KING'S College LONDON

Hardware Design

Tutorial 1

Instructor: Dr. Haiyu Mao
TA: Zihao Pu

23.01.2026

Recap: Signed number

- **e.g. 3-bit numbers. In computer design, we usually use 2's complement design because the circuit is simple.**

Num	Sign-mag	1's comp.	2's comp.
+3	011	011	011
+2	010	010	010
+1	001	001	001
+0	000	000	000
-0	100	111	-
-1	101	110	111
-2	110	101	110
-3	111	100	101
-4	-	-	100

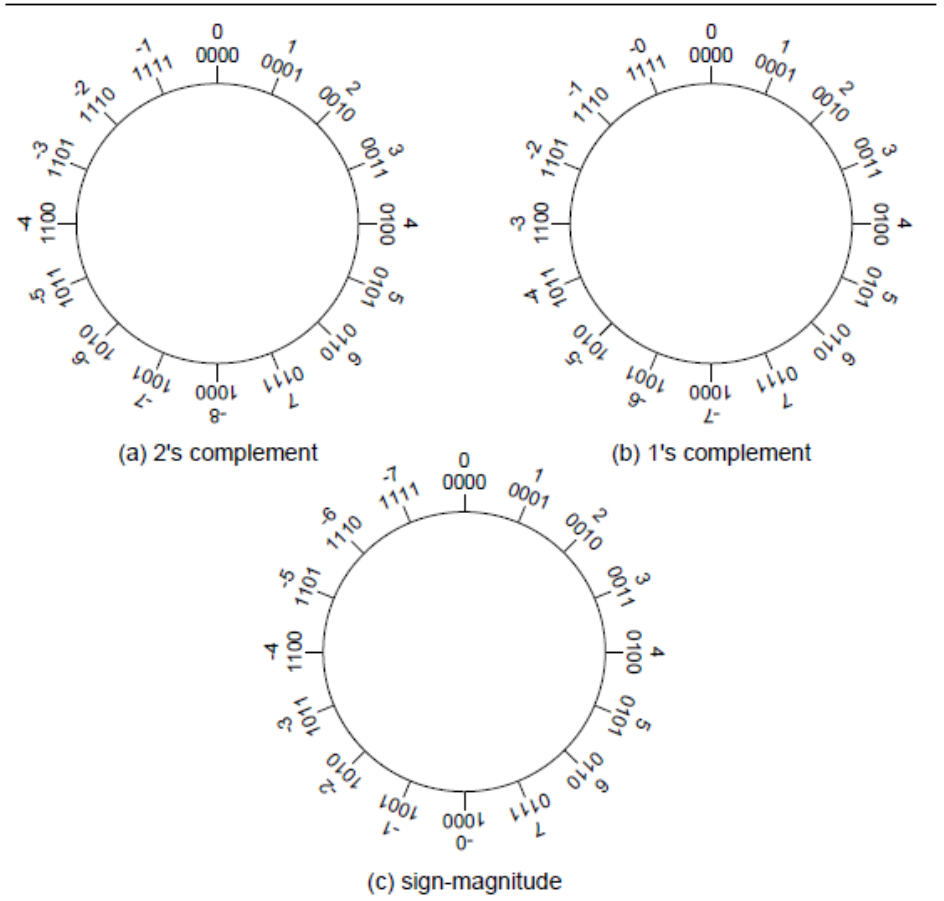
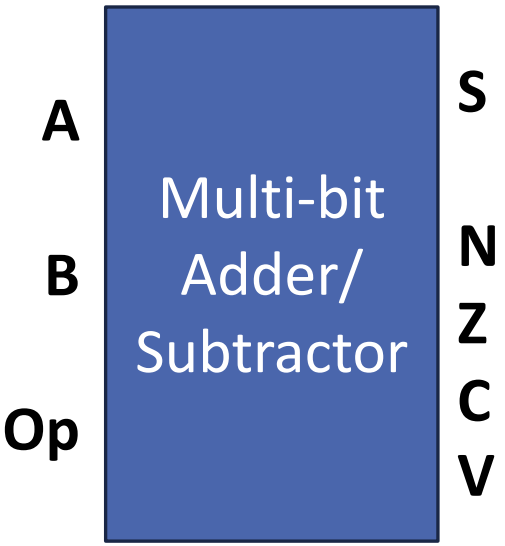


Figure 10.9: Number wheel showing three encodings of negative numbers (a) 2's complement, (b) 1's complement, and (c) sign-magnitude.

