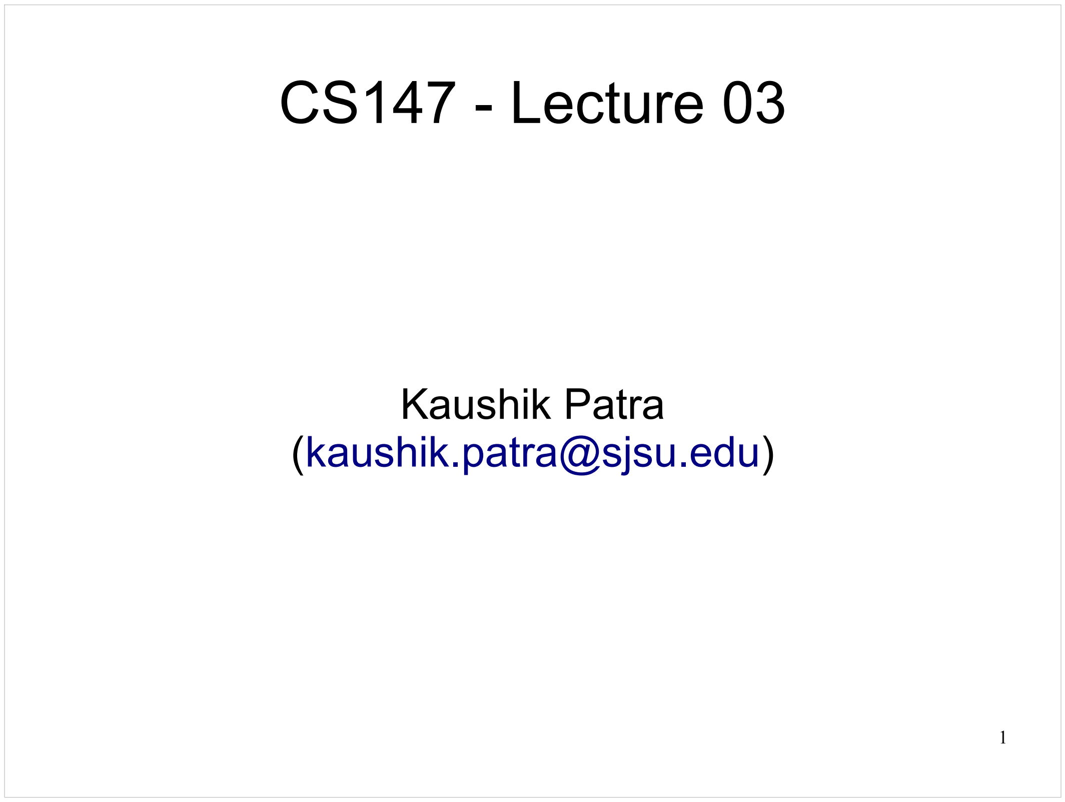


CS147 - Lecture 03

Kaushik Patra
(kaushik.patra@sjsu.edu)

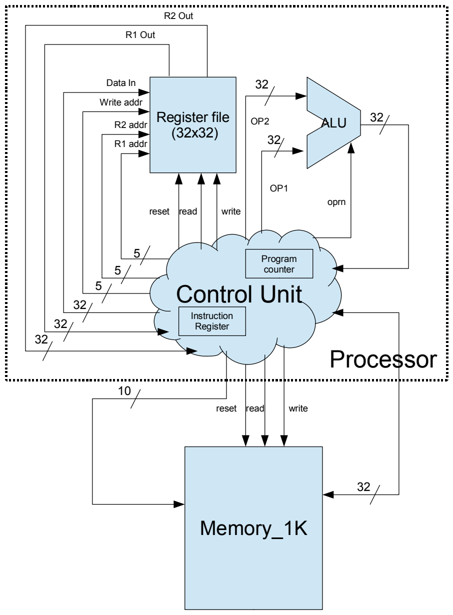
1

- Topics
  - Digital Synthesis
  - Number representation

[Chapter 1 (1-2, 1-3) of Logic & Computer Design Fundamentals, 4th Edition, M. Morris Mano, Charles R. Kime]

Digital Synthesis ...

