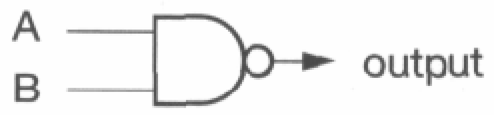
COMPUTER SYSTEMS ORGANIZATION

Abstractions

Spring 2012 -- IIIT-H -- Suresh Purini

Logic Gates – An Example of an Abstraction

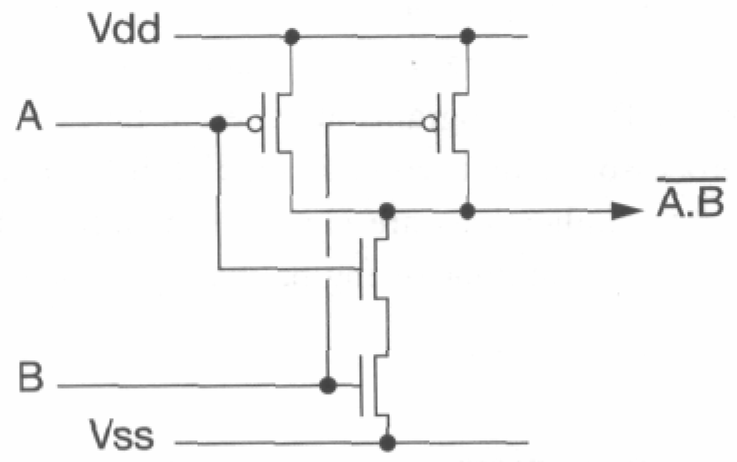


Logic symbol

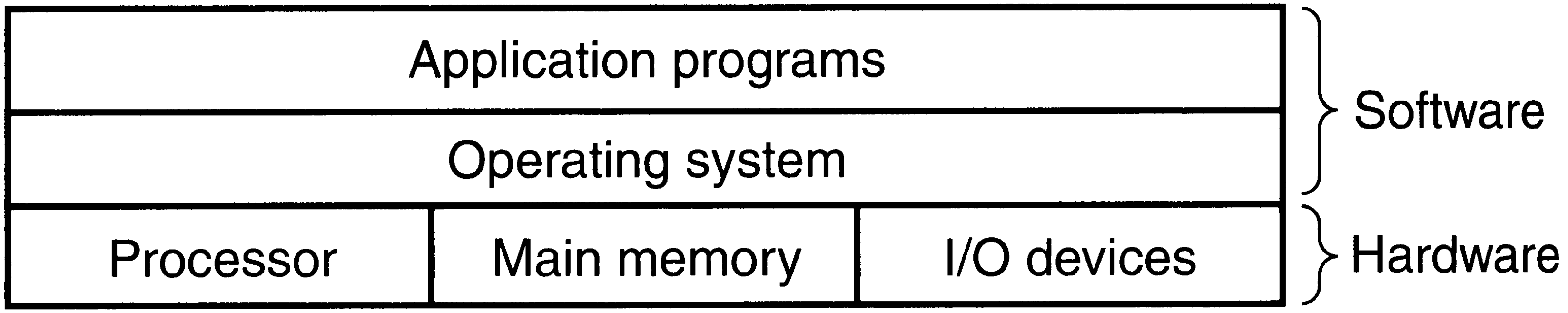
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Truth table

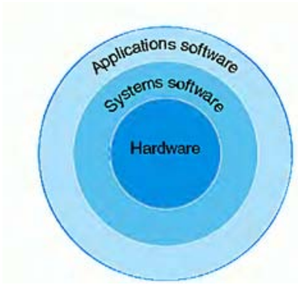
Transistor realization of the NAND gate



Operating System – An Abstraction Layer



- Operating System abstracts away hardware details through a well-defined set of interface functions called System Calls.

