# Penn State Abington
# CMPEN 271
# Lecture Set #14

2017 – 2018 © R. Avanzato (with solutions)

**Topics:**

- Signed Number Systems — **Video part 1 of 4** ←
- Sample Questions

- Binary Subtraction — **Video part 2 of 4**
- Overflow, Digital Circuits

- Arithmetic Logic Unit (ALU) — **Video part 3 of 4**
- HDLs: VHDL, Verilog

- HW #7 (display negative value) — **Video part 4 of 4**
- Summary

# Engineering in the Real -World

- Read free online version of IEEE Spectrum Magazine
- Real-world engineering applications and research directions
  - Sustainable technologies
  - Robotics
  - Automotive
  - Semiconductors
  - Wireless communication
  - Fiber Optics
  - Microprocessor Design
  - Internet Application
  - Aerospace
  - Biomedical
  - Smartphones, Tablets
  - Consumer Electronics – Gaming etc
  - Careers in Engineering
- Become an IEEE student member
- Web http://www.spectrum.ieee.org/

# Signed Number Systems

- In many digital circuit applications (not all), it is required that the system store and manipulate **negative values** (such as -2, -128, -25 etc)

- One example is a **microprocessor** – microprocessors can do arithmetic functions and need to represent and perform operations on negative numbers.

- There are **many ways to represent negative numbers** in a digital system, and there are advantages and disadvantages to each.

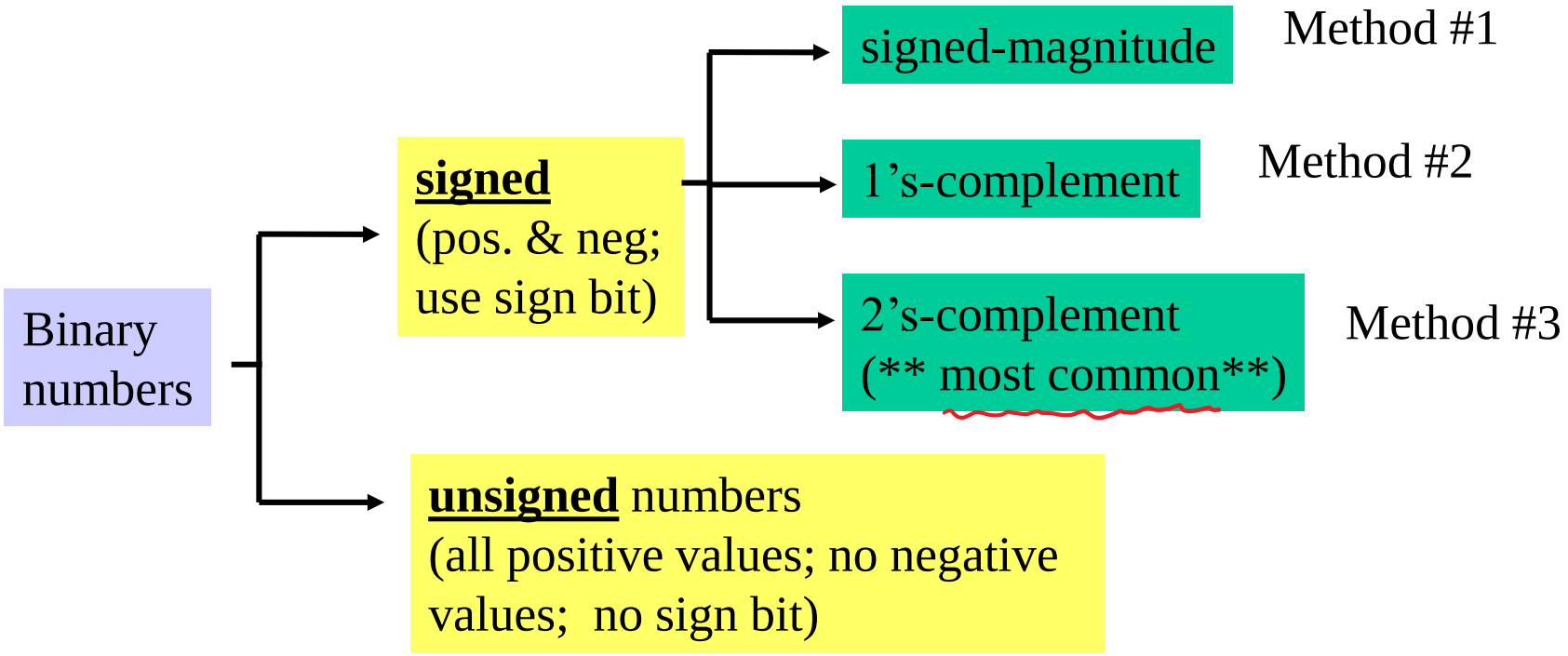
- We will cover some of the **more common and useful techniques** for representing negative numbers in a binary system.

- We will also discuss how to design and **implement digital circuits** to manipulate negative values.

- There are **no "-" negative signs** represented in digital circuits (only 1's and 0's)

# Signed Number Systems

How do you represent negative numbers in binary?




There are 3 common methods for representing negative values in digital circuits (including microprocessors)

# Background: Unsigned numbers

- Up to this point in the course we have been dealing only with **positive numbers** (integers).

- For example**. If we had 4 bits**, we can represent 2^4 or 16 possible binary codes, which represented the positive values 0 (0000) to 15 (1111). There were no negative values represented. This is called an **unsigned number system**

- If we had a **5-bit unsigned number system**, then there are 2^5 or 32 possible binary codes and we would represent the positive values between 0 (00000) and 31 (11111). Be able to generalize this to any n-bit unsigned system.

- Basically, an **unsigned number system** uses the binary codes to represent positive numbers only (no negative values). There are some engineering problems which require unsigned number systems.

- In the next sections, we will look at **3 methods to represent "signed" number** systems which include both positive and negative values. Remember, if you have 5 bits, you can only represent 32 binary codes, either all positive (unsigned system) or a combination of positive and negative values (signed system).

# Signed Number Systems

Consider a 4-bit **unsigned** number system.

What is **range** of values?