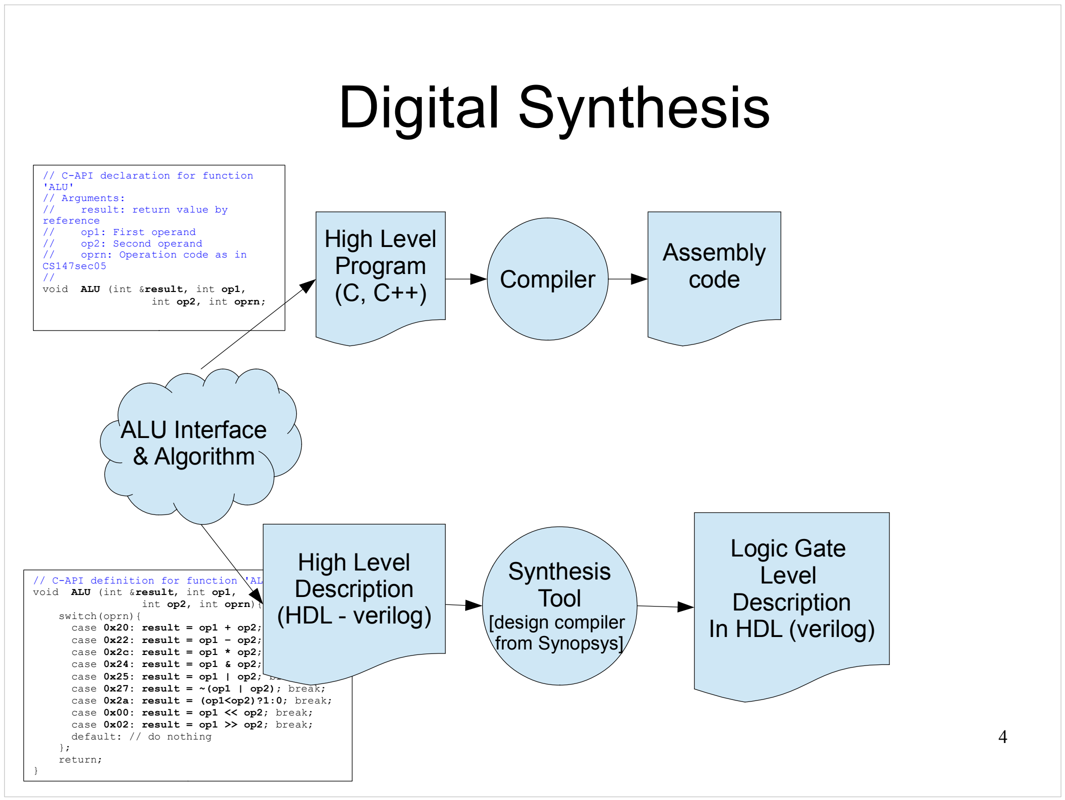
Digital Synthesis



- How to bring conceptual processor and memory system into physical reality?
- All these component concepts can be described at high level using hardware description language (HDL)
  - e.g. verilog, VHDL
  - Similar to programing language C/Java

3

- This simple question of 'how to turn an hardware concept into physical reality' has created multi-billion dollar industry which are broadly categorized as 'EDA' (Electronic Design Automation).
  - e.g. Synopsys, Cadence, MentorGraphics, etc.
  - These companies build and sell softwares those can bring the hardware concepts into physical reality.
- HDL provides language constructs similar to any other high level programing language like C or JAVA. Instead of describing algorithms in software context, HDL describes algorithm in hardware context.
  - We have already seen in previous lectures that architectural objects (like ALU or memory) can be described using C, which is nothing but a description in software context.
  - We have also seen in lab exercise the description of ALU using HDL.



Digital Synthesis

```
// C-API declaration for function
'ALU'
// Arguments:
//    result: return value by
reference
//    op1: First operand
//    op2: Second operand
//    oprn: Operation code as in
CS147sec05
//
void  ALU (int &result, int op1,
                 int op2, int oprn;
```

```
// C-API definition for function 'AL
void  ALU (int &result, int op1,
                 int op2, int oprn){
    switch(oprn){
      case 0x20: result = op1 + op2;
      case 0x22: result = op1 - op2;
      case 0x2c: result = op1 * op2;
      case 0x24: result = op1 & op2;
      case 0x25: result = op1 | op2;
      case 0x27: result = ~(op1 | op2); break;
      case 0x2a: result = (op1<op2)?1:0; break;
      case 0x00: result = op1 << op2; break;
      case 0x02: result = op1 >> op2; break;
      default: // do nothing
    };
    return;
}
```

High Level Program (C, C++)
Compiler
Assembly code
ALU Interface & Algorithm
High Level Description (HDL - verilog)
Synthesis Tool [design compiler from Synopsys]
Logic Gate Level Description In HDL (verilog)
4

- The very first step towards turning hardware concept into reality is to convert high level description of the hardware (e.g. Behavioral level description – as in alu.v in project 01) into an equivalent logic gate level description.

  - This is similar to the compilation process that translate high level language into equivalent assembly level language.