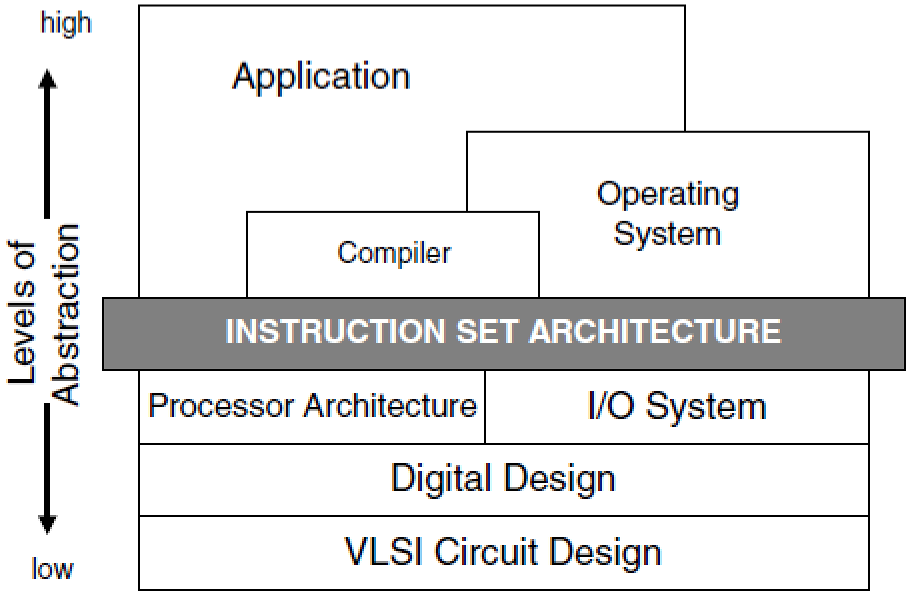


Instruction Set Architecture (ISA)

- ISA is an abstraction for the Software to interface with the Hardware.
- Advantage: Multiple implemenations for the same ISA.
  - Ex: AMD Opteron 64 and Intel Pentium 4 are different Implementations of the same IA32 ISA.

". . . the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls, the logic design, and the physical implementation."

➢**Amdahl, Blaaw, and Brooks, 1964**



What does ISA consists of?

- Instruction Set
- Instruction Format
- Data Types and Data Structures (Integer, Floating Point, …)
- Addressing Modes
- Exceptional Conditions
- …….