0 → 15

**How many values** can be represented?

16

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

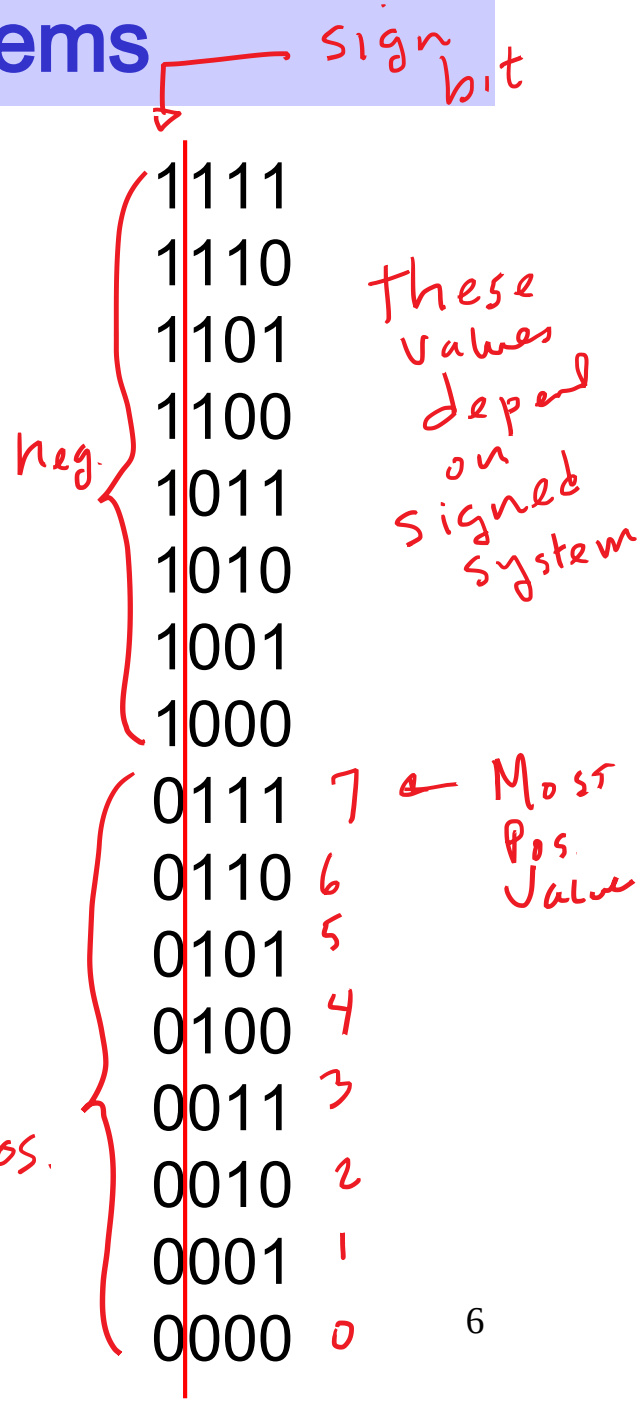
Now, consider a 4-bit **signed** number system. Introduce **sign bit** as msb.

If sign bit = 0, then value is **positive**. If sign bit = 1, then value is **negative**.

What is **range** of values?

**How many positive** values can be represented? 8

**How many negative** values can be represented? 8



# Signed-magnitude

A **signed-magnitude system** can represent both positive and negative values in binary. The most significant (leftmost) bit of the binary number is reserved as the **sign bit**. If the **sign bit = 0, then the value is positive**. If the **sign bit = 1, then the number is negative**. The remaining bits represent the magnitude of the number. **Negative values are produced by setting the sign bit of the positive value to 1.**

Examples:
Assume a 6-bit number system

| binary | decimal |
|---|---|
| 000010 | + 2 |
| 100010 | -2 |
| 001111 | 15 |
| 101111 | -15 |
| 000000 | 0 |
| 100000 | -0 |

sign bit (leftmost; msb)

- **How many numbers** can be represented with 6 bits?
- What is the most positive value that can be represented in the signed-magnitude system with 6 bits?
  *Answer*: 011111 = 31 (base 10)
- What is the **most negative value** that can be represented in the signed-magnitude system with 6 bits?
  *Answer*: 111111 = - 31 (base 10)
- What are the **limitations** of the signed-magnitude system? *Answer*: 1) complicated circuitry for addition and subtraction, 2) there are two codes for zero (000000, 100000)

# Signed-Magnitude Number System

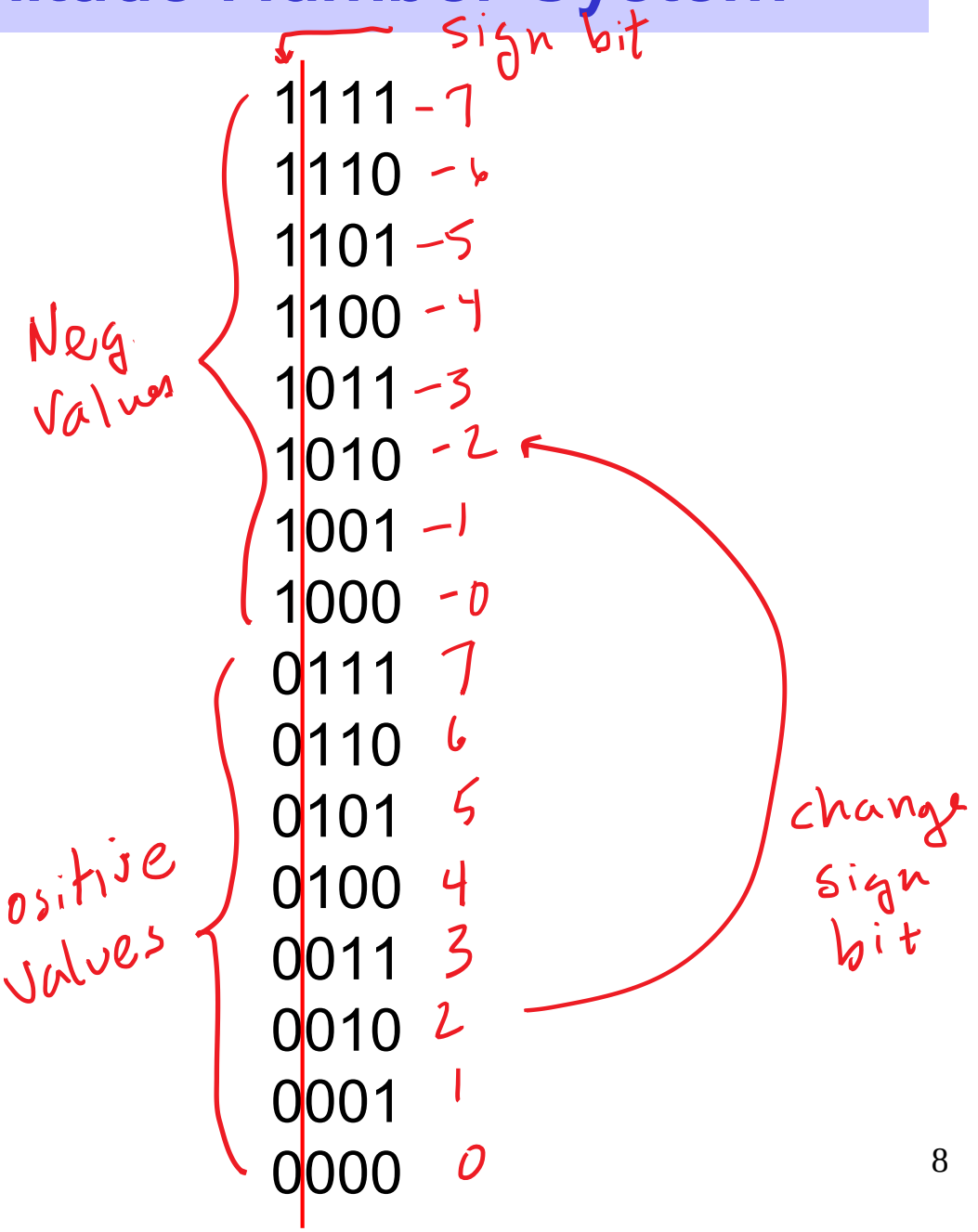
Assume a 4-bit **signed** number system using the **signed –magnitude system**.

