10/10/2020

Assignment - 3

Computer Organisation and Architechture

Y. S. V. Sumanth
19131A05R6
CSE -4

1) Represent the following decimal numbers in both binary sign/magnitude and twos complement using 16 bits.

a) +512 b) -29

Ans Binary value of 512 – 0000 0010 0000 0000 } in 16 bits
Binary value of 29 – 0000 0000 0001 1101

Sign Magnitude:

+512 = 0 000 0010 0000 0000 (MSB is kept 0 because of "+" sign)

-29 = 1 000 0000 0001 1101 (MSB is changed to 1 because number has a "-" sign).

Twos complement:

+512 = 0000 0010 0000 0000 (2's compliment is not possible because it is not a negative number)

-29 = 1111 1111 1110 0011 (Since negative number we find the 2's complement of given number)

2) Represent the following twos complement values in decimal

a) 1101011 b) 0101101

Ans a) 1101011

Because this starts with a leftmost 1 (MSB), it is a negative number. The magnitude of the negative number is determined by flipping the bits and adding 1

2's complement 010100 + 1 = 010101 (21). Original value is -21.

b) 0101101

Because this starts with MSB 0, it is a positive number

So it magnitude is +45.

3) Calculate (72530 - 13250) using tens complement arithmetic. Assume rules similar to those for twos complement arithmetic.

Ans

Given 72530 - 13250

Let M = 72530 , N = 13250

We have M - N = [M + (-N)]

```
  72530
+ 86750   → tens complement of 13250
 159280
```

Discard carry digit : - 100000

Result is 59280

∴ 72530 - 13250 = 59280

```
  99999
  13250
  86749
     +1
  86750
```

4) Assume numbers are represented in 8 bit twos complement number representation. Show the calculation of the following.

a) 6 + 13 b) -6 + 13 c) 6 - 13 d) -6 - 13.

Ans Procedure: Add the two numbers including their sign bits, and discard any carry (Left most bit position) of the sign bit position

a)
```
+6   00000110
+13  00001101
+19  00010011
```

b)
```
-6   11111010   → 2's complement
+13  00001101
+7   00000111
```

d)
```
-6   11111010
-13  11110011   } 2's complement
-19  11101101
```

c)
```
+6   00000110
-13  11110011   → 2's complement
-7   11111001
```

Note: The operation performed is always addition, including the sign bits. Any carry out of the sign bit position is discarded, and negative results are automatically in 2's complement form.

5) Find the following differences using twos complement arithmetic

Ans a) 111000 − 110011

```
  111000
(+)001101  → twos complement of 110011
 (1)000101
```

carry is discarded

So Result is 000101

```
  110011
  001100
    +1
  001101
```

Twos complement

b) 11001100 − 101110

```
  11001100
(+)11010010  → 2's complement of 00101110
 (1)10011110
```

carry is discarded

So Result is 10011110.

```
  00101110
  11010001
     +1
  11010010
```

Twos complement

c) 111100001111 − 110011110011

```
  111100001111
(+)001100001101  → 2's complement of 110011110011
 (1)001000011100
```

carry is discarded.

So Result is 0010 0001 1100.

```
  110011110011
  001100001100
     +1
  001100001101
```

Twos complement

d) 11000011 − 11101000

```
  11000011
(+)00011000
  11011011
```

Since no carry is generated, Result should be converted to twos complement form.

So Result is − 00100101.

(since MSB is 1)

```
  11101000
  00010111
     +1
  00011000
```

Twos complement

6) Given $x = 0101$ and $y = 1010$ in twos complement notation (i.e ($x=5$ and $y=-6$) compute the product with Booth algorithm.

Ans Given $x = 5 = 0101$

$y = -6 = 1010$ using Booth Algorithm.