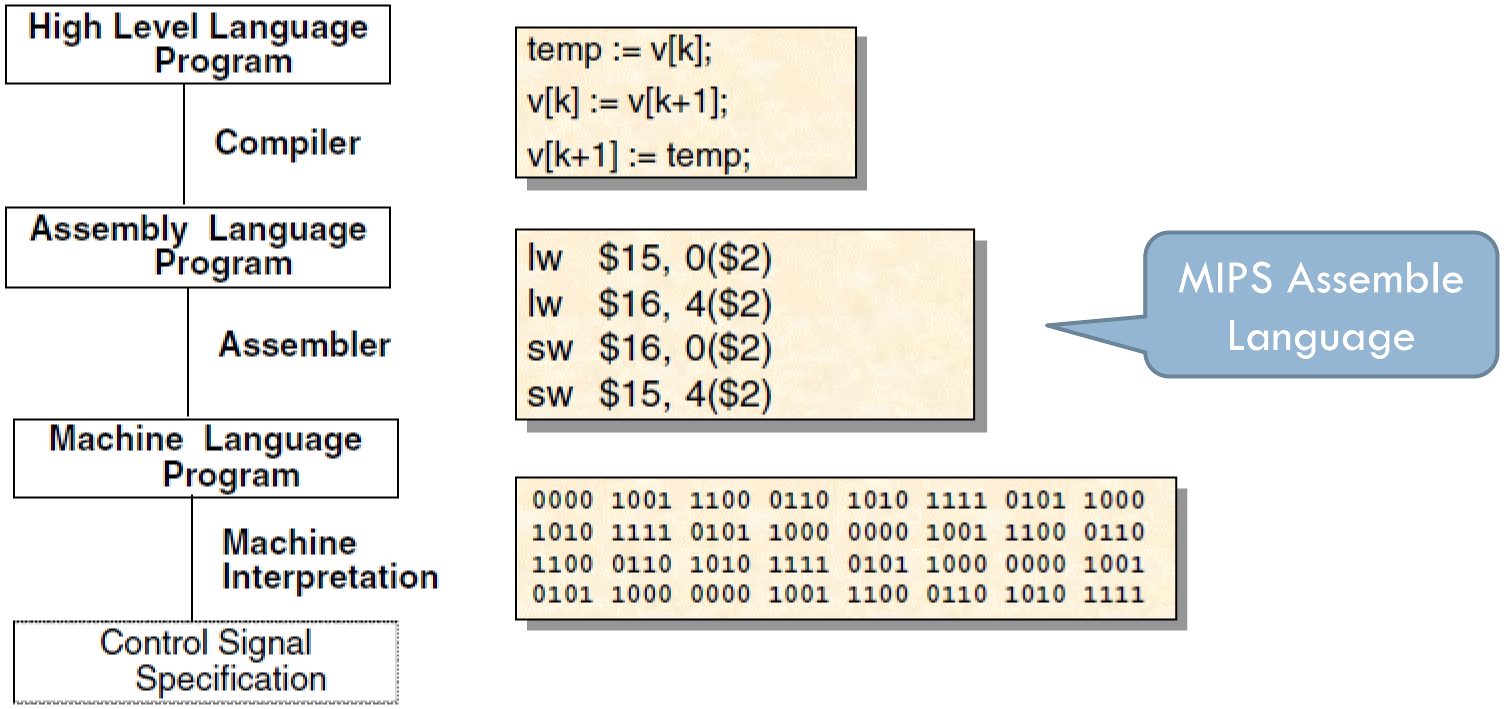
Programming Abstractions

We can program a microprocessor using

a) Instruction opcodes (also called Machine Code)
b) Assembly language
c) High level programming languages

- The level of abstraction increases from Top to Bottom.
- As the level of abstraction increases, ease of programmability also increases!
- Hmm, but we may lose the fine-grained control over the underlying hardware?