What is range of values? -7 to +7

What is the most positive value? +7

What is the most negative value? -7

Generalize to n bits



# 1's-complement System

A **1's-complement (or one's-complement) system** can represent both positive and negative values in binary. The most significant (leftmost) bit of the binary number is reserved as the **sign bit**. If the sign bit = 0, then the value is positive. If the sign bit = 1, then the number is negative. **Negative values are produced by inverting (complementing) each bit of the positive value**.

Examples:
Assume a 6-bit number system

| binary | decimal |
|---|---|
| **0**00010 | 2 |
| **1**11101 | -2 |
| | |
| **0**01111 | 15 |
| **1**10000 | -15 |
| | |
| **0**00000 | 0 |
| **1**11111 | -0 |

sign bit (leftmost)

- **How many numbers** can be represented with 6 bits?
- What is the **most positive value** that can be represented in the one's-complement system with 6 bits?
  Answer: 011111 = 31 (base 10)
- What is the **most negative value** that can be represented in the one's-complement system with 6 bits?
  Answer: 100000 = - 31 (base 10)
- What are the **limitations** of the one's-complement system? Answer: 1) complicated circuitry for addition and subtraction, 2) there are two codes for zero (000000, 111111)

# 1's-complement Number System

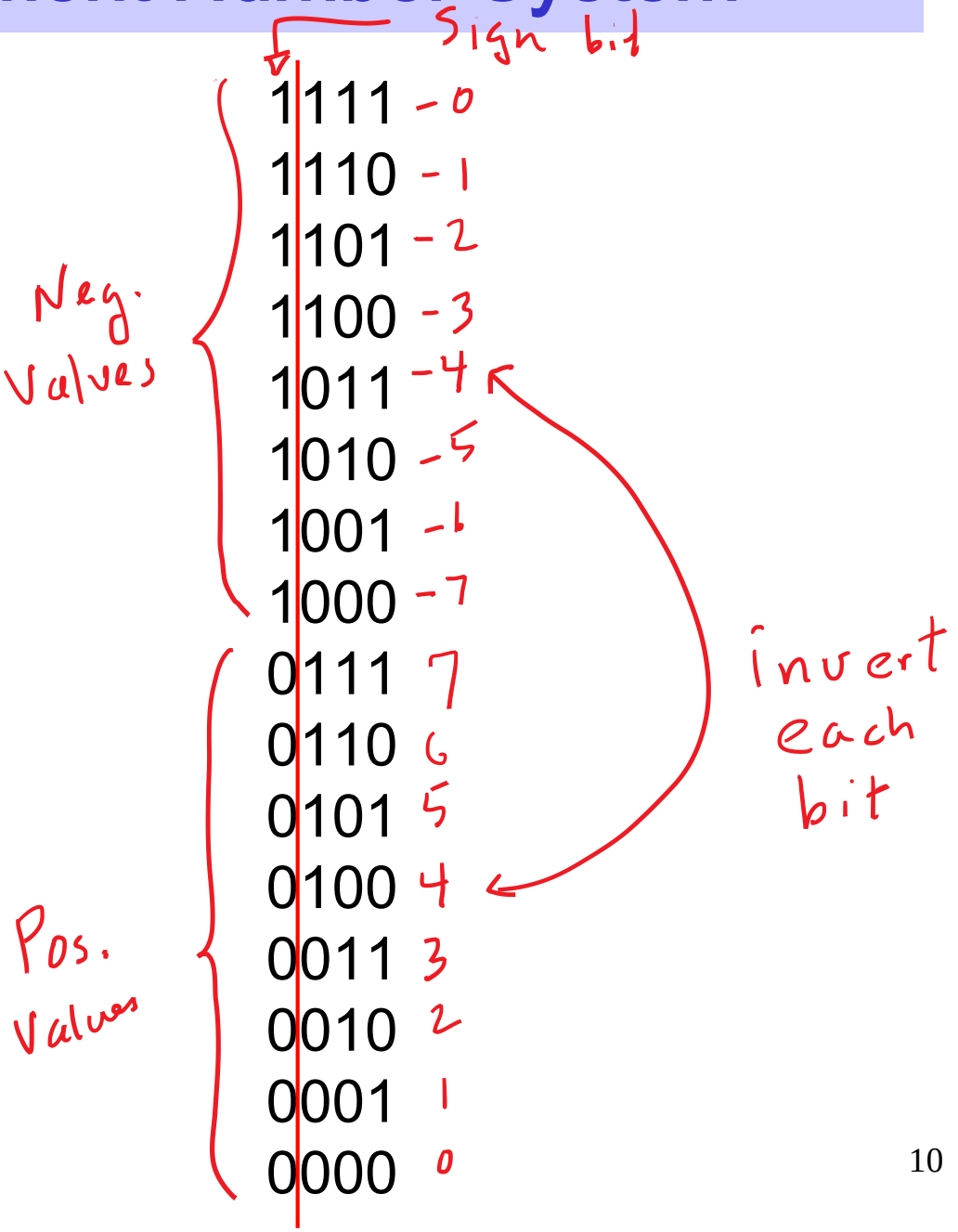
Assume a 4-bit **signed** number system using the **1's-complement system**.

What is range of values?

What is the most positive value?

What is the most negative value?

Generalize to n bits.