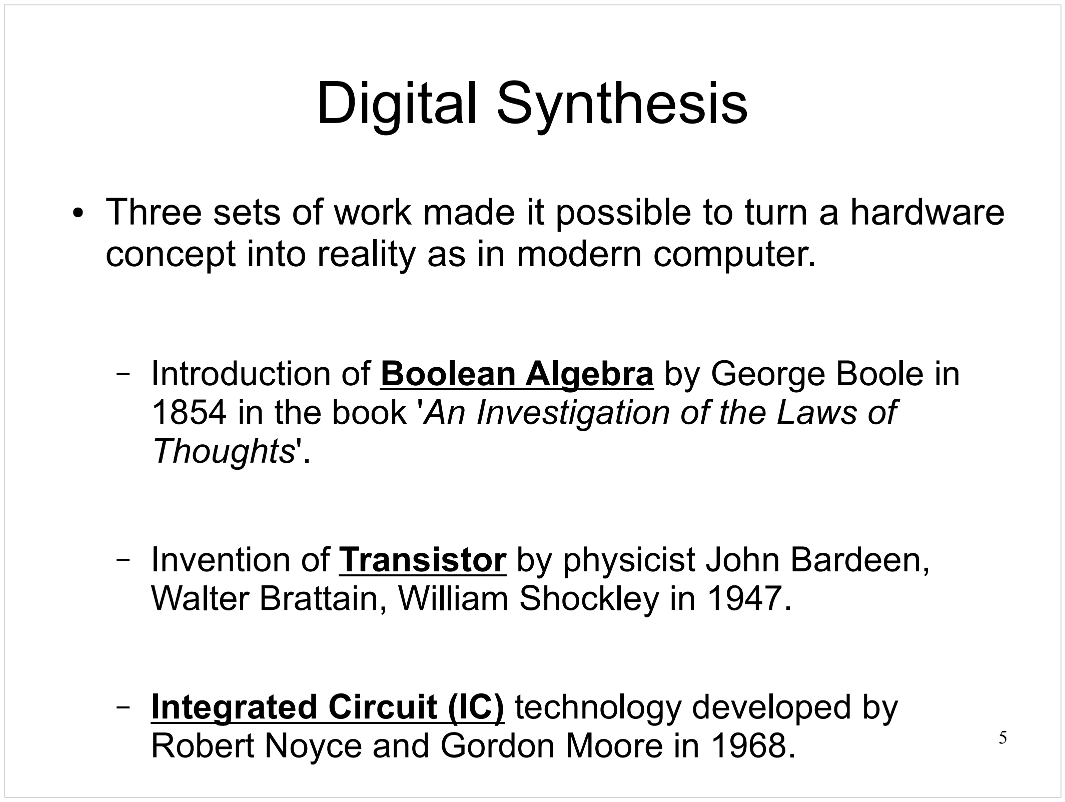
  - The process of translating high level hardware description into equivalent logic gate level description is called digital synthesis.

  - Logic gates are hardware entities or objects that implements Boolean logic functions – like not, nand, nor (these are fundamental logic gates from hardware point of view). There are more functions like and, or, xor, xnor.

  - Logic gates (implementing corresponding fundamental boolean functions) are the basic entities that a synthesis tool uses to translate an high level hardware description into a description that uses only logic gates,

- It is still in research stage to translate a model described in C language into an equivalent logic gate level description.

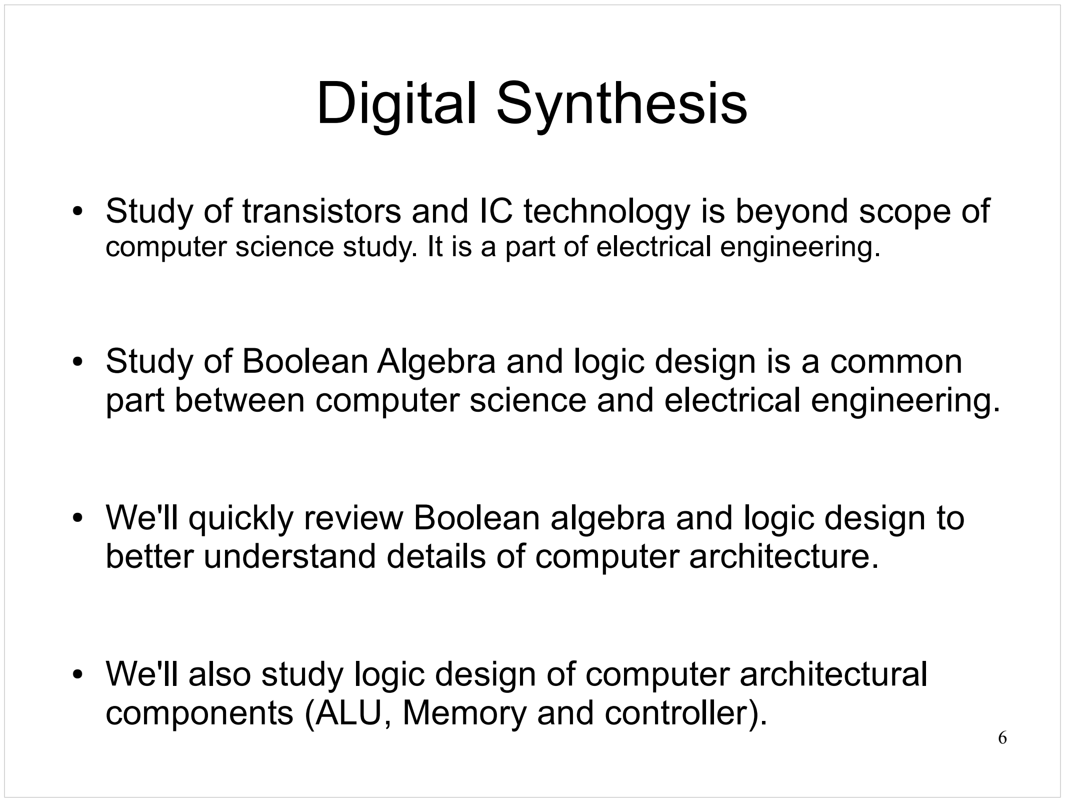
- The Verilog language haas been extended into System-verilog language which supports many C like syntax.



