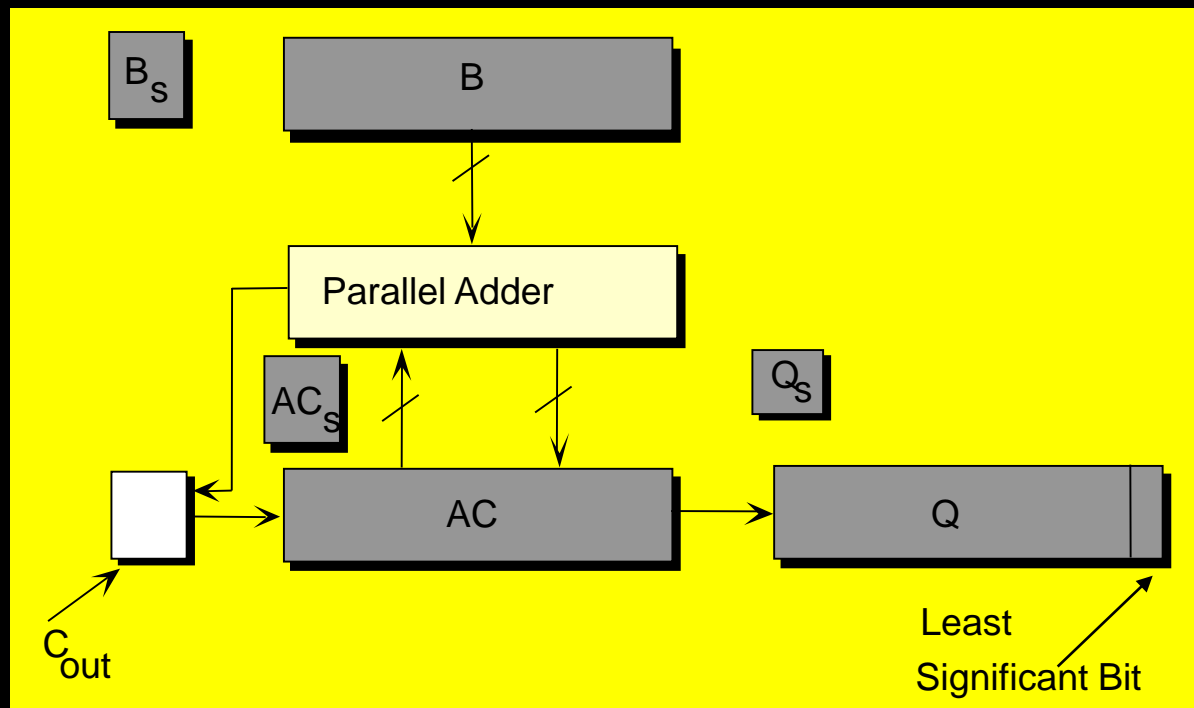
Computer Organization

◆**Arithmetic Logic Unit –** Division Algorithm (Signed magnitude numbers)

✳ Division can be carried out as a sequence of n (n is the length of divisor) iterations.

✳ Dividend is a double register.

✳ One bit of the quotient is generated in each iteration.

✳ At the end of the operation, the quotient is in the 1st half part of the double register (low-order part), and remainder is in the 2nd half part.

✳ Sign of the quotient is the XOR of the signs of dividend and divisor.

✳ Sign of the remainder is the same as the sign of the dividend.

◆**Arithmetic Logic Unit** – Division Algorithm (Example)

Extended → E   A   Q   B
Sign Bit
              0   0000   1011   0011

	E	A	Q		
0	0000	1011	Shift left EAQ.
0	0001	011 ☐	Subtract B.
1	1101		
1	1110	011	E=1 $\Rightarrow$ Q $\leftarrow$ 0.
0	0011		Restore A.
0	0001	0110	
0	0010	110	Shift left EAQ. Subtract B.
1	1101		
1	1111	110	E=1 $\Rightarrow$ Q $\leftarrow$ 0.
0	0011		Restore A.
0	0010	1100	
0	0101	100	Shift left EAQ. Subtract B.
1	1101		
0	0010	100	E=0 $\Rightarrow$ Q $\leftarrow$ 1.
0	0010	1001	
0	0101	001	Shift left EAQ. Subtract B.
1	1101		
0	0010	001	E=0 $\Rightarrow$ Q $\leftarrow$ 1.
0	0010	0011	
	R	Q	

Computer Organization

◆**Arithmetic Logic Unit** – Division Algorithm (Example)

✴ Restoring Method

	E	A	Q	Q_n	B	
Extended Sign Bit ———	0	0000	1011		0011	
	0	0000	1011			Shift left EAQ.
	0	0001	011 ☐			Subtract B.
	1	1101				
	1	1110	011 ☐			$E=1 \Rightarrow Q_1 \leftarrow 0$.
	0	0011				Restore A.
	0	0001	0110			Shift left EAQ.
	0	0010	110 ☐			Subtract B.
	1	1101				
	1	1111	110 ☐			$E=1 \Rightarrow Q_1 \leftarrow 0$.
	0	0011				Restore A.
	0	0010	1100			Shift left EAQ.
	0	0101	100 ☐			Subtract B.
	1	1101				
	0	0010	100 ☐			$E=0 \Rightarrow Q_1 \leftarrow 1$.
	0	0010	1001			Shift left EAQ.
	0	0101	001 ☐			Subtract B.
	1	1101				
	0	0010	001 ☐			$E=0 \Rightarrow Q_1 \leftarrow 1$.
	0	0010	0011			
		R	Q			

