COMPUTER ORGANIZATION AND ARCHITECTURE

Course Code : CSE 2151

Credits : 04

EXERCISE

- Give a short sequence of machine instructions for the task
  - Add the contents of memory location A to those of location B, and place the answer in location C
  - Following instructions are the only instructions available to transfer data between the memory and the general-purpose registers.
    - Load Ri, LOC
    - Store Ri, LOC
  - Do not change the contents of either location A or B.
- **Solution:**
  - Load R3, A
  - Load R4, B
  - Add R5, R3, R4
  - Store R5, C

NUMBER REPRESENTATION AND ARITHMETIC OPERATIONS

- Number representation
  - Decimal number system:
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    - Each digit has a position value in terms of powers of 10
    - $123 = 1 * 10^2 + 2 * 10^1 + 3 * 10^0$
  - Binary number system
    - 0, 1
    - Each digit has a position value in terms of powers of 2
    - $101 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0$
    - n-bit vector $B = b_{n-1} \ldots b_1 b_0$,
      - where $b_i = 0$ or 1 for $0 \leq i \leq n-1$
    - unsigned integer value $V(B)$ in the range 0 to 2^{n-1}, where
      - $V(B) = b_{n-1} \times 2^{n-1} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$

INTEGER

- Unsigned integer
  - If the integers are represented using 4 bit
    - $0_{(10)}$ in binary?

      0000
    - $15_{(10)}$ in binary?

      1111 $= 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0$
- Signed integer

$b_3b_2b_1b_0$	Value in decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

INTEGER

- Unsigned integer
  - If the integers are represented using 4 bit
    - $0_{(10)}$ in binary?

      0000
    - $15_{(10)}$ in binary?

      1111 $= 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0$
- Signed integer
  - Sign and magnitude
  - One's complement
  - Two's complement
  - In all three systems,
    - the **positive numbers** have the **same** bit representation
    - the **negative numbers** have **different** bit representation
    - positive numbers - the leftmost bit is 0
    - negative numbers- the leftmost bit is 1

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+ 7	+ 7	+ 7
0 1 1 0	+ 6	+ 6	+ 6
0 1 0 1	+ 5	+ 5	+ 5
0 1 0 0	+ 4	+ 4	+ 4
0 0 1 1	+ 3	+ 3	+ 3
0 0 1 0	+ 2	+ 2	+ 2
0 0 0 1	+ 1	+ 1	+ 1
0 0 0 0	+ 0	+ 0	+ 0
1 0 0 0	- 0	- 7	- 8
1 0 0 1	- 1	- 6	- 7
1 0 1 0	- 2	- 5	- 6
1 0 1 1	- 3	- 4	- 5
1 1 0 0	- 4	- 3	- 4
1 1 0 1	- 5	- 2	- 3
1 1 1 0	- 6	- 1	- 2
1 1 1 1	- 7	- 0	- 1

SIGNED INTEGER: SIGN AND MAGNITUDE

- Negative values - most significant bit of the corresponding positive value changed from 0 to 1
- Sign- MSB
- Magnitude (or number)- remaining bits
- 0 represented as positive and negative

Positive Value	$b_3\ b_2b_1b_0$	$b_3\ b_2b_1b_0$	Negative Value
+7	0111	1111	-7
+6	0110	1110	-6
+5	0101	1101	-5
+4	0100	1100	-4
+3	0011	1011	-3
+2	0010	1010	-2
+1	0001	1001	-1
+0	0000	1000	-0

SIGNED INTEGER: ONE'S COMPLEMENT

- Negative values - complementing each bit in the corresponding positive value
- Negative to positive- complementing each bit in the corresponding negative value

  +6 0110

  -6 1001
- For n-bit numbers, this operation is equivalent to subtracting the number from $2^n - 1$.
- Example: +6 to -6 in a 4-bit representation
  - 2^n-1 =15 1111