Digital Synthesis

- Three sets of work made it possible to turn a hardware concept into reality as in modern computer.
  - Introduction of **Boolean Algebra** by George Boole in 1854 in the book '*An Investigation of the Laws of Thoughts*'.
  - Invention of **Transistor** by physicist John Bardeen, Walter Brattain, William Shockley in 1947.
  - **Integrated Circuit (IC)** technology developed by Robert Noyce and Gordon Moore in 1968.

- Boolean algebra is the mathematical foundation of computer operation. Any software programs can be broken down into Boolean logic expression.
  - Since boolean logic only assumes two values – 0 and 1 (false and true) it is very easy to represent them in electrical context. Presence of current can be treated as 1 and absence of current can be treated as 0.
  - Boolean algebra is also known as 'Switching algebra' which defines some switching relation between variables which can assume values on or off (1 or 0).
- Transistor is an electronic switch using which any electric circuit can be turned off or on (equivalent to logic 0 or 1) by electrical signal. This means there is a possibility to electronically implement switching algebra using transistors. Computer can be thought of a electronic machine with billions of transistors implementing a very complex boolean function.
- Cheaper mass production of devices with billions of interconnected transistors is possible through IC technology. This technology is analogous to printing news paper in printing press. Like the invention of commercial printing press by Johannes Gutenberg in 1439 changed the world around us, the invention of IC technology changed the world around us. Each electronic gadget we use in our daily life is possible due to invention of IC technology.

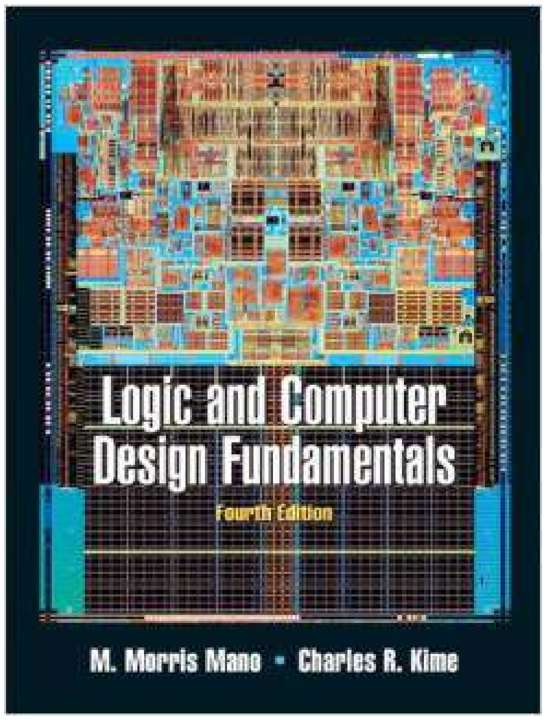


Digital Synthesis

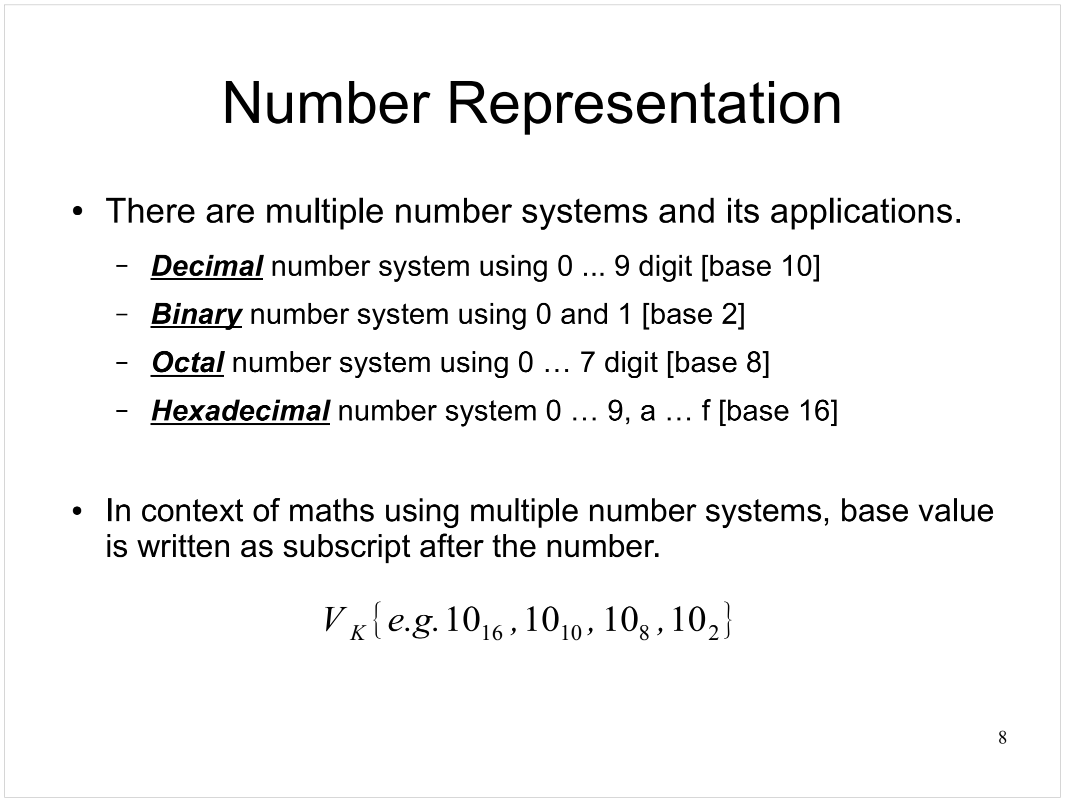
- Study of transistors and IC technology is beyond scope of computer science study. It is a part of electrical engineering.
- Study of Boolean Algebra and logic design is a common part between computer science and electrical engineering.
- We'll quickly review Boolean algebra and logic design to better understand details of computer architecture.
- We'll also study logic design of computer architectural components (ALU, Memory and controller).