# 2's-complement System

A **2's-complement (or two's-complement) system** can represent both positive and negative values in binary. The most significant (leftmost) bit of the binary number is reserved as the **sign bit**. If the sign bit = 0, then the value is positive. If the sign bit = 1, then the number is negative. **Negative values are produced by inverting (complementing) each bit of the positive value and then adding 1.**

Examples:
Assume a 6-bit number system

| binary | decimal |
|---|---|
| **0**00010 | 2 |
| **1**11110 | -2 |
| | |
| **0**01111 | 15 |
| **1**10001 | -15 |
| | |
| **0**00000 | 0 |

sign bit (leftmost)

- **How many numbers** can be represented with 6 bits?
- What is the **most positive value** that can be represented in the two's-complement system with 6 bits?
  Answer: 011111 = 31 (base 10)
- What is the **most negative value** that can be represented in the two's-complement system with 6 bits?
  Answer: 100000 = - 32 (base 10)
- What are the **advantages** of the two's complement system? Answer: 1) simplified circuitry for addition and subtraction, 2) single code for zero (000000)

# 2's-complement Number System

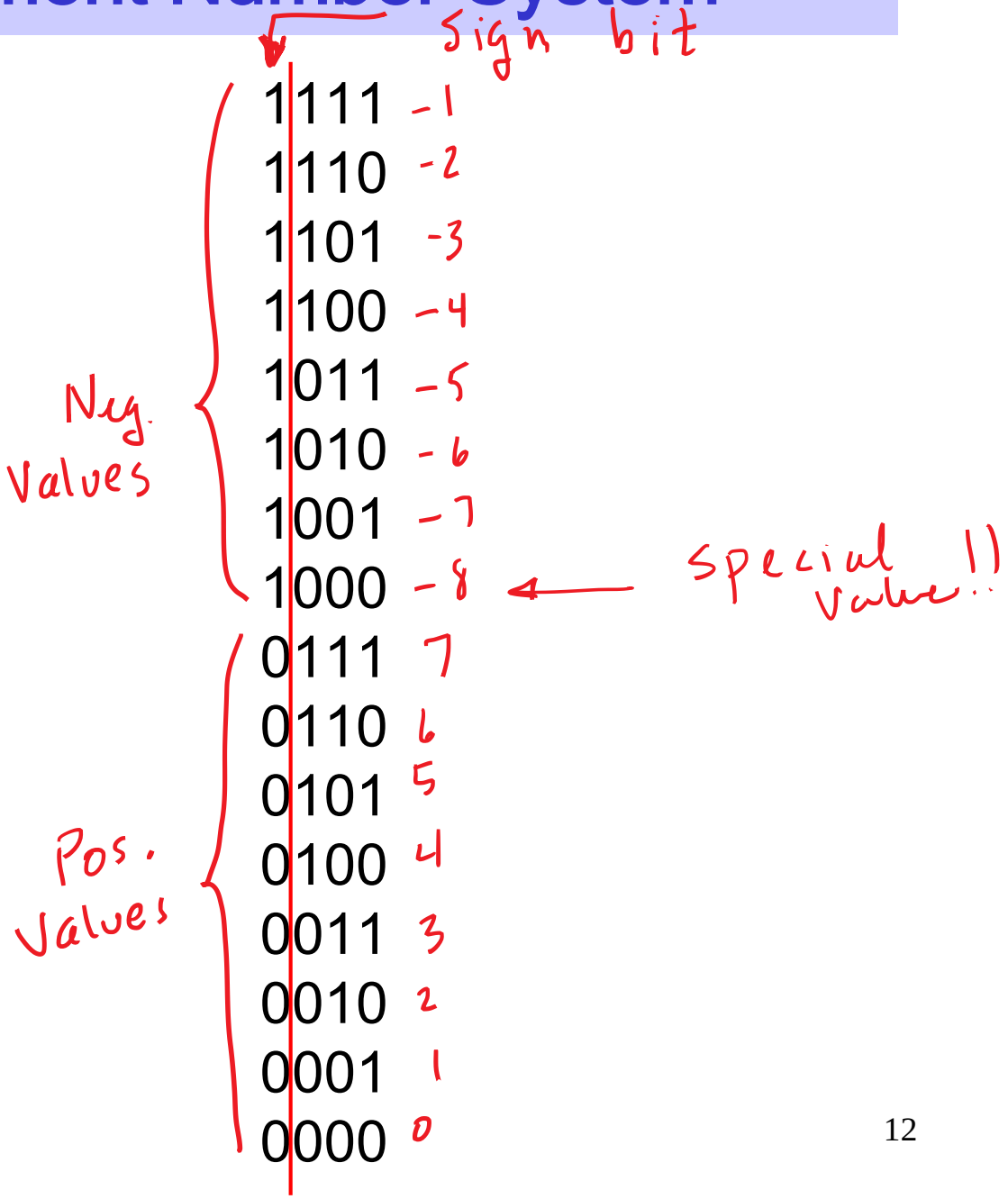
Assume a 4-bit **signed** number system using the **2's-complement system**.

What is range of values?

What is the most positive value?

What is the most negative value?

Generalize to n bits.



# Practice Exercises

Assume a **5-bit system**. This means each answer must consist of 5 bits (no more, no less). Any carry bits are discarded. Generalize to n bits.

| Decimal | Signed-magn. | 1's-compl. | 2's-compl. |
|---|---|---|---|
| 10 | 01010 | 01010 | 01010 |
| -10 | 11010 | 10101 | 10110 |
| 2 | 00010 | 00010 | 00010 |
| -2 | 10010 | 11101 | 11110 |
| 15 | 01111 | 01111 | 01111 |
| -15 | 11111 | 10000 | 10001 |

sign bit

SAME!

SAME!

SAME!

# Range of Values - 1

Consider a **4-bit number system**:

| Decimal | Signed-magn. | 1's-compl. | 2's-compl. |
|---|---|---|---|
| 7 | +7 0111 | +7 0111 | +7 0111 |
| 6 | 0110 | 0110 | 0110 |
| 5 | 0101 | 0101 | 0101 |
| 4 | 0100 | 0100 | 0100 |
| 3 | 0011 | 0011 | 0011 |
| 2 | 0010 | 0010 | 0010 |
| 1 | 0001 | 0001 | 0001 |
| 0 | 0000, 1000 | 0000, 1111 | 0000 |
| -1 | 1001 | 1110 | 1111 |
| -2 | 1010 | 1101 | 1110 |
| -3 | 1011 | 1100 | 1101 |
| -4 | 1100 | 1011 | 1100 |
| -5 | 1101 | 1010 | 1011 |
| -6 | 1110 | 1001 | 1010 |
| -7 | 1111 = -7 | 1000 = -7 | 1001 |
| -8 | ------ | ------ | 1000 ← -8 |

positive no.'s are same in all 3 systems

# Range of Values - 2

Always start with binary!