Initial Values	M	A	Q	Q-1	Cycle
Initial	0101	0000	1010	0	
Shift	0101	0000	0101	0	First cycle
A ← A-M	0101	1011	0101	0	Second cycle
Shift	0101	1101	1010	1	
A ← A+M	0101	0010	1010	1	Third cycle
Shift	0101	0001	0101	0	
A ← A-M	0101	1100	0101	0	Fourth cycle
Shift	0101	1110	0010	1	

AQ → gives the product value of the multiplicand and multiplier

Multiplier in Q = 1010 (i.e -6)

Multiplicand in M = 0101 (i.e 5)

AQ = 1110 0010

↓

Since MSB is 1, it is a negative number in its 2's complement form.

So the number AQ = 0011101

flipping its bits. +1

0011110 (Actual value is 30)

∴ Product is −30.

7) Use the Booth Algorithm to multiply 23 (multiplicand) by 29 (multiplier) where each number is represented using 6 bits.

Ans Given multiplicand 23 - 010111
multiplier 29 - 011101

Initial Values	M	A	Q	Q-1	Cycle
Initial	010111	000000	011101	0	
A ← A-M	010111	101001	011101	0	First
Shift	010111	110100	101110	1	
A ← A+M	010111	001011	101110	1	Second
Shift	010111	000101	110111	0	
A ← A-M	010111	101110	110111	0	Third
Shift	010111	110111	011011	1	
Shift	010111	111011	101101	1	Fourth
Shift	010111	111101	110110	1	Fifth
A ← A+M	010111	010100	110110	1	Sixth
Shift	010111	001010	011011	0	

AQ → gives the product of the multiplicand and multiplier

AQ → 001010 011011
↓
Since MSB is 0, it is a positive number.

∴ Product value is 667.

∴ Product of 010111 and 011101 is 001010 011011

8) Show how the following floating point additions are performed (where significands are truncated to 4 decimal digits). Show the results in normalised form.

Ans a) $5.566 * 10^2 + 7.777 * 10^2$

$= (5.566 + 7.777) * 10^2$

$= 13.343 * 10^2$

$= 1.3343 * 10^3$

$= 1.3343 * 10^3$ (truncated to 4 decimal digits)

b) $3.344 * 10^1 + 8.877 * 10^{-2}$

$= 3.344 * 10^1 + 0.008877 * 10^1$

$= (3.344 + 0.008877) * 10^1$

$= 3.352877 * 10^1$

$= 3.3528 * 10^1$ (truncated to 4 decimal digits)

9) Show how the following floating point subtractions are performed (where significands are truncated to 4 decimal digits). Show the results in normalised form.

Ans a) $7.744 * 10^{-3} - 6.666 * 10^{-3}$

$= (7.744 - 6.666) * 10^{-3}$

$= 1.078 * 10^{-3}$.

$= 1.0780 * 10^{-3}$ (truncated to 4 decimal digits)