Levels of Programming Abstractions



```
temp := v[k];
v[k] := v[k+1];
v[k+1] := temp;
```

```
lw   $15, 0($2)
lw   $16, 4($2)
sw   $16, 0($2)
sw   $15, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Algorithms, Data Structures and Programs

Algorithms + Data Structures = Programs

Niklaus Wirth

Programming Languages and Abstractions

A Programming Language provides

- Data Abstractions
  - int, float, bool etc. data types.
  - Mechanism for hierarchal composition of new Data Abstractions
    - Structures, arrays, Unions etc.
- Data Processing Abstractions
  - Arithmetic and Boolean Operations, String Operations etc.
- Control Abstractions – while, if, for constructs etc.

Key Idea: We should be able to realize these abstractions through the ISA of a given processor!

Machine Representation of Numbers

- Integers
  - Unsigned Integers
  - Signed Integers
- Floating Point Numbers

Question: How to represent these primitive data types in bits and perform arithmetic on them?

What does a bit string mean to us?

- What is the decimal Value for the 4-bit string 1001?
  - Unsigned – 9
  - 1's complement – -6
  - 2's complement – -7
  - Sign-Magnitude – -1
- So how we interpret a bit string depends on the semantics we give to that bit string

Unsigned and Signed Integers

Signed or Unsigned Integers – Doesn't Matter! They still need to be represented as a sequence of bits.

- Consider sequences of w-bits. (Ex: 0101 is a 4-bit sequence)
- There are 2^w possible such sequences.