Assume **a <u>5-bit system</u>** with **<u>2's complement</u>** representation. Generalize!

group of 4!

| | **binary** | **decimal** | **hex** |
|---|---|---|---|
| Most positive value | 01111 | 15 | 0F |
| ... | | | |
| 2 | 00010 | 2 | 02 |
| 1 | 00001 | 1 | 01 |
| 0 | 00000 | 0 | 00 |
| -1 | 11111 | -1 | 1F |
| -2 | 11110 | -2 | 1E |
| ... | | | |
| Most neg. value + 1 | 10001 | -15 | 11 |
| Most negative value | 10000 | -16 | 10 |

Most neg #

# Range of Values - 3

Assume a **6-bit system** with **2's complement** representation. Generalize!

| | binary | decimal | hex |
|---|---|---|---|
| Most positive value | 011111 | 31 | 1F |
| 2 | 000010 | 2 | 02 |
| 1 | 000001 | 1 | 01 |
| 0 | 000000 | 0 | 00 |
| -1 | 111111 | -1 | 3F |
| -2 | 111110 | -2 | 3E |
| Most neg. value + 1 | 100001 | -31 | 21 |
| Most negative value | 100000 | -32 | 20 |

Most neg. #

Note: never use a negative sign( - ) with binary, hex, or octal numbers. The sign bit serves as a negative sign. So, -2 is just 111110, not -111110.

# Range of Values - 4

Assume an **8-bit system** with **2's complement** representation. Generalize!

| | binary | decimal | hex |
|---|---|---|---|
| Most positive value | 01111111 | 127 | 7F |
| 2 | 00000010 | 2 | 02 |
| 1 | 00000001 | 1 | 01 |
| 0 | 00000000 | 0 | 00 |
| -1 | 11111111 | -1 | FF |
| -2 | 11111110 | -2 | FE |
| Most neg. value + 1 | 10000001 | -127 | 81 |
| Most negative value | 10000000 | -128 | 80 |

# Range of Values - 5

Assume a **16-bit system** with **2's complement** representation. Generalize!

| | binary | decimal | hex |
|---|---|---|---|
| Most positive value | 0111111111111111 | 32767 | 7FFF |
| 2 | 0000000000000010 | 2 | 2 |
| 1 | 0000000000000001 | 1 | 1 |
| 0 | 0000000000000000 | 0 | 0 |
| -1 | 1111111111111111 | -1 | FFFF |
| -2 | 1111111111111110 | -2 | FFFE |
| Most neg. value + 1 | 1000000000000001 | -32767 | 8001 |
| Most negative value | 1000000000000000 | -32768 | 8000 |

Most neg #

# Practice Exercises

#1 The value, -1, is represented in a 5-bit 2's complement system as

a) 11110 b) 11111 c) 00001 d) 10001 e) -00001

#2 The value, -1, is represented in a 5-bit 2's complement system in hex is

a) 11 b) 1E c) 01 d) 1F

#3 The value, +3, is represented in a 5-bit 1's complement system as

a) 10011 b) 00010 c) 10011 d) 00011

sign bit

#4 The most negative value in a 5-bit, 2's complement signed number system is

a) -15 b) -16 c) -8 d) -7

# Penn State Abington
# CMPEN 271
# Lecture Set #14

**Topics:**

- Signed Number Systems — **Video part 1 of 4**
- Binary Subtraction — **Video part 2 of 4** ←
- Overflow, Digital Circuits
- Arithmetic Logic Unit (ALU) — **Video part 3 of 4**
- HDLs: VHDL, Verilog
- HW #7 (display negative value) — **Video part 4 of 4**
- Summary

# Subtraction with 2's complement

General approach:

A - B = A + (-B) = A + 2's-complement of B

Example:

Given an **6-bit, 2's complement system**, perform the following **subtraction**

| | | | | |
|---|---|---|---|---|
| **23** | same as | **23** | same as | **010111** (= 23) |
| - **10** | | + **-10** | | + **110110** (= -10 = 2's compl of 10) |
| ?? | | | | **1001101** = +13 (discard carry bit) |

*sign bit*

*discard carry bit.*
*6-bit system only here.*

# Subtraction with 2's complement

Example:

**Given a 6-bit, 2's complement system, perform the following subtraction**

| | | | | |
|---|---|---|---|---|
| 23 | same as | 23 | same as | 010111 ( = 23) |
| - 25 | | + -25 | | + 100111 ( = 2's compl of 25) |
| ?? | | | | 111110 = -2 (no carry bit) |

Notes: 25 (base 10) = 011001(base 2)
-25 = 2's complement of 25 = 100111

GENERALIZE!

- Using the 2's complement system, subtraction is converted to an addition operation. This simplifies circuitry. Most computers use 2's complement representation of signed (positive and negative) numbers.
- Can you take the negative (2's compl.) of a negative number? Yes.

# Number System Overflow - 1

Given a **6-bit, 2's complement system**, the **most positive value** that can be represented = 31, and the **most negative value** is -32.

**Question**: What happens in a 6-bit system when we add 2 numbers such that the sum exceeds the most positive value. For example 18 + 15 = 33. The values 18 and 15 can be represented in a 6-bit system, but 33 cannot be represented. This is called an **overflow**. Avoid this.

| | |
|---|---|
| 18 | 010010 (= 18) |
| + 15 | + 001111 (= 15) |
| ?? | 100001 (Negative! = -31 !) |

| | |
|---|---|
| 31 | 011111 ( = 31) …another example |
| + 1 | + 000001 ( = 1) |
| ?? | 100000 ( = -32 !!!! overflow) |

# Number System Overflow - 2

In **C/C++ (e.g. Visual Studio C++)** the "int" data type may be represented as a signed, 16-24 bit value using 2's complement (this varies from compiler to compiler). Find the most positive value that can be represented . Also, find the most negative value that can be represented.

Consider the following C/C++ code segment:

```
int main()
{
   int x;
   x = ???most positive number???  + 1;
   cout << x;          //What is the displayed value?
 }                     // Is there an error or warning message?
```

# Number System Overflow - 3

In **C/C++ (e.g. Visual Studio C++)** the "int" data type is represented as a signed, 20-24-bit value using 2's complement. Consider another "feature" of 2's complement.