6

- Module 2 – Boolean Algebra & Digital Logic Review will be taught from following book Chapter 2,3,4,5.
- **Logic and Computer Design Fundamentals (4th Edition)** by M. Morris Mano (Author), Charles Kime (Author)


Logic and Computer Design Fundamentals
Fourth Edition
M. Morris Mano · Charles R. Kime

Number Representation ...



Number Representation

- There are multiple number systems and its applications.
  - ***Decimal*** number system using 0 ... 9 digit [base 10]
  - ***Binary*** number system using 0 and 1 [base 2]
  - ***Octal*** number system using 0 … 7 digit [base 8]
  - ***Hexadecimal*** number system 0 … 9, a … f [base 16]
- In context of maths using multiple number systems, base value is written as subscript after the number.

$$V_K\{e.g.\, 10_{16}, 10_{10}, 10_8, 10_2\}$$

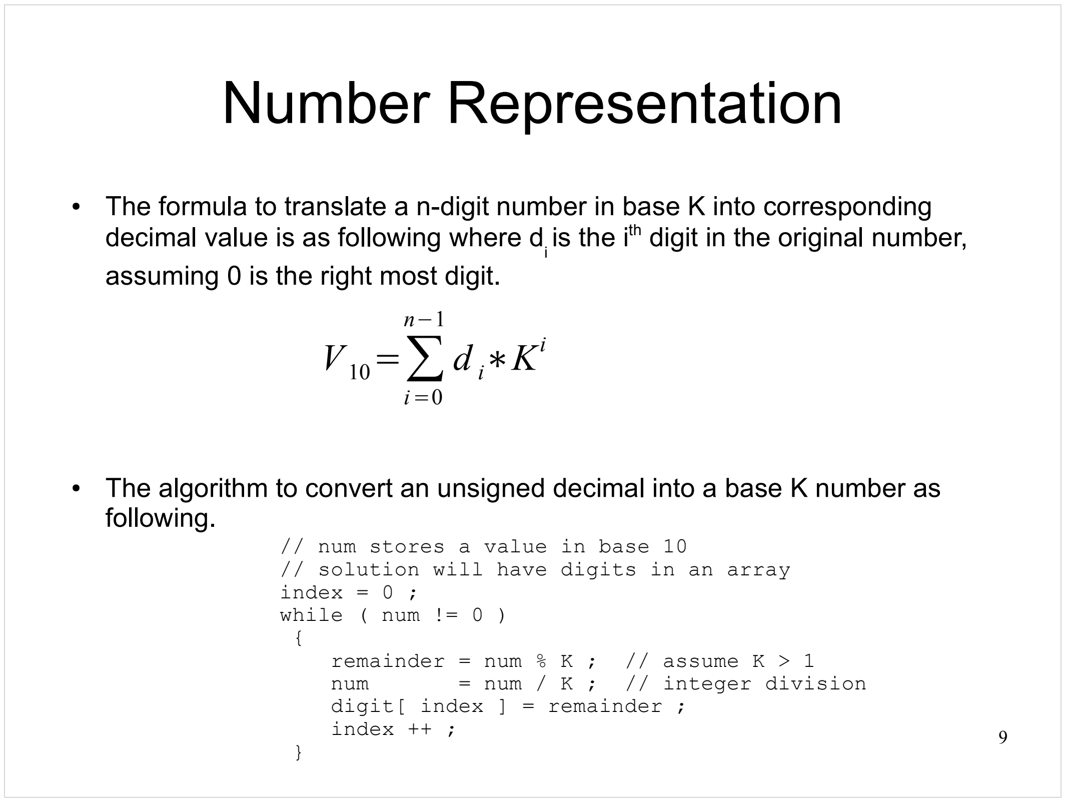
8

- Decimal number system is the most natural number system that has been adopted by human to perform mathematics.

- Binary number system has been adopted in electronic computation since the digit value can assume only two values – 0 and 1. Representing just 0 and 1 is very in electronic domain. Presence of current or voltage can be used as 1 and absence of the same can be used as 0.

