## Lecture Summary

### Operations Representation—continued

3. Overflow: hardware produces mathematically incorrect result.
4. Overflow conditions: Observation
5. Overflow detection: when overflow condition, watch sign bit.
   - Addition: overflow if sign bit of result is different from operands.

   **Example 11.** Design a signed addition overflow detector.

   - Subtraction.

   **Exercise 16.** Design a signed subtraction overflow detector.

6. Overflow in unsigned numbers.
7. Hardware handling of overflow: signed instructions (add, addi, sub, slt, slti) check overflow, unsigned instructions (addu, addiu, subu, sltu, sltiu) ignore overflow.

### Number Representation: Real Numbers

1. Decimal real numbers: normalized scientific notation.
2. Normalized binary floating point (FP) numbers.

### Hardware Representation of Real Numbers

1. Parts of FP to store.
2. FP fields.
3. Limited precision storage:

#### IEEE 754

1. Field formats (coding): single & double precision numbers.
3. IEEE 754 is sign-magnitude representation different from 2’s complement, even though both indicate sign at MSB. (Note: magnitude ≡ absolute value).

   **Example 12.** Show the 5-bit sequence for –6 using 2’s complement and sign-magnitude representations. Assume 0 for positive, and 1 for negative.

   - Sign-magnitude: 10110 (magnitude of –6 stored directly in lower 4 bits).
   - 2’s complement: 11010 (magnitude is not stored directly).

### Reading List

1. 3.3-3.6

### Learning Outcomes

- Define integer overflow.
- State the reason behind integer overflow.
- Explain how signed overflow happens.
- Identify conditions for signed overflow in both addition and subtraction.
Outline steps for detecting signed overflow for both addition and subtraction.
State why overflow is not handled in unsigned arithmetic.
Describe the effects of the MIPS instructions (in terms of overflow behavior): $\text{slt}$/$\text{sltu}$, $\text{add}$/$\text{addu}$, $\text{sub}$/$\text{subu}$, $\text{addi}$/$\text{addiu}$.
Explain the main advantage of using scientific notation to represent real numbers.

Keywords
[Integer] Overflow, Floating-point (FP), normalized [FP], [scientific] notation.
Exercise 17

Use Exercise 17 and your textbooks from previous courses to review:

- Determine the meaning of a digit in a positional number system given its position, the symbol in that position, and the number of symbols in the system.
- Interpret the numerical value of a binary sequence. (Stated another way: determine the natural number given its representation in the binary number system.)
- Show how bits are allocated to represent unsigned integers given a store size.
- Show how bits are allocated to represent 2's complement signed integers given a store size.
- Determine the largest and smallest unsigned integers which can be represented (coded) given a store size.
- Determine the largest and smallest 2's complement signed integers which can be represented given a store size.
- Determine the binary code given a natural number (=unsigned integer) and a store size.
- Determine the 2's complement binary code given a signed integer and a store size.
- Perform limited precision addition of 2 binary words with carry.
- Perform limited precision 2's complement subtraction of 2 binary words.

1. Convert to natural decimal.
   (a) 110112
   (b) 1001002
   (c) 1011112
   (d) 111010012

2. Invert (negate) the 2's complement code.
   (a) 100100
   (b) 11011
   (c) 102
   (d) 6410

3. Convert 2's complement to decimal.
   (a) 100100 (6-bit)
   (b) 01111111 (8-bit)
   (c) 11101001 (8-bit)
   (d) 0x8020 (16-bit).

4. Represent in 8-bit.
   (a) 16
   (b) −129
   (c) −44
   (d) 350

5. Find the minimum word to perform the arithmetic operation 63+18.

6. ★Write programs to solve exercises 1–4.

For 6&7, represent decimal in hardware first.

7. Perform unsigned operation in 8-bit register.
   (a) 255 + 25
   (b) 5 + 134
   (c) 238 – 23
   (d) 23 – 238

8. Perform signed operation in 8-bit register.
   (a) 45 + 54
   (b) 96 – 32
   (c) (−1) – 63
   (d) 120 + 96

9. Perform MIPS arithmetic operation. Convert to decimal to check your answers.
   (a) 189dfa70 + 7f00d456
   (b) 0x76789012 – 19d7825a
   (c) 0x80000090 – 6000120c

10. Write the 8-bit code range for the following:
    (a) Signed integers (2's comp coded).
    (b) Unsigned integers.
    (c) Bit representations of first and last 3 positive integers.
    (d) Bit representations of first and last 3 negative integers.
Answers

1. Natural numbers \{0,1,2,\ldots\} are stored as unsigned integers (a) 27 (b) 36 (c) 47 (d) 233
2. (a) 011100 (b) 00101 (c) 10 (d) 1000000 (64\text{H}). What is the rule for 2's complement if number is power of 2?
3. (a) –28 (b) 127 (c) –23 (d) –215+32
4. [a] 00010000 (b) out of 8-bit 2's complement range (c) 11010100 (d) can't be represented in 8-bit 2's complement
5. Although 63 is a 6-bit binary (111111), the operation can not be performed in 6-bit 2's complement. Moreover, the sum can't be stored in 7-bit 2's complement. The minimum is 8-bit precision (store).
6. C function for exercise 1 (add test main(), compile, run, trace by hand or with debugger).

```c
/* Convert unsigned binary to decimal
   input: unsigned binary number as C string (null-terminated char array), example, "1000" 
#include: stdio.h, string.h 
return: decimal as long (%ld in printf) 
*/
long ubin2dec(char *binary)
{
  int  bit;
  const int base = 2;
  long  sum = 0, weight = 1;
  bit = strlen(binary);
  while ( bit-- )  /*** bit (loop control): n to 1 ***/
  { /*** bit (index): n-1 to 0 ***/
    sum += (binary[bit] - '0') * weight;
    weight *= base;
  }  /*** bit exit value: -1 ***/
  return sum;
}
```
7. (a) 00011000 (b) 10001011 (c) 11010111 (d) 00101001
8. (a) 01100011 (no overflow) (b) 01000000 (no overflow) (c) 11000000 (no overflow) (d) 11011000 (overflow)
9. MIPS uses 2's complement for arithmetic. (a) 0x979ece66 (overflow) (b) 0x5ca10db8 (no overflow) (c) 0x1fffe84 (overflow)
10. (a) 256 signed integers: -128, -127, ... , -1, 0, 1, 2, ..., 127 (b) 256 unsigned: 0, 1, 2, ..., 255 (c) 0000 0000, 0000 0001, 0000 0010, 0111 1101, 0111 1110, 0111 1111 (or 0x00, 0x01, 0x02, 0x7d, 0x7e, 0x7f) (d) 2's complement: 1111 1111, 1111 1110, 1111 1101, 1000 0010, 1000 0001, 1000 0000, (or 0xff, 0xfe, 0xfd, 0x82, 0x81, 0x80)