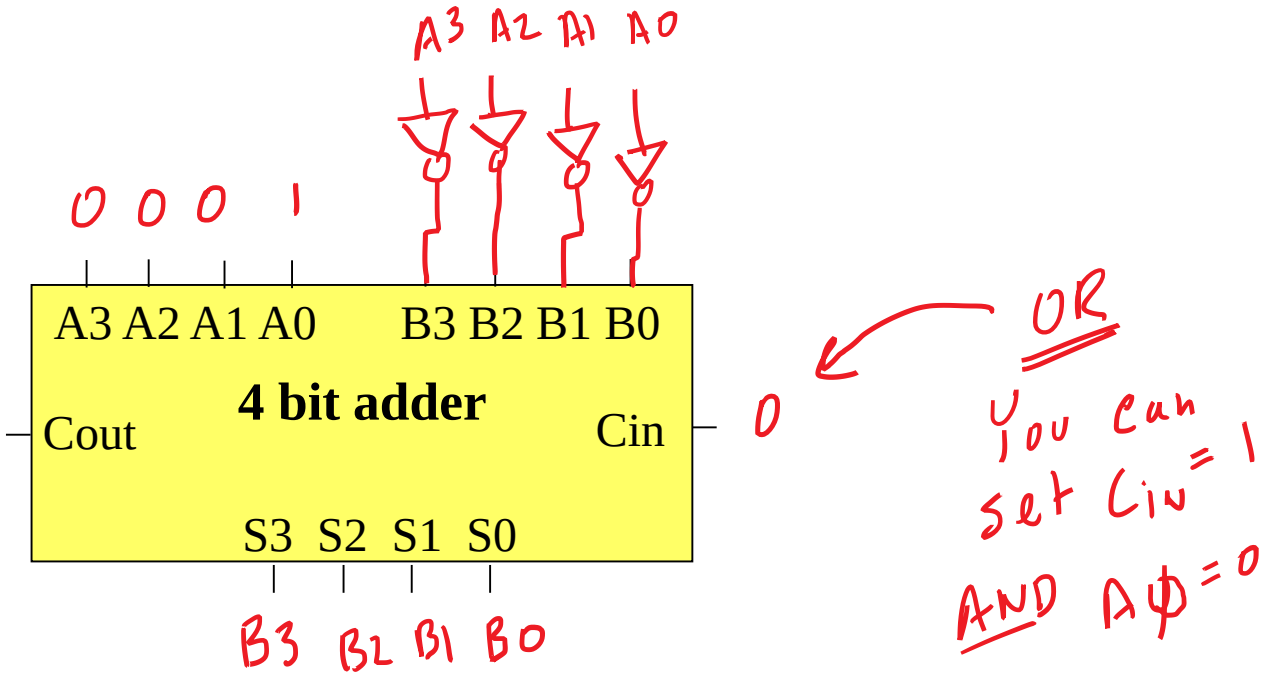
Consider the following C/C++ code segment:

```
int main()
{
   int x, y;
   x =  ?????most negative number?????;
   y = -x;
   cout << y;        //What is the displayed value?  Explain.
 }
```

# Digital Circuits

Exercise: Design a digital circuit to generate the 2's complement of a 4-bit binary number. There are 4 inputs (A3 A2 A1 A0) and 4 outputs (B3 B2 B1 B0).

Hint: use 4-bit adder and any additional gates as needed.



# Digital Circuits

$A \oplus 0 = A$

$A \oplus 1 = A'$

Design a digital circuit that will generate either the **sum or difference** of two, 4-bit signed numbers in a 4-bit system. If a control input, called X, is 0, then the circuit will perform the operation A + B. If the control input X = 1, then the circuit will perform the operation A - B ( = A + 2's-compl. B). There are 9 inputs and 4 outputs. Verify operation.

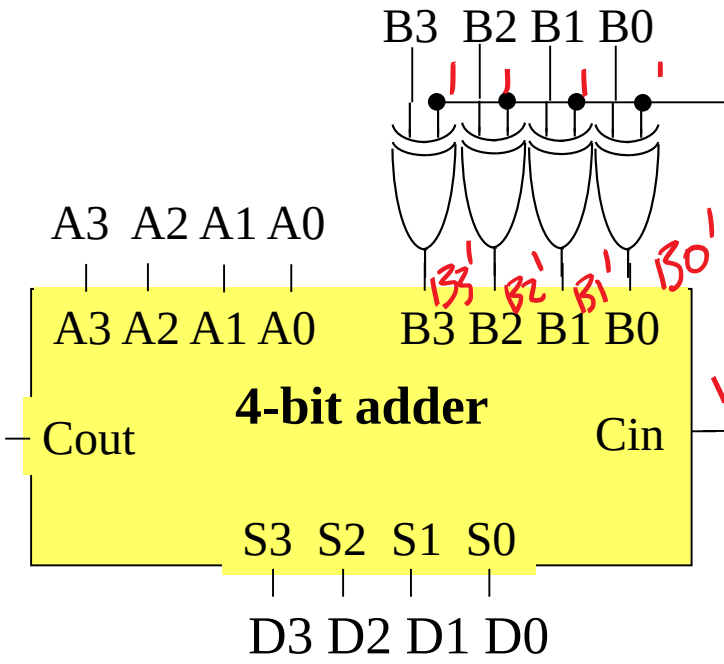

① When X = 0

A + B

② When X = 1

= A + 2scomp(B)

= A - B




Use XOR gates.


Select

# Penn State Abington
# CMPEN 271
# Lecture Set #14

**Topics:**

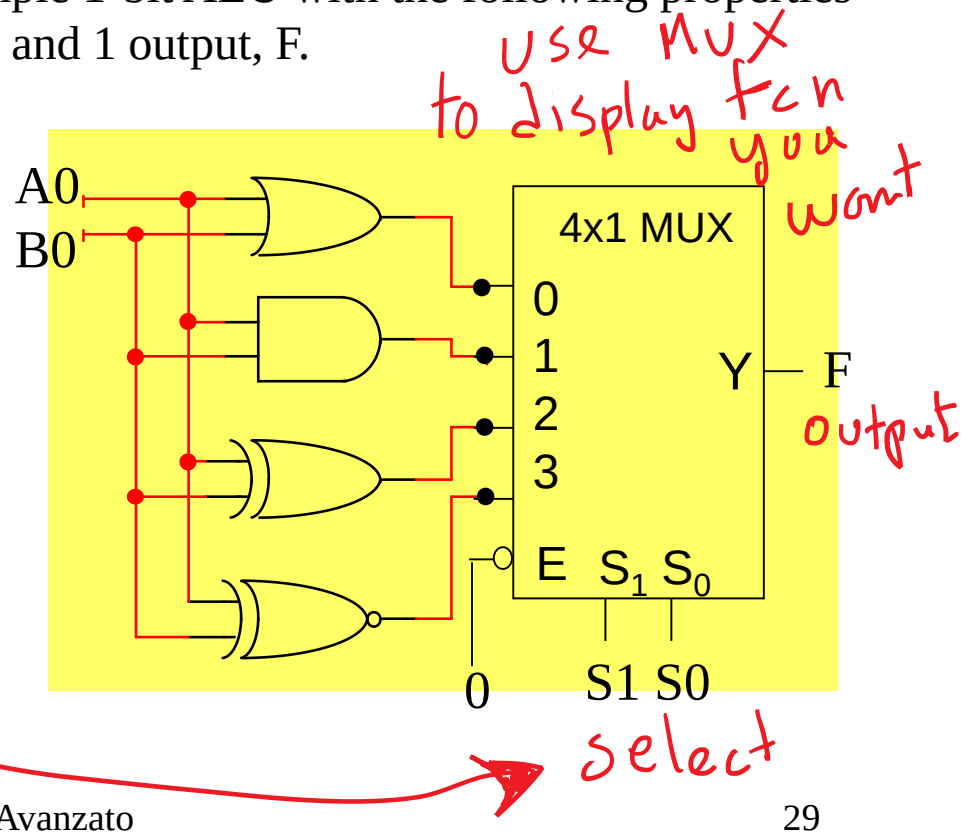
- Signed Number Systems — **Video part 1 of 4**
- Binary Subtraction — **Video part 2 of 4**
- Overflow, Digital Circuits
- Arithmetic Logic Unit (ALU) — **Video part 3 of 4** ←
- HDLs: VHDL, Verilog
- HW #7 (display negative value) — **Video part 4 of 4**
- Summary