- Octal number system was adopted in context of computer systems with 12, 24 or 36 bit word length (e.g. PDP-3, ICL 1900, IBM Main frame, etc.). This is to represent binary numbers in a very compact format. This system takes 3 bits and group them into a octal digit. For example: **111**101**110**100_2 is 7564_8.

- Most modern computer uses 8, 16, 32 or 64 bit as word length. To represent the binary number in compact form Hexadecimal system is adopted. This system takes group of 4 bits and and represent them in single hex digit. For example: **1111**0101**1100**0011_2 is f5c3_{16} .



Number Representation

- The formula to translate a n-digit number in base K into corresponding decimal value is as following where d_i is the i^{th} digit in the original number, assuming 0 is the right most digit.

$$V_{10} = \sum_{i=0}^{n-1} d_i * K^i$$

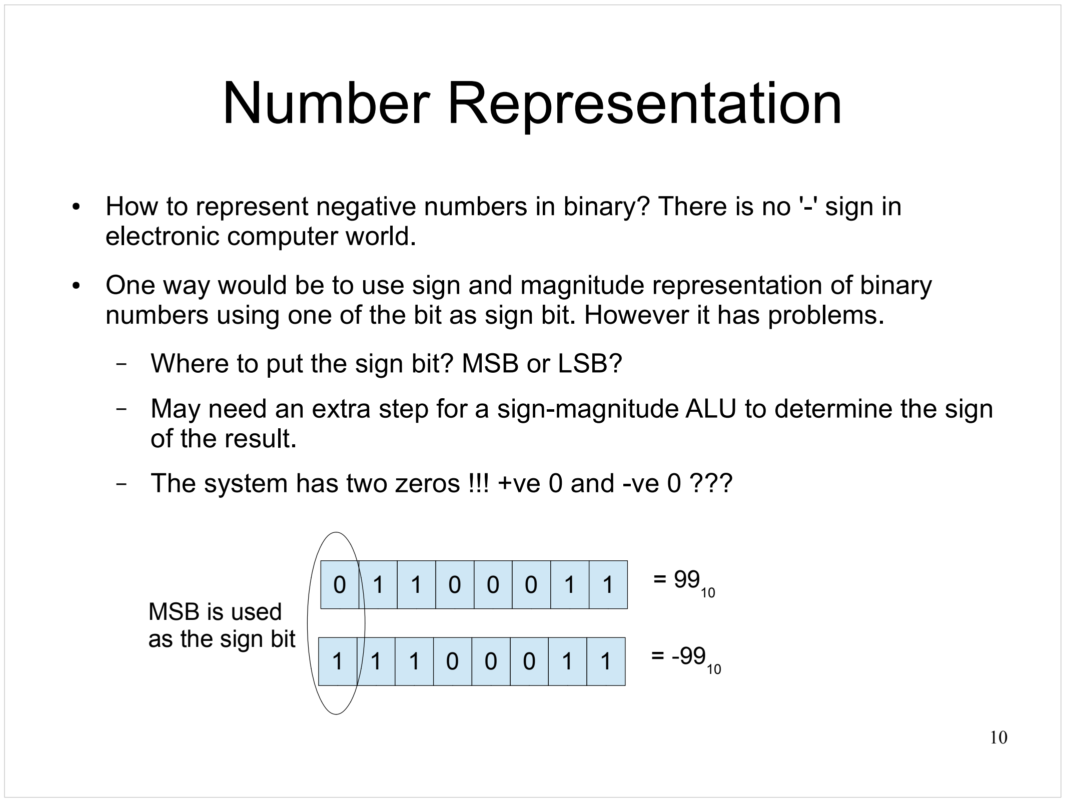
- The algorithm to convert an unsigned decimal into a base K number as following.

```
// num stores a value in base 10
// solution will have digits in an array
index = 0 ;
while ( num != 0 )
 {
    remainder = num % K ;  // assume K > 1
    num       = num / K ;  // integer division
    digit[ index ] = remainder ;
    index ++ ;
 }
```

9

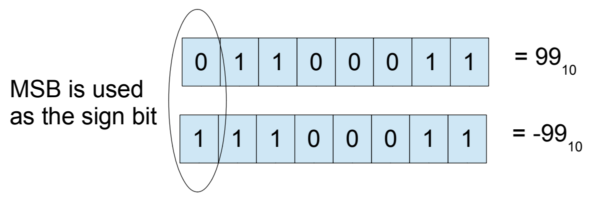
- Decimal value of 1010_2 is $(1*2^3 + 0*2^2 + 1*2^1 + 0*2^0)_{10} = 10_{10}$
- Hexadecimal value of 1482_{10} is determined as following
  - Num = 1482; H[0] = num % 16 = 10 or A_{16}; num = num / 16 = 92;
  - Num = 92; H[1] = num % 16 = 12 or C_{16}; num = num /16 = 5;
  - Num = 5; H[1] = num % 16 = 5 or 5_{16}; num = num /16 = 0;
  - Hence the hexadecimal is $5CA_{16}$.
- Decimal value of $5CA_{16}$ is $(5*16^2 + 12*16^1 + 10*16^0)_{10} = 1482_{10}$

•The algorithm snippet has been take from the site http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/toBaseK.html. More details can be found in that note.