b) $8.844 * 10^{-3} - 2.233 * 10^{-1}$

$= 0.08844 * 10^{-1} - 2.233 * 10^{-1}$

$= (0.08844 - 2.233) * 10^{-1}$

$= -2.224156 * 10^{-1}$

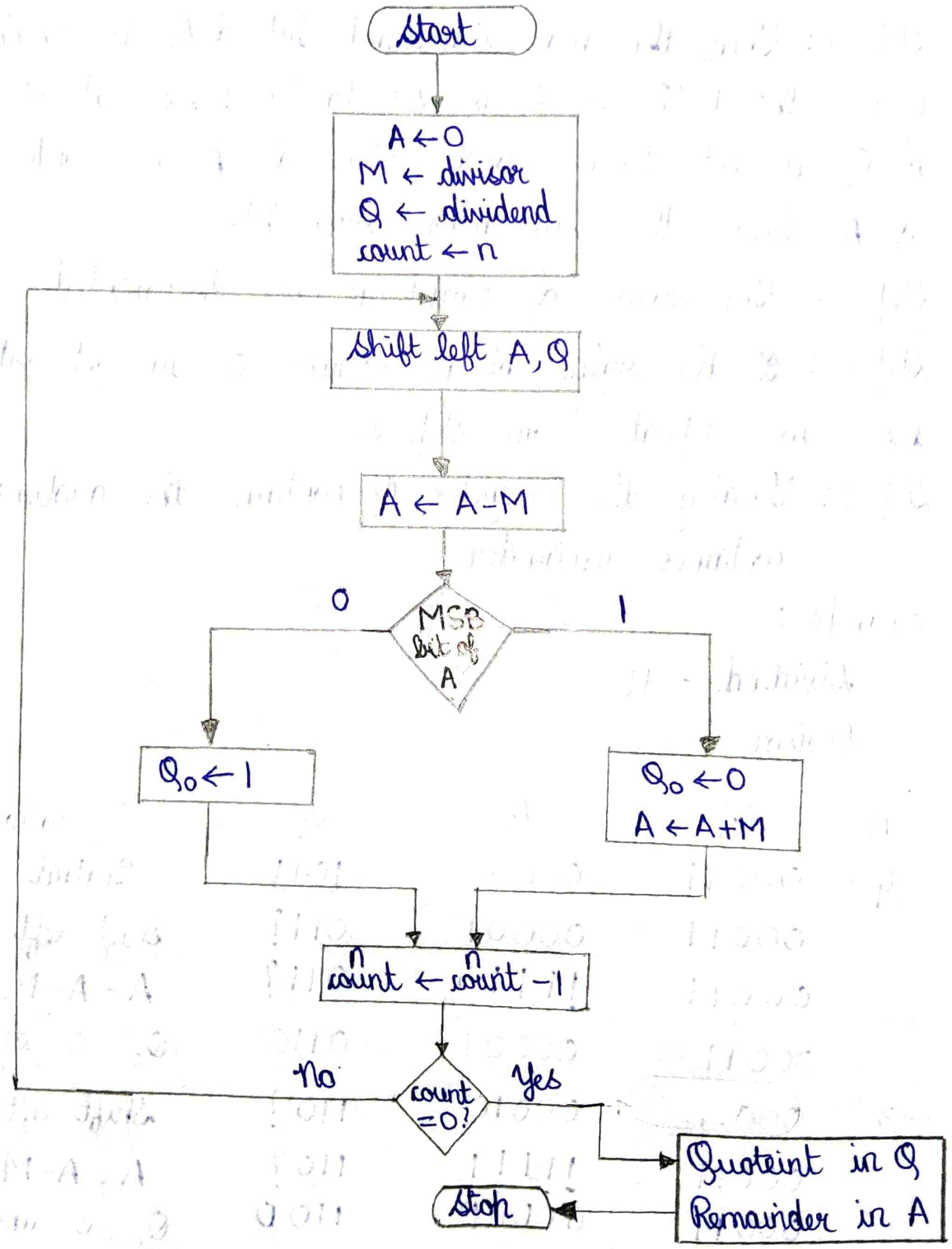
$= -2.2241 * 10^{-1}$ (truncated to 4 decimal digits)

10) Explain Division restoring algorithm with an example.

Ans. A division algorithm provides a quotient and a remainder when we divide two numbers.

In division restoring algorithm, restoring term is due to fact that value of register A is restored after each iteration.

Flow Chart for Division Restoring Algorithm



Steps involved:

Step 1: First the registers are initialised with corresponding values (Q = Dividend, M = Divisor, A = 0, n = no of bits in dividend)

Step 2: Then the content of register A and Q is shifted left as if they are a single unit.

Step 3: The content of register M is subtracted from A and result is stored in A.

Step 4: Then the most significant bit of A is checked and if it is 0 the LSB of Q is set to be 1 else it is 1, the LSB of Q is set to 0 and value of A is registered i.e value of A before the subtraction with M.

Step 5: The value of count n is decremented

Step 6: If the value of n becomes 0 we get out of the loop else we repeat from step 2.

Step 7: Finally, the register Q contain the quotient and A containes remainder.