Digital Design – A System Approach

Recap: ALU Status Flags: **N,Z,C,V**

Sign	Name	Usage	Source
N	Negative	1 if result is negative, 0 if not	S_n
Z	Zero	1 if result is 0, 0 if not	$\lvert \boldsymbol{S}$
C	Carry	1 if there is a carry-out; 0 if not	C_n
V	oVerflow	1 if there is overflow; 0 if not	$C_n \oplus C_{n+1}$



Recap: ALU: oVerflow circuit deign

- Overflow definition
  - For adding: if both inputs are positive, but result is negative; or if both inputs are negative, but result is positive.
  - For subtracting: if A is positive, B is negative, but result is negative; or if A is negative, B is positive, but result is positive.
- For unsigned addition:
- $C = 0$: No overflow
- $C = 1$: Overflow

Adding example:
For 8-bit signed number, available value is between -128 to 127. Try to calculate 70+80, we expect 150, but the result become negative: -106

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

- Defne V(Overflow): for **signed** addition/subtraction:
- $V = 0$: No overflow
- $V = 1$: Overflow

Recap: ALU: Idea of oVerflow

as	bs	cis	qs	cos	ovf	comment
0	0	0	0	0	0	Both inputs positive, both carrys 0, no overflow
0	0	1	1	0	1	Both inputs positive, carry in 1, overflow
0	1	0	1	0	0	Input signs different, carry in 0, no overflow
0	1	1	0	1	0	Input signs different, carry in 1, no overflow
1	1	0	0	1	1	Both inputs negative, carry in 0, overflow
1	1	1	1	1	0	Both inputs negative, carry in 1, no overflow

Table 10.3: Cases for inputs and carry into sign bit of adder to detect overflow. Columns show sign bit of a and b (as and bs) carry into and out of sign bit (cis and cos) and output of sign bit (qs). Overflow only occurs if the carrys into and out of the sign bit are different.

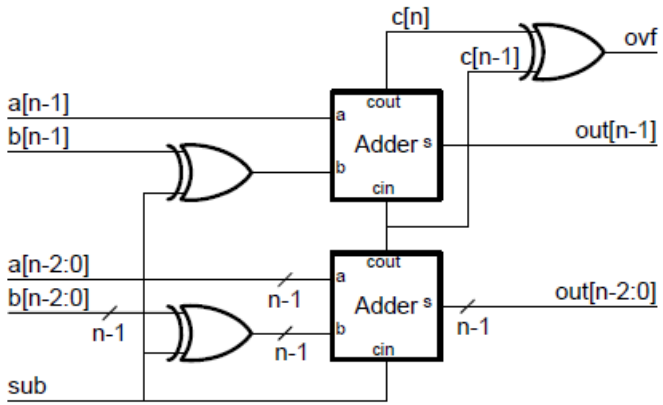
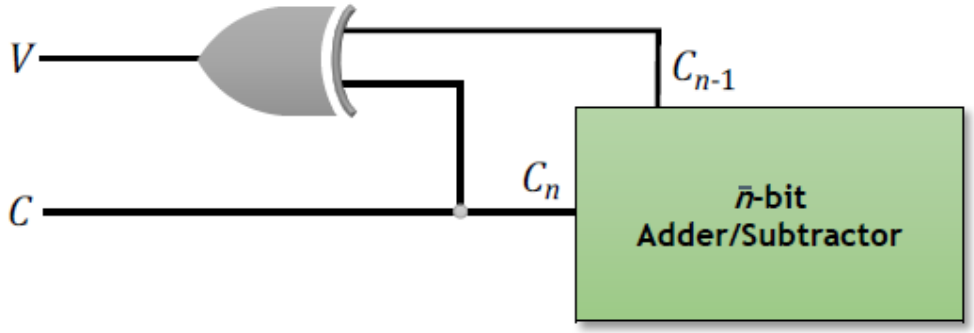


Figure 10.12: A 2's complement add/subtract unit with overflow detection based on carry in and out of the last bit.