Number Representation

- How to represent negative numbers in binary? There is no '-' sign in electronic computer world.
- One way would be to use sign and magnitude representation of binary numbers using one of the bit as sign bit. However it has problems.
  - Where to put the sign bit? MSB or LSB?
  - May need an extra step for a sign-magnitude ALU to determine the sign of the result.
  - The system has two zeros !!! +ve 0 and -ve 0 ???



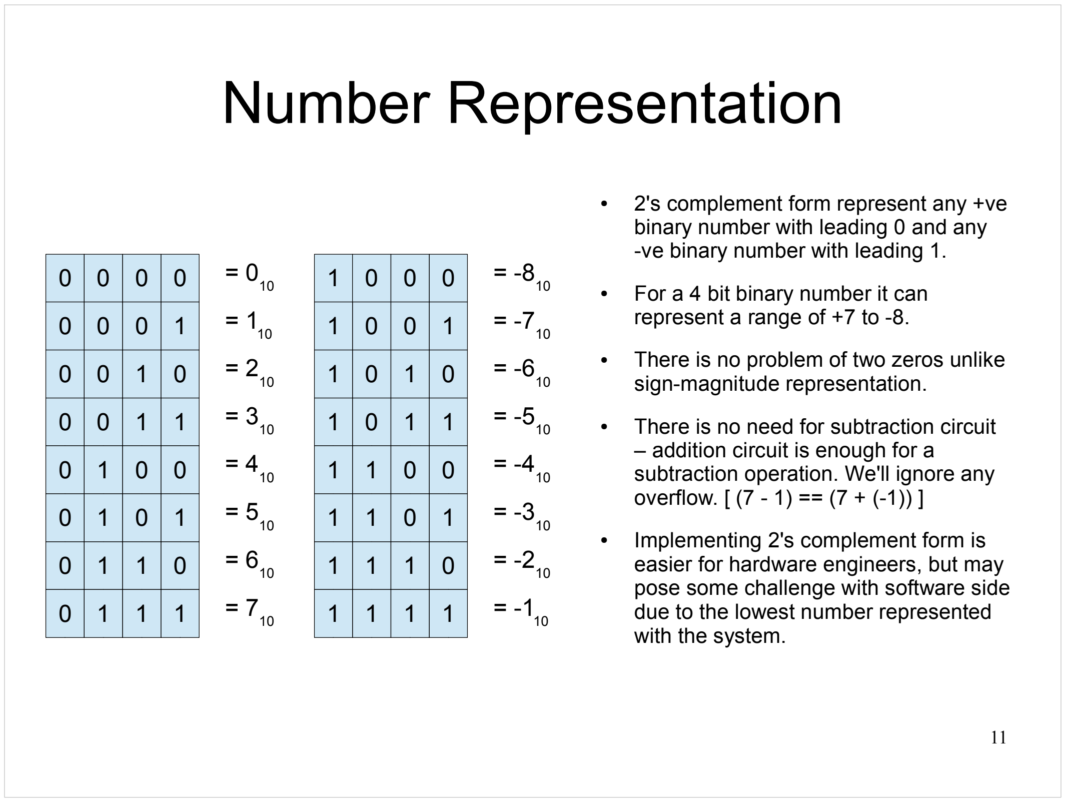
10

- MSB = **M**ost **S**ignificant **B**it, which is the left most bit of a binary number.
- LSB = **L**east **S**ignificant **B**it, which is the right most bit of a binary number.

- Early computer used this sign-magnitude approach. However, it presented problem for both hardware engineers (adding extra steps to determine result's sign bit) and software engineers (what to do with +ve 0 and -ve 0). Extra circuitry was needed of subtraction operation.

- In search of a better representation binary subtraction operation was carefully observed. If there is a need of subtracting a bigger number from a smaller one, what do we do? We start borrow from leading zeros which would result in string of 1s.

- It was then made that any number with leading zeros will be +ve and with ones will be -ve. This is a 2's complement form which is now used in every computer.



Number Representation

0	0	0	0	= 0_{10}
0	0	0	1	= 1_{10}
0	0	1	0	= 2_{10}
0	0	1	1	= 3_{10}
0	1	0	0	= 4_{10}
0	1	0	1	= 5_{10}
0	1	1	0	= 6_{10}
0	1	1	1	= 7_{10}

1	0	0	0	= -8_{10}
1	0	0	1	= -7_{10}
1	0	1	0	= -6_{10}
1	0	1	1	= -5_{10}
1	1	0	0	= -4_{10}
1	1	0	1	= -3_{10}
1	1	1	0	= -2_{10}
1	1	1	1	= -1_{10}

- 2's complement form represent any +ve binary number with leading 0 and any -ve binary number with leading 1.
- For a 4 bit binary number it can represent a range of +7 to -8.
- There is no problem of two zeros unlike sign-magnitude representation.
- There is no need for subtraction circuit – addition circuit is enough for a subtraction operation. We'll ignore any overflow. [(7 - 1) == (7 + (-1))]
- Implementing 2's complement form is easier for hardware engineers, but may pose some challenge with software side due to the lowest number represented with the system.