Example:

Dividend = 11

Divisor = 3

n	M	A	Q	Operation
4	00011	00000	1011	Initial
	00011	00001	011?	Shift left A,Q
	00011	11110	011?	A = A - M
	00011	00001	0110	Q_0 = 0 and restore A
3	00011	00010	110?	Shift left A,Q
	00011	11111	110?	A = A - M
	00011	00010	1100	Q_0 = 0 and restore A

2	00011 00011 00011	00101 00010 00010	100? 100? 1001	Shift left A,Q A ← A-M $Q_0 = 1$
1	00011 00011 00011	00101 00010 00010	001? 001? 0011	Shift left A,Q A ← A-M $Q_0 = 1$

Finally Q register contains the quotient i.e 0011 – 3 and A register contains the remainder i.e 00010 – 2

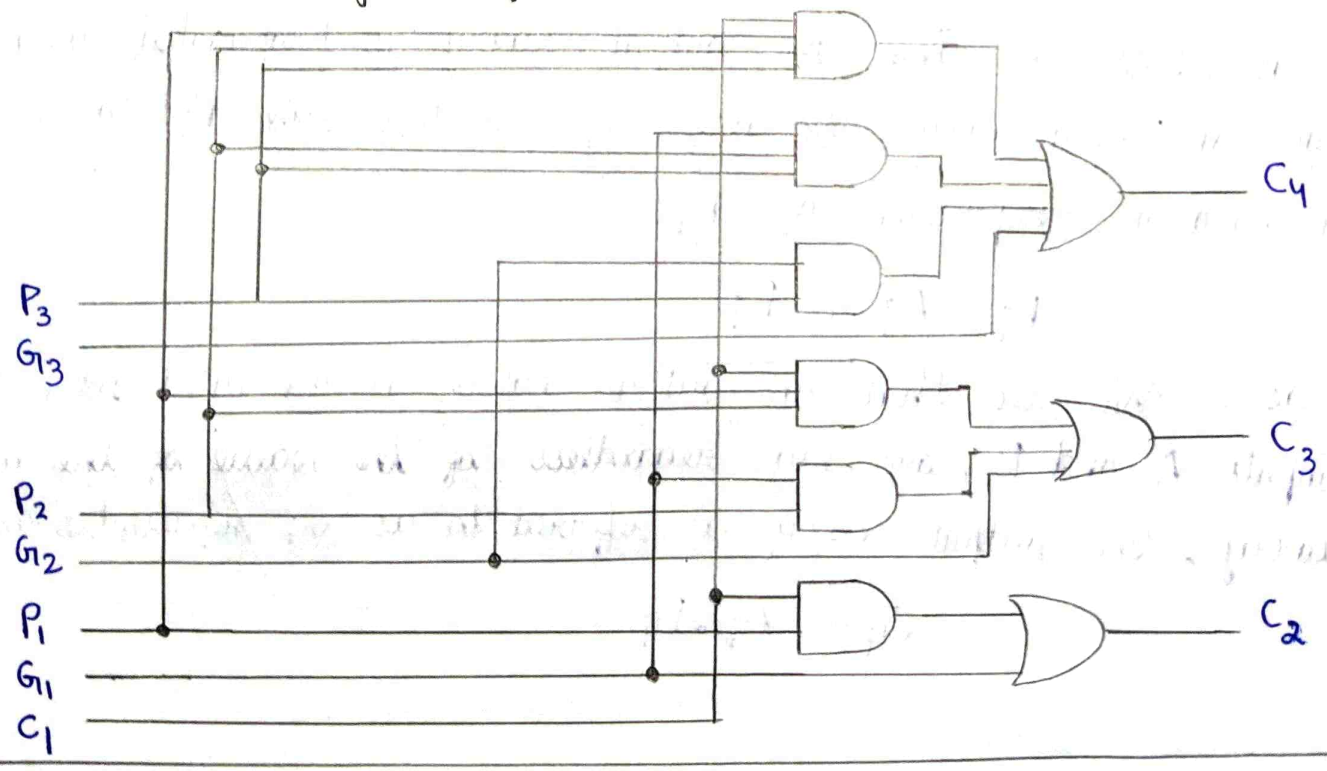
Quotient – 3 , Remainder – 2

11) Draw and explain the 4 bit carry look ahead adder circuit.

Ans **Carry Look Ahead Adder:**

A carry look ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed group of bits of the adder is reduced to two level logic.

Circuit diagram of 4-bit carry look ahead adder



Let us consider a full adder. We have the inputs A, B and C_{in} If we consider the addition of these three variables in every possible case.