# Arithmetic Logic Unit (ALU)

An **arithmetic logic unit** (ALU) is a combinational circuit that performs a set of logic and arithmetic functions based on control inputs. ALUs are also available as MSI devices (e.g. 74181, 74381, 74382). ALUs are built-in components of microprocessors (i.e. the ALU is in the µp chip itself).

Design Problem. Design and simulate a simple 1-bit ALU with the following properties given in the table below. There are 2 inputs and 1 output, F.

| Select Inputs | | Output function |
|---|---|---|
| S1 | S0 | |
| 0 | 0 | F = A0 + B0 |
| 0 | 1 | F = A0 * B0 |
| 1 | 0 | F = A0 xor B0 |
| 1 | 1 | F = A0 xnor B0 |

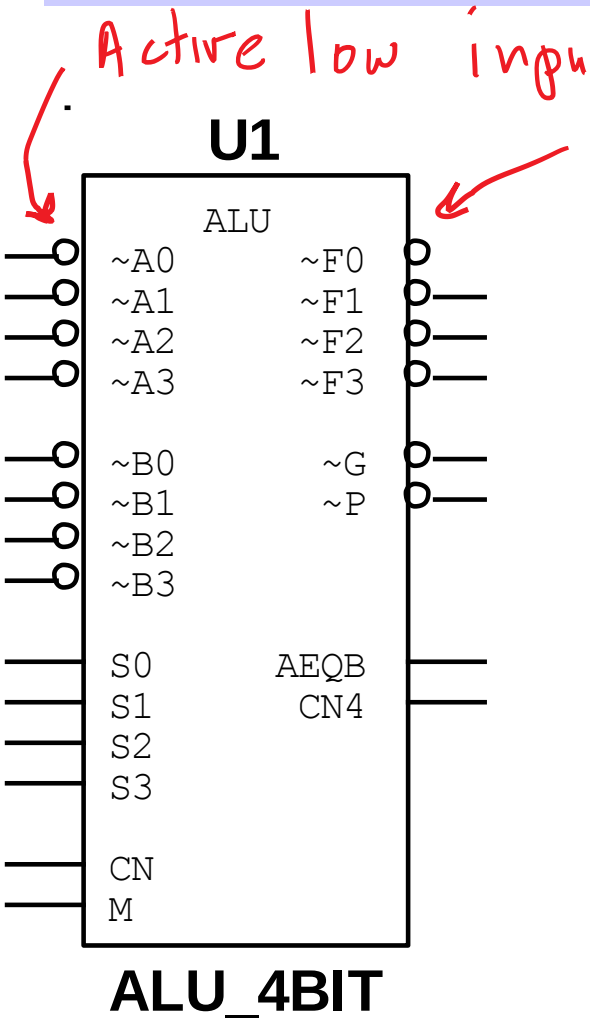


# Arithmetic Logic Unit (ALU)

Design Problem. Design and simulate a 4-bit ALU with the following properties. You may use MSI functions in the solution. A and B are both 4-bit numbers. There are 8 data inputs (A and B) and 3 control select inputs (s2, s1, s0). There are 4 outputs (X3 X2 X1 X0). (Cout output is optional.) Design must be well-organized. An ALU is an important internal component to a microprocessor.

| Select Inputs | | | Function Performed (output) |
|---|---|---|---|
| S2 | S1 | S0 | |
| 0 | 0 | 0 | A + B (use 4-bit adder) |
| 0 | 0 | 1 | A - B (A + 2's-compl. of B) |
| 0 | 1 | 0 | A + 1 (increment A by 1) |
| 0 | 1 | 1 | A - 1 (decrement A by 1) |
| 1 | 0 | 0 | A' (1's complement of A) |
| 1 | 0 | 1 | A xor B (bitwise) |
| 1 | 1 | 0 | A + B (bitwise OR; not addition) |
| 1 | 1 | 1 | A * B (bitwise AND; not multiplication) |

# Arithmetic Logic Unit (ALU)




- 74181 is a TTL ALU chip
- Was the first complete ALU on a single chip
- Was very popular in older days of digital design
- Largely replaced by microprocessor technology
- Logic is incorporated into other digital circuits
- Available in MultiSim
- Notice the active-low inputs and outputs

# Arithmetic Logic Unit (ALU)

| Mode Select Inputs | | | | Active LOW Operands & $F_n$ Outputs | | Active HIGH Operands & $F_n$ Outputs | |
|---|---|---|---|---|---|---|---|
| L = 0 | H = 1 | | | Logic | Arithmetic (Note 2) | Logic | Arithmetic (Note 2) |
| S3 | S2 | S1 | S0 | (M = H) | (M = L) ($C_n$ = L) | (M = H) | (M = L) ($C_n$ = H) |
| L | L | L | L | $\overline{A}$ | A minus 1 | $\overline{A}$ | A |
| L | L | L | H | $\overline{AB}$ | AB minus 1 | $\overline{A} + B$ | A + B |
| L | L | H | L | $\overline{A} + \overline{B}$ | $A\overline{B}$ minus 1 | $\overline{A}\,B$ | $A + \overline{B}$ |
| L | L | H | H | Logic 1 | minus 1 | Logic 0 | minus 1 |
| L | H | L | L | $\overline{A} + \overline{B}$ | A plus $(A + \overline{B})$ | $\overline{AB}$ | A plus $A\overline{B}$ |
| L | H | L | H | $\overline{B}$ | AB plus $(A + \overline{B})$ | $\overline{B}$ | $(A + B)$ plus $A\overline{B}$ |
| L | H | H | L | $\overline{A} \oplus \overline{B}$ | A minus B minus 1 | $A \oplus B$ | A minus B minus 1 |
| L | H | H | H | $A + \overline{B}$ | $A + \overline{B}$ | $A\overline{B}$ | AB minus 1 |
| H | L | L | L | $\overline{A}\,B$ | A plus $(A + B)$ | $\overline{A} + B$ | A plus AB |
| H | L | L | H | $A \oplus B$ | A plus B | $\overline{A} \oplus \overline{B}$ | A plus B |
| H | L | H | L | B | $A\overline{B}$ plus $(A + B)$ | B | $(A + \overline{B})$ plus AB |
| H | L | H | H | $\overline{A} + B$ | A + B | AB | AB minus 1 |
| H | H | L | L | Logic 0 | A plus A (Note 1) | Logic 1 | A plus A (Note 1) |
| H | H | L | H | $A\overline{B}$ | AB plus A | $A + \overline{B}$ | $(A + B)$ plus A |
| H | H | H | L | AB | $A\overline{B}$ minus A | A + B | $(A + \overline{B})$ plus A |
| H | H | H | H | A | A | A | A minus 1 |