- It is up to us to interpret each of these sequences in the way we would like to.
- In other words we can associate decimal values to these n-bit strings according to our convenience.

Unsigned Integers

Unsigned Integer Interpretation for n-bit strings:

$$B2U_w : \{ 0, 1 \}^w \rightarrow \{ 0, \ldots , 2^w - 1 \}$$

$$B2U_w(\vec{x}) \doteq \sum_{i=0}^{w-1} x_i 2^i$$

Ex: $B2U_4(0101) = 5$, $B2U_4(1001) = 9$, …

- $U_{w\text{-}max} = B2U_w(111..1) = 2^w - 1$ (Max Integer Value)
- $U_{w\text{-}min} = B2U_w(000\ldots.0) = 0$ (Min Integer Value)

Signed Integers – Sign-Magnitude Representation

Sign-Magnitude Integer Interpretation for w-bit strings:

$$B2S_w : \{ 0, 1 \}^w \rightarrow \{ -2^{w-1}+1, \ldots 0, \ldots , 2^{w-1}-1\}$$

$$B2S_w(\vec{x}) \doteq (-1)^{x_{w-1}} \cdot \left(\sum_{i=0}^{w-2} x_i 2^i \right)$$

Ex: $B2S_4(0101) = 5$, $B2S_4(1001) = -1$,

$B2S_4(0000) = 0$, $B2S_4(1000) = -0$

(w-1)th bit is the sign-bit.

- $S_{w\text{-max}} = B2S_w(011..1) = 2^{w-1} - 1$ (Max Integer Value)
- $S_{w\text{-min}} = B2S_w(100....0) = -2^{w-1} + 1$ (Min Integer Value)

Signed Integers – One's Complement Representation

One's complement Integer Interpretation for w-bit strings:

$$B2O_w : \{ 0, 1 \}^w \rightarrow \{ -2^{w-1}+1, \ldots 0, \ldots, 2^{w-1}-1\}$$

$$B2O_w(\vec{x}) \doteq -x_{w-1}(2^{w-1}-1) + \sum_{i=0}^{w-2} x_i 2^i$$

Ex: $B2O_4(0101) = 5$, $B2S_4(1001) = -6$,

$B2O_4(0000) = 0$, $B2S_4(1111) = -0$

- $O_{w\text{-max}} = B2O_w(011..1) = 2^{w-1} - 1$ (Max Integer Value)