A	B	C_{in}	Sum	Carry	Condition
0	0	0	0	0	
0	0	1	1	0	No Carry generated
0	1	0	1	0	
0	1	1	0	1	Carry Propagated
1	0	0	1	0	No carry
1	0	1	0	1	Carry Propagated
1	1	0	0	1	
1	1	1	1	1	Carry generated

On analyzing the above truth table, we see that the carry is 1 when.

1) Either the value of A or B is one, as well as C_{in} is 1, or
2) Both A and B have the value 1.

Now there are 2 new variables Carry Generate (G_i) and Carry Propagate (P_i)

Case 1: We see that the output carry is propagated, when we give an input carry. We will refer to this with P_i. So the mathematical expression of P_i

$$P_i = A_i \oplus B_i$$

Case 2: We see that an output carry is generated when both inputs A and B, are high regardless of the value of the input carry. The output carry is referred to as G_i represented as

$$G_i = A_i \cdot B_i$$

Now, for a full adder

$$Sum = A_i \oplus B_i \oplus C_{in}$$

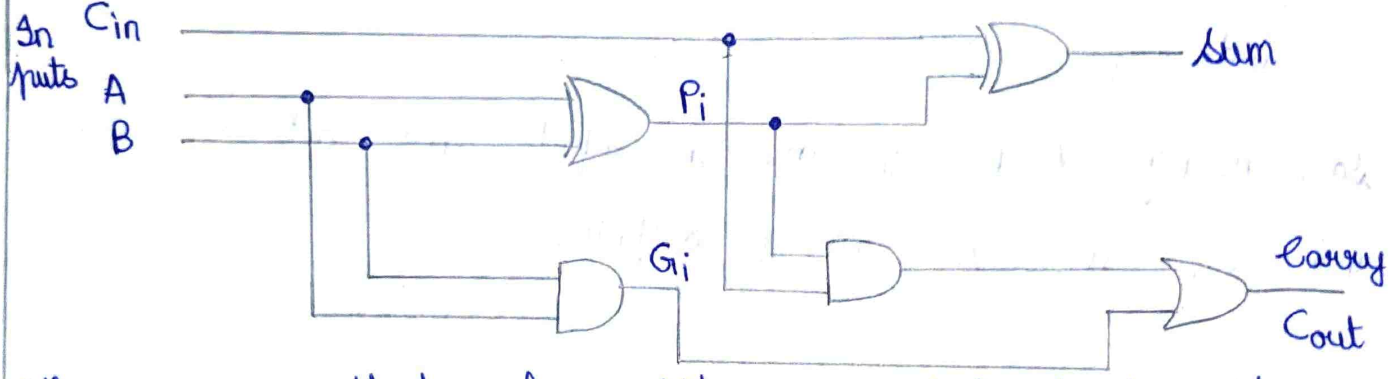
$$Carry = C_{in}(A_i \oplus B_i) + A_i B_i = A_i B_i + B_i C_{in} + A_i C_{in}$$

Now the above equations in terms of carry propagate (P_i) and carry generate (G_i)

$$Sum = P_i \oplus C_{in}$$

$$Carry = G_i + P_i . C_{in}$$

Logic circuit of sum and carry is given below.



The carry output of 4 bit carry look ahead adder at each stage.

$$C_1 = G_0 + P_0 C_{in}$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

From the above Boolean equations we can observe that C_4 does not have to wait for C_3 and C_2 to propagate but actually C_4 is propagated at the same time as C_3 and C_2. They all depend only on the initial carry C_{in}. So the disadvantage of the ripple carry adder is overcomed.

In the carry look ahead adder's logical circuit, there are 3 blocks "P and G1 Generator", "Carry look ahead" and "adder block."

Input Augend and Addend is provided to the P and G1 generator block whose output is connected with CLA and the adder block.

So the process continues with initial carry C_0 only being depended to get all the remaining carries C_1, C_2 C_3 and C_4.

So, giving A, B, C_{in} are as inputs we get C_1, C_2, C_3 and C_4 as the carry outputs.