COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE
BACHELOR OF COMPUTER APPLICATIONS (B.C.A)
III Semester

Mr. Vijay Prakash Mishra

Assistant Professor

Department of Computer Application

Jagatpur P. G. College, Varanasi

(Affiliated to Mahatma Gandhi Kashi Vidyapeeth, Varanasi)

Email- vijayprakashmishra1971@gmail.com

UNIT - III

Computer Arithmetic

General concept

- Decimal addition

```
(carry)  1_
         19
       + 7
         26
```

- Binary addition

```
( carry)  111_
          10011
        + 111
          11010
```

- `16+8+2 = 26`

Signed and unsigned additions

- Unsigned addition in 4-bit arithmetic

```
( carry)   11_
           1011
         + 0011
           1110
```

- `11 + 3 = 14`
  `(8 + 4 + 2)`

- Signed addition in 4-bit arithmetic

```
( carry)   11_
           1011
         + 0011
           1110
```

- `-5 + 3 = -2`

Signed and unsigned additions

- ***Same rules*** apply even though bit strings represent ***different values***
- Sole difference is ***overflow handling***

Overflow handling (I)

- No overflow in signed arithmetic

```
( carry)  111_
          1110
        + 0011
        ------
          0001
```

- `-2 + 3 = 1 (correct)`

- Signed addition in 4-bit arithmetic

```
( carry)  1___
          0110
        + 0011
        ------
          1001
```

- `6 + 3 ≠ -7 (false)`

Overflow handling (II)

- In signed arithmetic an overflow happens when
  - The ***sum of two positive numbers*** exceeds the maximum positive value that can be represented using n bits: $2^{n-1} - 1$
  - The s***um of two negative numbers*** falls below the minimum negative value that can be represented using n bits: -2^{n-1}

Example

- Four-bit arithmetic:
  - Sixteen possible values
  - Positive overflow happens when result > 7
  - Negative overflow happens when result < -8
- Eight-bit arithmetic:
  - 256 possible values
  - Positive overflow happens when result > 127
  - Negative overflow happens when result < -128

An interesting consequence

- Most C compilers ignore overflows
  - C compilers must use unsigned arithmetic for their integer operations
- Fortran compilers expect overflow conditions to be detected
  - Fortran compilers must use signed arithmetic for their integer operations

Negating a number

- Toggle all bits then add one

In 4-bit arithmetic (I)

0000	0	1111	+1 = 0000	0
0001	1	1110	+1 = 1111	-1
0010	2	1101	+1 = 1110	-2
0011	3	1100	+1 = 1101	-3
0100	4	1011	+1 = 1100	-4
0101	5	1010	+1 = 1011	-5
0110	6	1001	+1 = 1010	-6
0111	7	1000	+1 = 1001	-7

Decimal multiplication

```
(carry) 1_
          37
       x  12
       ----
          74
         370
       ----
         444
```

- What are the rules?
  - Successively multiply the multiplicand by each digit of the multiplier starting at the right shifting the result left by an extra left position each time each time but the first
  - Sum all partial results

Binary multiplication

```
(carry)
 111
   1101
  x 101
  ------
   1101
     00
 110100
-------
1000001
```

- What are the rules?
  - Successively multiply the multiplicand by each digit of the multiplier starting at the right shifting the result left by an extra left position each time each time but the first
  - Sum all partial results
- **Binary multiplication is easy!**