- $O_{w\text{-min}} = B2O_w(100\ldots.0) = -2^{w-1} + 1$ (Min Integer Value)

What is the Shortcut for converting Decimal to One's Complement and vice-versa?

Signed Integers – Two's Complement Representation

Two's complement Integer Interpretation for w-bit strings:

$$B2T_w : \{ 0, 1 \}^w \rightarrow \{ -2^{w-1}, \dots 0, \dots , 2^{w-1}-1\}$$

$$B2T_w(\vec{x}) \doteq -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

Ex: $B2T_4(0101) = 5$, $B2T_4(1001) = -7$,

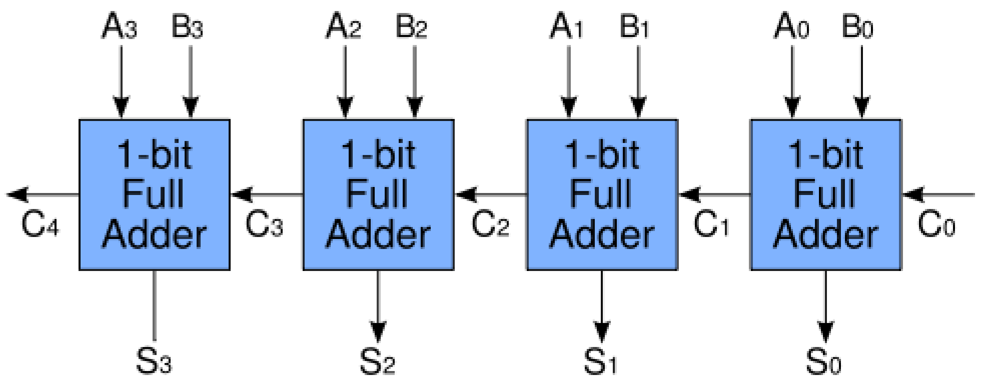
$B2T_4(0000) = 0$, $B2T_4(1111) = -1$

- $T_{w\text{-}max} = B2T_w(011..1) = 2^{w-1} - 1$ (Max Integer Value)
- $T_{w\text{-}min} = B2T_w(100\dots0) = -2^{w-1}$ (Min Integer Value)

What is the Shortcut for converting Decimal to Two's Complement and vice-versa?

Unsigned Addition

- 4-bit Ripple Carry Adder

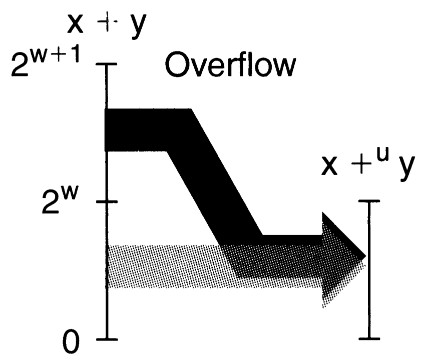


- Overflow Condition: Carry out bit $C_4 = 1$

Overflow in Unsigned Integer Addition

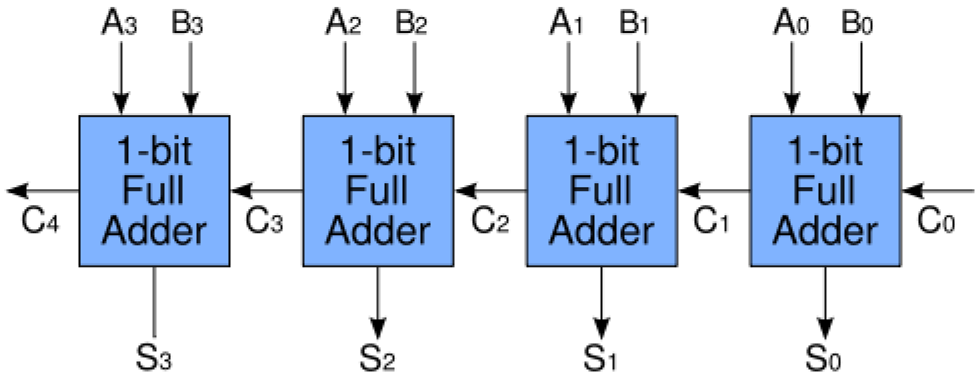
- W-bit Unsigned Integer Addition
  - Overflow: When a carry-bit onto (w+1)th bit position is generated.

$$x +_w^{\mathrm{u}} y \quad = \quad \begin{cases} x + y, & x + y < 2^w \\ x + y - 2^w, & 2^w \le x + y < 2^{w+1} \end{cases}$$



Two's Complement Addition

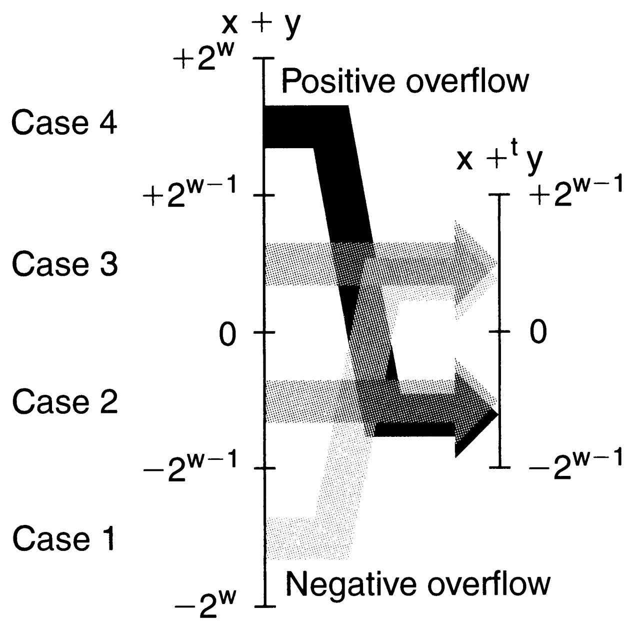
- Can we use a 4-bit ripple carry adder to perform two's complement addition?



- Overflow Condition: C_4 xor $C_3 = 1$

Overflow in Signed (2's Complement) Integer Addition

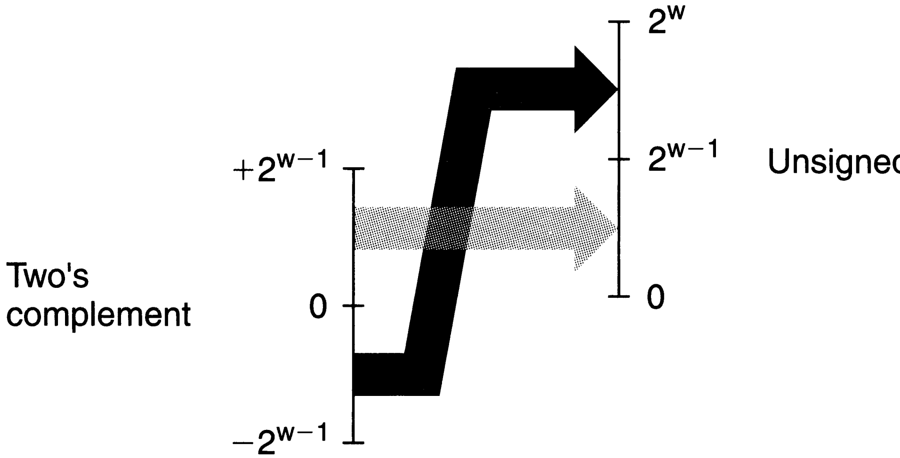
- W-bit signed addition