Computer Organization

◆ **Arithmetic Logic Unit –** Division Algorithm (Divide Overflow)

✹ In the division algorithm for signed magnitude numbers, if the 1st half part of dividend is larger than or equal to the divisor, then the quotient overflows to the remainder part. This phenomenon is called divide overflow. As a result, a wrong answer will be generated.

✹ To avoid divide overflow and also due to the fact that division is a time consuming operation, the divide overflow condition is checked at the beginning of the operation. In case of divide overflow, proper actions will be taken (in the form of say an interrupt).

◆ **Arithmetic Logic Unit –** Methods of Division

✹ There are several different algorithms for division:

- Restoring Method
- Non-Restoring Method
- Direct Comparison

Computer Organization

◆**Arithmetic Logic Unit –** Methods of Division

✹Restoring Method: In the restoring method, the contents of the partial remainder is restored whenever it is detected that the divisor is larger than the partial remainder.

Computer Organization

◆**Arithmetic Logic Unit –** Methods of Division

✸ Non-Restoring Method: In non-restoring method, if it is detected that the divisor is larger than the partial remainder, its contents are not restored.

✸ However, in the next iteration, instead of subtracting the divisor from the partial remainder, it will be added to the partial remainder.

Computer Organization

◆**Arithmetic Logic Unit –** Methods of Division

✸Direct Comparison: divisor is compared against partial remainder. If it is smaller than or equal to partial remainder, it will be subtracted and quotient digit is set to 1. Otherwise, just the quotient digit is set to zero.

Computer Organization

◆**Arithmetic Logic Unit – Methods of Division**

✸ Naturally, restoring and non-restoring techniques are equivalent.

✸ Consider the consecutive sequence of iterations for both restoring and non-restoring techniques (next slide).

✸ As can be seen, both schemes generate the same value as the partial remainder. However, the non-restoring technique should be faster than the restoring method since it requires fewer number of operations.

Computer Organization

◆**Arithmetic Logic Unit –** Methods of Division

Restoring

Subtract in the second
Iteration

$(2 ((A-B)+B) \ -B) \ = 2A-B$

Subtract, Restore, and Shift
in the first iteration

Non-Restoring

Addition in the second Iteration

$2 (A-B) \ +B = 2A-B$

Subtraction and shift
in the first iteration

◆ Arithmetic Logic Unit – Division Algorithm (Example)

✹ Restoring Method

	E	A	Q	Q_n	B	
Extended Sign Bit ———	0	0000	1011		0011	
	0	0000	1011			Shift left EAQ.
	0	0001	011 ☐			Subtract B.
	1	1101				
	1	1110	011 ☐			$E=1 \Rightarrow Q_1 \leftarrow 0$.
	0	0011				Restore A.
	0	0001	0110			Shift left EAQ.
	0	0010	110 ☐			Subtract B.
	1	1101				
	1	1111	110 ☐			$E=1 \Rightarrow Q_1 \leftarrow 0$.
	0	0011				Restore A.
	0	0010	1100			Shift left EAQ.
	0	0101	100 ☐			Subtract B.
	1	1101				
	0	0010	100 ☐			$E=0 \Rightarrow Q_1 \leftarrow 1$.
	0	0010	1001			Shift left EAQ.
	0	0101	001 ☐			Subtract B.
	1	1101				
	0	0010	001 ☐			$E=0 \Rightarrow Q_1 \leftarrow 1$.
	0	0010	0011			
		R	Q			

73

Computer Organization

◆**Arithmetic Logic Unit –** Division Algorithm (Example)

✳ Non-restoring Method

E	A	Q	B
0	0000	1011	0011
0	0000	1011	Shift left EAQ.
0	0001	011☐	Subtract B.
1	1101		
1	1110	011☐	E=1 ⇒ Q_1 ← 0.
1	1110	0110	Shift left EAQ.
1	1100	110☐	Add B.
0	0011		
1	1111	110☐	E=1 ⇒ Q_1 ← 0.
1	1111	1100	Shift left EAQ.
1	1111	100☐	Add B.
0	0011		
0	0010	100☐	E=0 ⇒ Q_1 ← 1.
0	0010	1001	Shift left EAQ.
0	0101	001☐	Subtract B.
1	1101		
0	0010	0011	E=0 ⇒ Q_1 ← 1.
0	0010	0011	
	R	Q	

Computer Organization

◆**Arithmetic Logic Unit –** Floating Point Numbers

✸ A number N in base r can be represented in many ways. For example, decimal number 2.945 can be represented as:

.2945 * 10^{+1}

29.45 * 10^{-1}

.

.

.