11

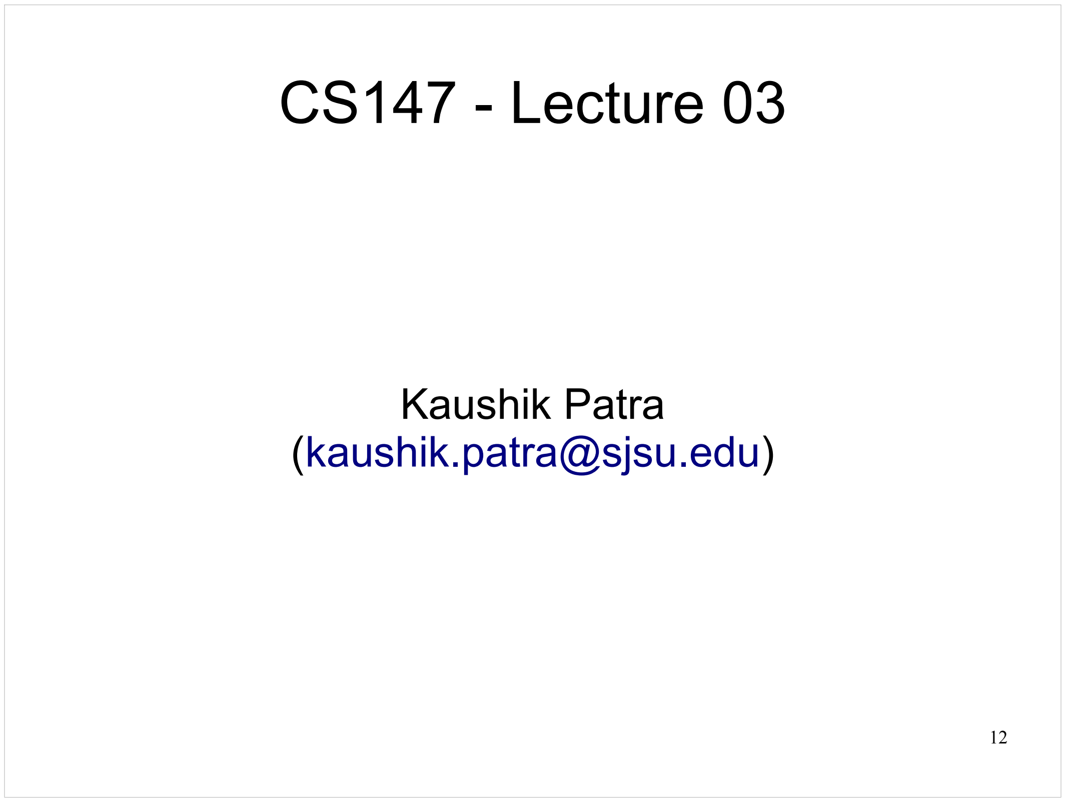
- For a four bit binary number when MSB is 1 for the first time (i.e the number is 1000_2) its unsigned decimal value is 8. However, since any leading 1 means a -ve number it is assumed to be -8 in decimal. From that point onwards, we can add the unsigned value represented by rest of the 3 lower bits to -8 and get the corresponding -ve number. For example 1011_2 is ($-8_{10} + 011_2 = -8_{10} + 3_{10} = 5_{10}$).

- 2's complement is computed by inverting every bit of a binary number and add 1 to it. For example 2's complement of 0011_2 (3_{10}) is ($1100_2 + 1_2 = 1101_2 = -3_{10}$).

- 2's complement of a -ve number will give it's +ve counter part, except for the lowest -ve number represented in the sytem. For example 2's complement of 1101_2 (-3_{10}) is ($0010_2 + 1_2 = 0011_2 = 3_{10}$). However, 2's complement of 1000_2 (-8_{10}) is ($0111_2 + 1_2 = 1000_2 = -8_{10}$). This boundary condition may pose some challenge to software engineering side.

- Since all the leading bits will be 1 in case of binary -ve number in 2's complement representation, it is sufficient to test the MSB for to determine the sign of the number. This also gives the ease of extending a number in 2's complement form. If it is required to sign extend a number from shorter size to longer size, it is sufficient to repeat the MSB of the shorter representation into rest of the higher bit positions in the longer representation. For example if 4-bit number 1101_2 has to be extended to 8-bit, the MSB (which is 1) needs to be repeated through rest of the higher order bits in the 8-bit representation. The result is 11111101_2

- Example of overflow discarding would be $(7-1)_{10} = (7 + (-1))_{10} = (0111 + 1111)_2 = \cancel{1}(0110)_2 = 6_{10}$



CS147 - Lecture 03

Kaushik Patra
(kaushik.patra@sjsu.edu)

12

- Topics
  - Digital Synthesis
  - Number representation

[Chapter 1 (1-2, 1-3) of Logic & Computer Design Fundamentals, 4th Edition, M. Morris Mano, Charles R. Kime]