  - Overflow
    - Two positive integers are added and the result is negative
    - Two negative integers are added and the result is positive



$$x +_w^t y = \begin{cases} x + y - 2^w, & 2^{w-1} \le x + y \\ x + y, & -2^{w-1} \le x + y < 2^{w-1} \\ x + y + 2^w, & x + y < -2^{w-1} \end{cases}$$

Positive Overflow
Normal
Negative Overflow

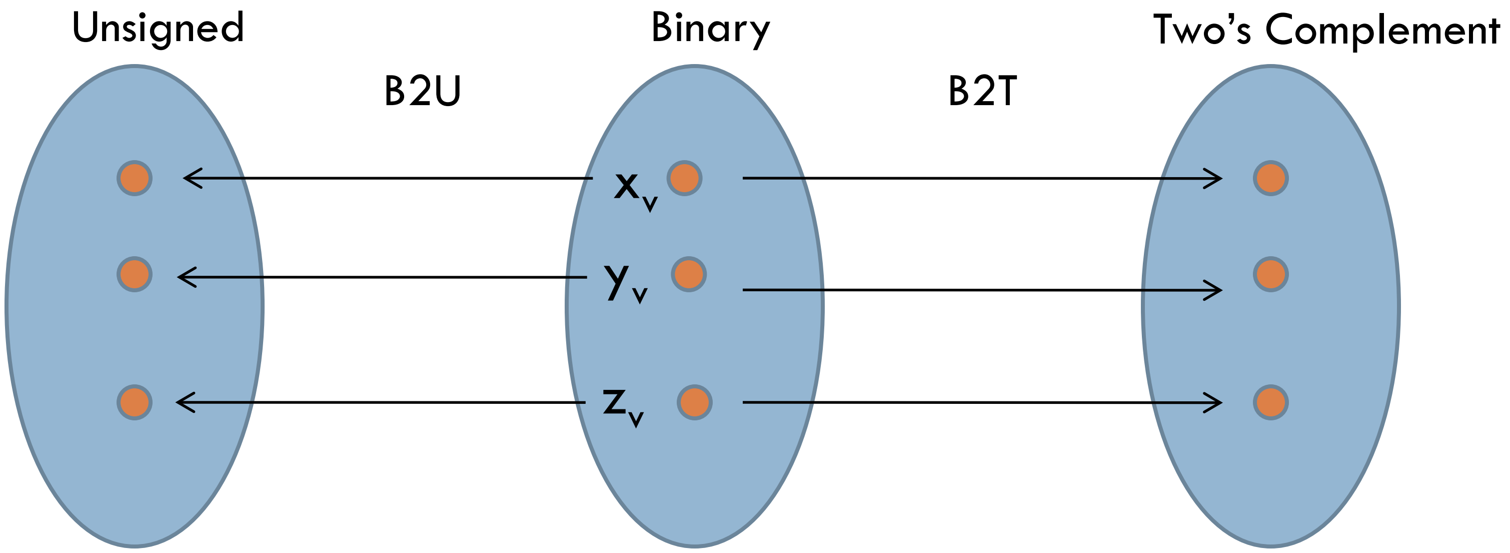
Unsigned and Signed (2's Complement) Integer Addition



Row ID	Binary	Unsigned	2's Complement
R_0	000	0	0
R_1	001	1	1
R_2	010	2	2
R_3	011	3	3
R_4	100	4	-4
R_5	101	5	-3
R_6	110	6	-2
R_7	111	7	-1

1. $R_2 + R_4 = R_6$
   1. 010 + 100 = 110
   2. 2 + 4 = 6
   3. 2 + (-4) = -2
2. $R_6 + R_7 = R_5$
   1. 110 + 111 = 1101
   2. 6 + 7 = 5 (13 mod 8)
   3. -2 + (-1) = -3
3. $R_2 + R_3 = R_5$
   1. 010 + 011 = 101
   2. 2 + 3 = 5
   3. 2 + 3 = -3

Isomorphism Between Binary, Unsigned and Two's Complement Integer Arithmetic (modular)



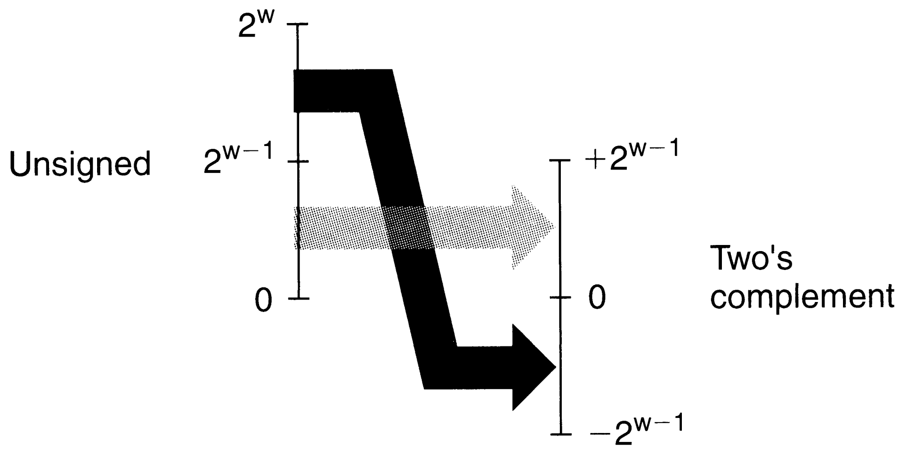
$$x_v + y_v = z_v \pmod{2^w}$$

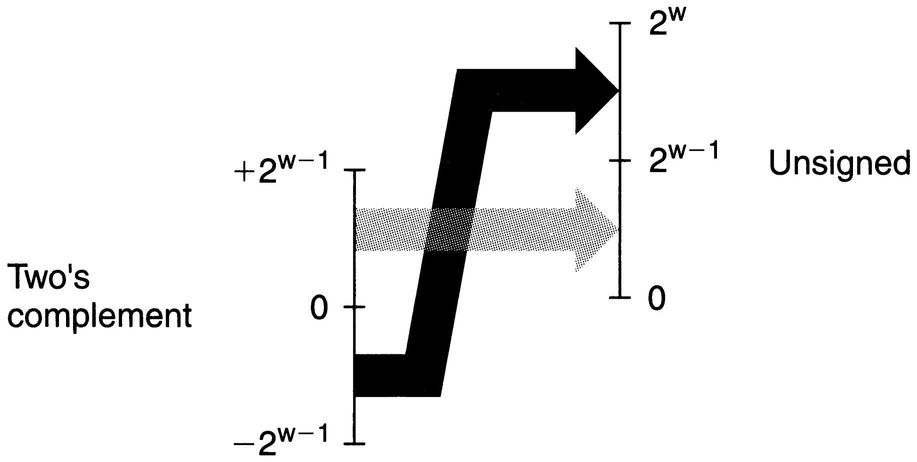
$$B2U_w(x_v) + B2U_w(y_v) = B2U_w(z_v) \pmod{2^w}$$

$$B2T_w(x_v) + B2T_w(y_v) = B2T_w(z_v) \pmod{2^w}$$

In all the above 3 cases, the mod operator is applied when the result doesn't lie within the set range (overflow)

Unsigned and Two’s Complement Representations





Binary and One's Complement Representations

Question: Does isomorphism exists between unsigned and One's Complement Representation of numbers?

Check this 4-bit example:

1100 + 1101 = 1001

-3 + (-2) = -5 (1010)