  +6 <u>0110</u>

  -6 1001
- 0 represented as positive and negative

Positive Value	$b_3\, b_2 b_1 b_0$	$b_3\, b_2 b_1 b_0$	Negative Value
+7	0111	1000	-7
+6	0110	1001	-6
+5	0101	1010	-5
+4	0100	1011	-4
+3	0011	1100	-3
+2	0010	1101	-2
+1	0001	1110	-1
+0	0000	1111	-0

SIGNED INTEGER: TWO'S COMPLEMENT

- 2's-complement of an n-bit number is done by subtracting the number from 2^n.
- How to subtract 2 numbers?
- 0101 – 0100
  - 001
- 01100 – 01000
  - 0100
- 010000 – 0101
  - 01011
- 01000 – 0011
  - ?

First Value	Second Value	Difference
0	0	0
0	1	1 (by borrowing)
1	0	1
1	1	0

SIGNED INTEGER: TWO'S COMPLEMENT

- Negative values - adding 1 to the 1's-complement of corresponding positive value
- Example: +6 to -6 in a 4-bit representation
  - 2^n-1 =15 1111

    +6 <u>0110</u> conversion using

    1001 1's complement

    +1 <u>0001</u>

    -6 **1010**
- For n-bit numbers, this operation is equivalent to subtracting the number from 2^n.
- Example: +6 to -6 in a 4-bit representation
  - 2^n 10000

    +6 <u>00110</u>

    -6 **01010**
- 0 represented as positive

Positive Value	$b_3\ b_2b_1b_0$	$b_3\ b_2b_1b_0$	Negative Value
+7	0111	1000	-8
+6	0110	1001	-7
+5	0101	1010	-6
+4	0100	1011	-5
+3	0011	1100	-4
+2	0010	1101	-3
+1	0001	1110	-2
+0	0000	1111	-1

SIGNED INTEGER: TWO'S COMPLEMENT

- 4 bits: -8 to +7

  -2^{4-1} to $+2^{4-1}-1$

- 5 bits: -16 to +15

  -2^{5-1} to $+2^{5-1}-1$

- 6 bits: -32 to +31

  -2^{6-1} to $+2^{6-1}-1$

- n bits: -2^{n-1} to $+2^{n-1}-1$

SIGNED INTEGERS

- For 4-bit numbers, the value −8 is representable in the 2's-complement system but not in the other systems.
- Sign-and-magnitude system seems the most natural
- 1's-complement system is easily related to this system
- 2's-complement appears unusual.
  - However, it leads to the most efficient way to carry out addition and subtraction operations.
  - It is the one most often used system in modern computers.

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+ 7	+ 7	+ 7
0 1 1 0	+ 6	+ 6	+ 6
0 1 0 1	+ 5	+ 5	+ 5
0 1 0 0	+ 4	+ 4	+ 4
0 0 1 1	+ 3	+ 3	+ 3
0 0 1 0	+ 2	+ 2	+ 2
0 0 0 1	+ 1	+ 1	+ 1
0 0 0 0	+ 0	+ 0	+ 0
1 0 0 0	- 0	- 7	- 8
1 0 0 1	- 1	- 6	- 7
1 0 1 0	- 2	- 5	- 6
1 0 1 1	- 3	- 4	- 5
1 1 0 0	- 4	- 3	- 4
1 1 0 1	- 5	- 2	- 3
1 1 1 0	- 6	- 1	- 2
1 1 1 1	- 7	- 0	- 1