Circuit design: you can check based on definition, or use the following simplified circuit.



Conventions of bases

- Prefix of bases in math:
  - Binary (base 2 number system): 0b
  - Octal (base 8 number system): 0o
  - Decimal (base 10 number system): 0d, or no prefix
  - Hexadecimal (base 16 number system): 0x

- Prefix of based in Verilog:
  - Binary (base 2 number system): ‘b e.g. 8’b10101010
  - Octal (base 8 number system): ‘o e.g. 8’o158
  - Decimal (base 10 number system): ‘d e.g. 8’d128
  - Hexadecimal (base 16 number system): ‘h e.g. 8’hab

Concept Question: von Neumann model

- State the **two key properties** of the von Neumann model.

  Answer:

  Stored program – instructions and data stored in the same memory

  Sequential execution – one instruction fetched, executed, completed at a time, controlled by PC

- List the five major components of the von Neumann model and briefly describe their functions.

Memory – stores both program instructions and data.
Processing Unit (ALU) – performs arithmetic and logic operations.
Control Unit – controls order of execution via PC & IR.
Input – provides data to computer from external sources.
Output – sends results out (monitor, printer, disk).

Question: Instruction Processing

- The following table shows a 16-bit register file. The following sub questions are independent.
- Q1: What is the value of R0 if the following code executed:
  - add R0, R1, R2
  - Answer: 0x5000
  - Explain: This instruction means R0=R1+R2, therefore the result is 0x8000+0xD000 = 0x15000

```
  0x   D 0 0 0
+ 0x   8 0 0 0
_______________
  0x 1 5 0 0 0
```

  - However, our register can only take 16 bits, the result in R0 is **0x5000**.
- Q2: Is the result the desired result? Or has there been an overflow?
  - Answer: **No. Overflow exist**

    Explanation: Consider all the numbers are signed 2's complement numbers, 0x8000 = -32768, and 0xD000 = -12288. Adding two negative number should result in negative. However, the final value in register 0x5000 = 20480, which is positive number. This case is called OVERFLOW.

Reg	Data
R7	0x0000
R6	0x0000
R5	0x1111
R4	0x1234
R3	0xABCD
R2	0x8000
R1	0xD000
R0	0x0000

Question: Instruction Processing

- The following table shows a 16-bit register file. The following sub questions are independent.
- Q3: What is the value of R3 if the following code executed:
  - sub R3, R1, R2
  - Answer: 0x5000
  - Explain: The instruction means R3 = R1 – R2, which is 0xD0000-0x8000 = 0x5000
  - A hand calculation:

```
  0x   D 0 0 0
- 0x   8 0 0 0
_______________
  0x   5 0 0 0
```

- Q4: Is the result the desired result? Or has there been an overflow?
  - Yes. The result is desired result.

Reg	Data
R7	0x0000
R6	0x0000
R5	0x1111
R4	0x1234
R3	0xABCD
R2	0x8000
R1	0xD000
R0	0x0000

Question: Instruction Processing

- The following table shows a 16-bit register file. The following sub questions are independent.
- Q5: What is the value of R0, after following code executed:
  - add R0, R1, R2
  - add R0, R0, R1

Answer: 0xD000

Explain:

R0 = R1 + R2 = 0xD000+0x8000 = 0x5000

Since R0 is updated to 0x5000, when you execute next instruction consecutively:

R0 = R0 + R1 = 0x5000 + 0x8000 = 0xD000

Reg	Data
R7	0x0000
R6	0x0000
R5	0x1111
R4	0x1234
R3	0xABCD
R2	0x8000
R1	0xD000
R0	0x0000