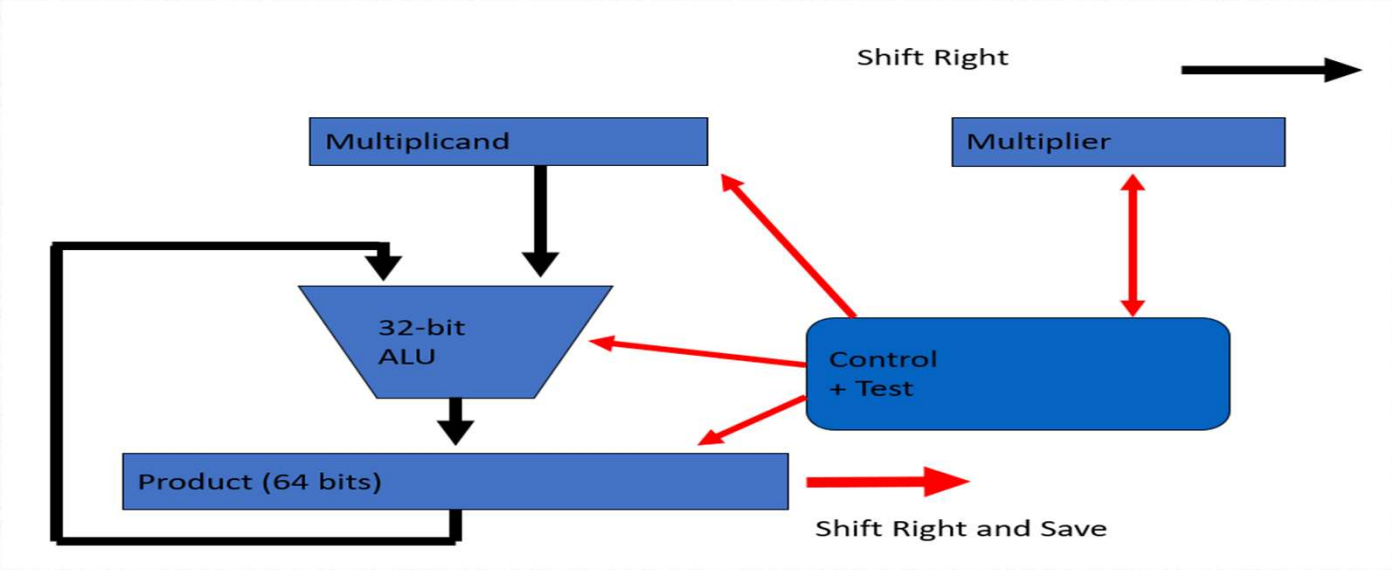
Binary multiplication table

X	0	1
0	**0**	**0**
1	**0**	**1**

Algorithm

- Clear contents of 64-bit product register
- For (i = 0; i <32; i++) {
  - If (LSB of multiplier_register ==1)
    - Add contents of multiplicand register to product register
  - Save LSB of product register
  - Shift ***right*** one position ***both*** multiplier register and product register
- } / / for loop

Multiplier: Second version



Example (I)

- Multiply **0011** by **0011**
- **<u>Start</u>**

Multiplicand	Multiplier	Product	Result
0011	**0011**	**--**	**--**

- **<u>First bit</u>**

Multiplicand	Multiplier	Product	Result
0011	**001<u>1</u>**	**0011**	**--**

Decimal division (long division)

```
   303
7 | 2126
  -210
     26
    -21
      5
```

- What are the rules?
  - Repeatedly try to subtract smaller multiple of divisor from dividend
  - Record multiple (or zero)
  - At each step, repeat with a lower power of ten
  - Stop when remainder is smaller than divisor

Binary division

```
        011
11 | 1011
     x
    -11
     1011
    >-11
x     101
    >>-11
      10
```

- What are the rules?
  - Repeatedly try to subtract powers of two of divisor from dividend
  - Mark 1 for success, 0 for failure
  - At each step, shift divisor one position to the right
  - Stop when remainder is smaller than divisor

Division Algorithm

- **For i in range(0,33) :** # from 0 to 32
  - *Subtract* contents of ***divisor register*** from ***remainder register***
  - **If remainder ≥ 0 :**
    - Shift ***quotient register to the left***
    - Set new rightmost bit to **1**

    Else :
    - ***Undo*** subtraction
    - Shift ***quotient register to the left***
    - Set new rightmost bit to **0**
  - Shift ***right*** one position contents of divisor register

FLOATING POINT OPERATIONS

- Used to represent ***real numbers***
- Very similar to ***scientific notation***

$$3.5\times10^{6},\ 0.82\times10^{-5},\ 75\times10^{6},\ \ldots$$

- Both decimal numbers in scientific notation and floating point numbers can be normalized:

$$3.5\times10^{6},\ 8.2\times10^{-6},\ 7.5\times10^{7},\ \ldots$$

Example

- Sign bit is zero:
  Number is positive
- Biased exponent is 127
  Power of two is zero
- Normalized binary value is
  1.0000000
- Number is **$1 \times 2^0 = 1$**

0	01…1	00000000000000000000000000000

Decimal floating point addition

- **$9.25 \times 10^3 + 8.22 \times 10^2 = ?$**
- Denormalize number with smaller exponent:
  $9.25 \times 10^3 + 0.822 \times 10^3$
- Add the numbers:
  $9.25 \times 10^3 + 0.822 \times 10^3 = 10.072 \times 10^3$
- Normalize the result:
  $10.072 \times 10^3 = 1.0072 \times 10^4$

Binary floating point addition

- Say **101 + 11** or $\mathbf{1.01 \times 2^2 + 1.1 \times 2^1}$
- Denormalize number with smaller exponent:
  $\mathbf{1.01 \times 2^2 + 0.11 \times 2^2}$
- Add the numbers:
  $\mathbf{1.01 \times 2^2 + 0.11 \times 2^2 = 10.00 \times 2^2}$
- Normalize the results
  $\mathbf{10.00 \times 2^2 = 1.000 \times 2^3}$

Excercises

- *How would you represent* 0.5 *in double precision?*
- *How would you convert this double-precision value into a single precision format?*
- *When doing accounting, we could do all the computations in cents using integer arithmetic. What would we win? What would we lose?*

Reference

- Reference Books:
- 1. Leventhal, L.A, “Introduction to Microprocessors”, Prentice Hall of India
- 2. Mathur, A.P., “Introduction to Microprocessors”, Tata McGraw Hill
- 3. Rao, P.V.S., “Prospective in Computer Architecture” , Prentice Hall of India

Declaration

"The content is exclusively meant for academic purpose and for enhancing teaching and learning. Any other use for economic/commercial purpose is strictly prohibited. The users of the content shall not distribute, disseminate or share it with anyone else and its use is restricted to advancement of individual knowledge. The information provided in this e-content is authentic and best as per knowledge".

Vijay Prakash Mishra
Assistant Professor
Department of Computer Application
Jagatpur P. G. College, Varanasi

THANK YOU