✸ A scientific or floating point format of a number is a unique representation in which a number is represented as the multiple of 2 parts:

a power of the base (called exponent)* a fraction (called mantissa)

✸ For example, $+16.23 = +.1623 * 10^{+2}$

Computer Organization

◆ **Arithmetic Logic Unit –** Floating Point Numbers

✹ Such a representation in the computer requires two registers, one to represent the exponent and one to represent the mantissa.

✹ Note that each part is a signed number which could be represented in signed magnitude or signed complement format.

✹ A floating point number is normalized if the most significant digit of the mantissa is non-zero.

✹ In most systems the exponent is represented in excess base. This means positive exponent. Therefore, a fixed value (e.g., biased value) will be added to the exponent. For example, IBM uses excess 64 for the power, and DEC10 uses excess 128 to represent the power.

Computer Organization

◆**Arithmetic Logic Unit –** Floating Point Numbers

✹ Convert $(-25.75)_{10}$ to a floating point number for IBM:

$(-25.75)_{10} = -(11001.11)_2 = -(.1100111 * 2^5)_2 =$

$(.0001100111 * 2^8)_2 = -.000110011100 * 16^2$

Computer Organization

◆ **Arithmetic Logic Unit –** Floating Point Addition and Subtraction

　✹ Check for zeros.

　✹ Align the mantissas (smaller exponent should be equated to the larger one).

　✹ Add or subtract the mantissas.

　✹ Normalize the result.

　✹ If $exp_1 \geq exp_2$ then

$$fl_1 \pm fl_2 = \left((mantissa_1) \pm (mantissa_2) * r_b^{-(exp_1 - exp_2)} \right) * r_b^{exp_1}$$

　✹ If $exp_1 < exp_2$ then

$$fl_1 \pm fl_2 = \left((mantissa_1) * r_b^{-(exp_2 - exp_1)} \pm (mantissa_2) \right) * r_b^{exp_2}$$

Computer Organization

◆**Arithmetic Logic Unit –** Floating Point Multiplication

✴ Check for zeros.

✴ Add the exponents.

✴ Adjust the exponent of the result.

✴ Multiply the mantissas.

✴ Normalize the result.

$$fl_1 * fl_2 = (mantissa_1) * (mantissa)_2 * r_b^{(exp_1 + exp_2)}$$

Computer Organization

◆ **Arithmetic Logic Unit –** Floating Point Division

✸ Check for zeros.

✸ Align the dividend (for divide overflow condition).

✸ Subtract the exponents.

✸ Adjust the exponent of the result.

$$fl_1 / fl_2 = (mantissa_1) / (mantissa)_2 * r_b^{(exp_1 - exp_2)}$$

Computer Organization

◆ **Arithmetic Logic Unit**

✳ Singed magnitude number: A method to represent signed numbers.

✳ 2^s complement number: A method to represent signed numbers.

✳ 1^s complement number: A method to represent signed numbers.

✳ Addition overflow: A case where the addition result is too big to fit in the destination register.

Computer Organization

◆ Arithmetic Logic Unit

* ✳ Divide overflow: A case when the quotient result is too big to fit in the quotient register.

* ✳ Add-and-shift algorithm: A method to perform multiplication.

* ✳ Booth's algorithm: A method to perform multiplication for 2^s complement numbers.

* ✳ Restoring technique: A method to perform division.

* ✳ Non-restoring technique: A faster method than restoring technique to perform division.

Computer Organization

◆Arithmetic Logic Unit

✹ Biased value: A fixed and positive value added to the exponent of a floating point number. The goal is to deal with positive (unsigned) exponents.

✹ Normalized number: A floating point number is normalized if the the first digit in the mantissa is a non zero digit.

✹ Aligning the mantissa: A process by which the mantissa is shifted to the right by the number of differences between two exponents.

✹ Mantissa: The fraction portion of a floating point number.

Computer Organization

◆ Questions:

✹ Prove that two unsigned numbers can be compared by means of 1^s complement addition.

✹ How do we detect overflow for signed magnitude addition?

✹ How do we detect overflow for 2s complement addition (note I am looking for another technique than the one discussed in this unit)?

✹ Prove that the add-and-shift algorithm, as implemented in this unit, does not generate an overflow.

✹ Justify the following statement. On average, Booth's algorithm is faster than add-and-shift algorithm.

Computer Organization

◆ Questions:

✴ On average, Booth's algorithm is faster than add-and-shift algorithm, this means that in some cases Booth's algorithm would be slower than add-and-shift algorithm. Can you give an example that supports this?

✴ Add-and-shift checks one bit of multiplier during each iteration, Booth's algorithm checks two bits of multiplier during each iteration and as we discussed Booth's algorithm is faster that add-and-shift. Can you come up with an algorithm that multilies two numbers faster than Booth's algorithm?

Computer Organization

◆Questions:

✸ In case of a floating point addition or subtraction operation, why should we always align the mantissa of the smaller number?

✸ In the case of multiplication of two floating point numbers, after adding the exponents, why should we subtract one excess base from the result?

✸ In the case of division of two floating point numbers, after subtracting the exponents, why should we add one excess base to the result?