UNSIGNED INTEGERS: OPERATIONS

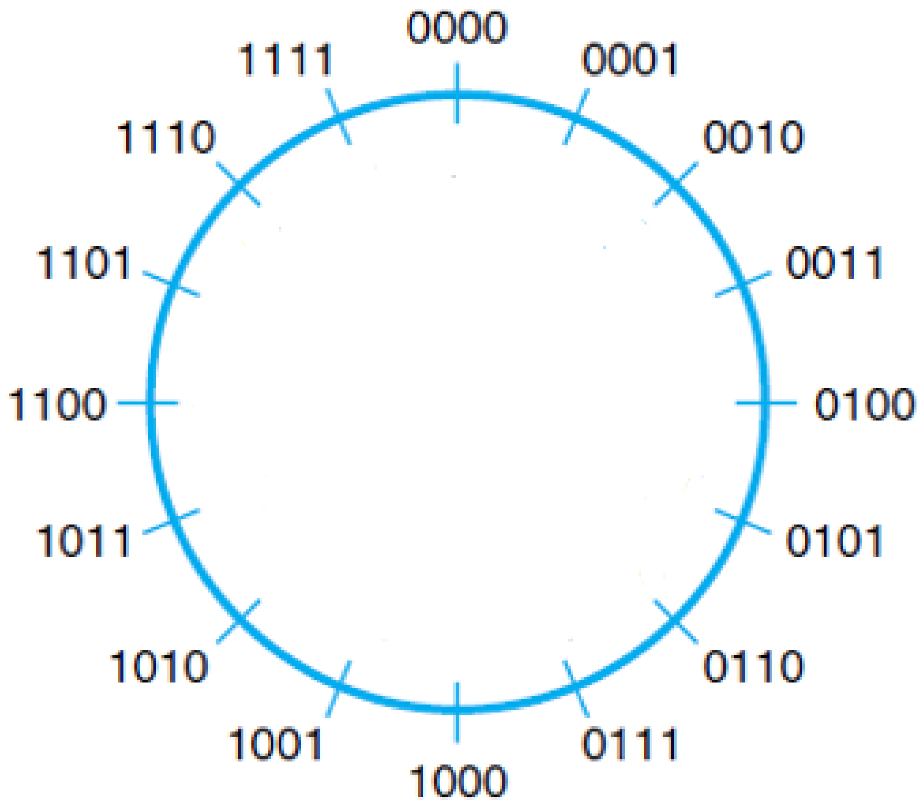
- ```
  #include <iostream>
  using namespace std;
  int main() {
      unsigned short x=65535, y=65537;
      cout<<x<<" "<<y<<endl;
      return 0;
  }
  ```
- x=65535, y=1
- unsigned short: 16 bits
  - 0 to 65535 (0 to 2^n-1)
  - If the value is out of range, it is divided by *largest_number+1* of that datatype, and only the remainder kept.
  - Here, 65537 % 65536= 1
  - Any number bigger than the largest number, "wraps around" the largest number in that type.
  ```

UNSIGNED INTEGERS: OPERATIONS

- ```
  #include <iostream>
  using namespace std;
  int main() {
      unsigned short x=0, y=-1;
      cout<<x<<" "<<y<<endl;
      return 0;
  }
  ```
- x=0, y= 65535
- wraps around to the top of the range.
  ```

UNSIGNED INTEGER: ADDITION

- **7** (0111) + **5** (0101) = ?
  - From 7 move 5 units in clockwise direction
  - 12
- 9 + 14=?
  - From 9 (1001) move 14 units in clockwise direction
  - 7



UNSIGNED INTEGERS: ADDITION

- 2-bit addition:

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 1\,0 \end{array}$$

Carry-out

- Multiple bit addition
  - Similar to decimal addition method
  - add bit pairs starting from the low-order end of the bit vectors, transmitting the carries toward the high-order end
  - The carry-out from the previous bit pair becomes the carry-in to the current bit pair
  - The carry-out from the bit pair in the right must be added to the current bit pair to generate the sum and carry-out at that position
  - If both bits of a pair are 1 and the carry-in is 1, then the sum is 1 and the carry-out is 1

UNSIGNED INTEGERS: SUBTRACTION

- ```
  #include <iostream>
  using namespace std;
  int main(){
      unsigned int x=1, y=2;
      cout<<x-y;
      return 0;
  }
  ```
- 4294967295
  - This occurs due to -1 wrapping around to a number close to the top of the range of a 4-byte integer
  - $2^{32}-1 \rightarrow$ 4294967296-1
  - 32 bits = 4 bytes (size of int)
- Hence, subtraction is not recommended.
  ```

TOPICS COVERED FROM

- Textbook 1:
  - Chapter 1: 1.4.1