do not worry about details here

**Note 1:** Each bit is shifted to the next most significant position.

**Note 2:** Arithmetic operations expressed in 2s complement notation.

74181 ALU specs

# Hardware Description Languages (HDLs)

- How to design, simulate and synthesize **complex digital circuits**?
- **Schematics Capture** (e.g. LogicWorks, Pspice, others) include MULTISIM
- **Hardware Description Languages** (text-based; alternative to schematics)

progr. languages

  - **VHDL** (VHSIC Hardware Description Language). Developed by US DoD. Now a IEEE standard; similar to ADA syntax
  - **Verilog** (pronounced "very log"). IEEE standard; similar to C syntax. (Verilog is generally perceived as easier to learn than VHDL)

- **FPGAs (field programmable gate arrays) >100,000 gates**

hardware

# Hardware Description Languages (Verilog)

Example of a simple Full Adder expressed in Verilog

Similar to C/C++ language

```
module addbit (
a      , // first input
b      , // Second input
ci     , // Carry Input
sum    , // Sum output
co       // Carry output
);
// Input Declaration
input   a  ;
input   b  ;
input   ci ;
// Ouput Declaration
output  sum;
output  co ;
// port data types
wire    a  ;
wire    b  ;
wire    ci ;
wire    sum;
wire    co ;

// Code starts Here
assign {co,sum} = a + b + ci;

endmodule // End Of Module addbit
```



Source: deeps.org

What is Verilog?

How does Verilog compare to C++?

What is the advantage of a HDL language such as Verilog? CAN Represent large, complex circuits

Compare HDL such as Verilog to schematics capture, such as MultiSim?

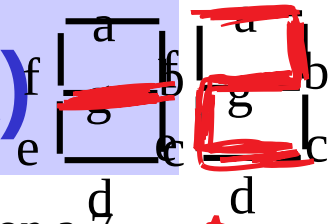
# Penn State Abington
# CMPEN 271
# Lecture Set #14

**Topics:**

- Signed Number Systems — **Video part 1 of 4**
- Binary Subtraction — **Video part 2 of 4**
- Overflow, Digital Circuits
- Arithmetic Logic Unit (ALU) — **Video part 3 of 4**
- HDLs: VHDL, Verilog
- HW #7 (display negative value) — **Video part 4 of 4** ←
- Summary

2 problems

# Homework #7 (due in 1 week)




#1 -- Design a circuit to **convert a 4-bit, 2's complement number** to display on a 7-segment LED with negative sign; use a separate 7-segment display to display the minus (neg.) sign (when appropriate); design and simulate using LogicWorksor MultiSIM. Pick appropriate test cases. There are 4 input switches (label lsb).
For example, if the input is 0010 (+2) then the first 7-segment display will be blank, and the second 7-segment display will show a "2". If the input is 1110 (= -2) then the first 7-segment will display a "-" sign and the second 7-segment will display a "2". (Do not show a "+" sign if the input is positive; leave blank for positive values.)

#2 – Design, simulate and test a circuit to **multiply a 3-bit (unsigned) number**: A2, A1, A0 by a **2-bit (unsigned) number** B1, B0. Output is unsigned binary (use probes, do not use 7-segment displays in this problem ). Carefully label all inputs and outputs. Pick appropriate test cases. Use MultiSIM or LogicWorks. This solution has a total of 5 input switches. Include appropriate test cases.

All pos. values

No 7-segm displays in this problem

- Use LogiWorks or MultiSIM simulation; include all test cases
- Upload to Canvas -- **place each solution in appropriate Canvas drop box**
- Due in 1 week
- Hint: design the circuits with paper and pencil and think about design before simulating in MultiSIM/LogicWorks.
- Ask questions.

# Summary

- **Number systems in a digital systems** can be broadly categorized into: 1) unsigned number systems (only positive values are represented) and 2) signed number systems (both positive and negative values are represented).
- All signed number systems use the **left-most (msb) bit as a sign bit** (if the sign bit is 1, then it indicates a negative value, and if the sign bit is 0, then the value is positive).
- The **3 most common signed number systems** are: 1) 1's complement, 2) 2's complement, and 3) sign and magnitude.
- 2's complement is the **most common system** for digital computers to represent positive and negative numbers and it has advantages in addition operations and the fact hat there is only one binary representation for zero.
- **Positive numbers are represented in the exact same** way in all signed number systems (but the negative numbers will differ for each system).
- An **Arithmetic Logic Unit (ALU)** is the set of digital circuits which performs arithmetic (addition, subtraction, etc.) and logic operations in a CPU
- **Hardware Design Languages (HDL)** are used to describe and design complex digital circuits using special programming languages such as VHDL, and Verilog. (HDLs are different from LogicWorks & MultiSim which are primarily schematics capture).

# Further Reading

- Mano,Kime, Logic and Computer Design Fundamentals, 2nd edition, Prentice Hall, (Chapter 3).
- Tocci, R., Digital Systems, Prentice Hall, (Chapter 6, Arithmetic Operations).
- www.xilinx.com (manufacturer of FPGA, VHDL, products; web site contains tutorials)
- www.altera.com (FPGAs, VHDL, tutorials)
- www.xess.com (VHDL tutorials)
- www.digilentinc.com
- http://www.asic-world.com/verilog/index.html (Verilog/VHDL tutorials)

# Research Projects

- Find a free Verilog or VHDL compiler/simulator and use it to simulate CMPEN 271 circuits and homeworks.

- Investigate and explain how computers store floating-point numbers .

- Investigate and demonstrate overflow in a variety of high level language compilers such as C++, Java, C#, etc.

- Research and implement a digital circuit to convert a binary number into a BCD value. Once the value is in BCD format, it is quite easy to display on one or more 7-segment displays. For example, how would you design a binary to BCD converter so that you could display the outputs of a 3-bit by 3-bit binary multiplier on 2, 7-segment displays?