Question: Memory Space

- 1. Given LC3 architecture: 16-bit word length, word addressable, 16-bit addresses. What is the maximum memory space?
  - Answer: 128KB
  - Explanation: Memory space = (# of addresses) * (bytes per address). In this question, (# of addresses) is 2^16, and (byte per address) is 2 since this architecture is word addressable and each word is 2 bytes.
- 2. Given MIPS architecture: 32-bit word length, byte addressable, 32-bit addresses .What is the maximum memory space?
  - Answer: 4GB
  - Explanation: Memory space = (# of addresses) * (bytes per address). In this question, (# of addresses) is 2^16, and (byte per address) is 1 since this architecture is byte addressable, meaning each address refers to a single byte.
- 3. Given x86-64 architecture: 64-bit word length, byte addressable, 48-bit addresses. What is the maximum memory space?
  - Answer: 256TB
  - Explanation: Memory space = (# of addresses) * (bytes per address). In this question, (# of addresses) is 2^16, and (byte per address) is 1 since this architecture is byte addressable, meaning each address refers to a single byte.

Question: Big-endian and little-endian

- Show how the value 0xabcdef12 would be arranged in the memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

Address	0x03	0x02	0x01	0x00
Big-endian	0x12	EF	CD	AB
Little-endian	0xAB	CD	EF	12

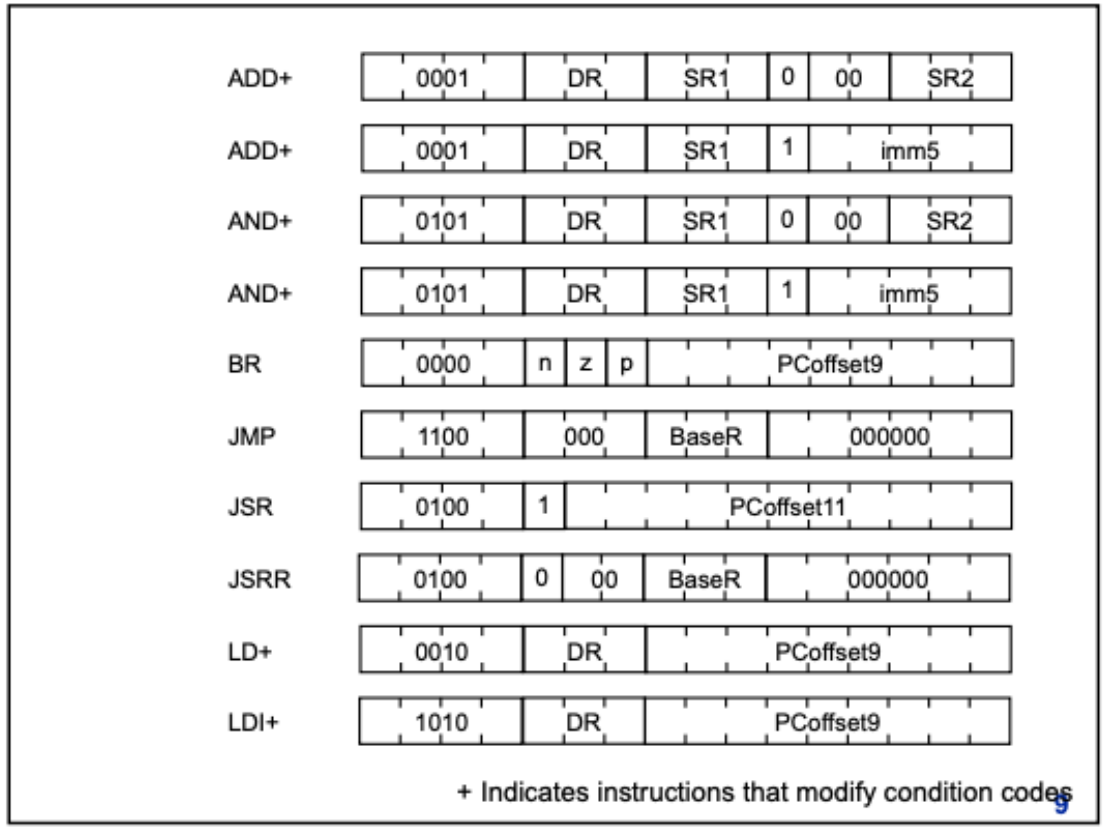
- Explanation:
- Big-endian means MSB on lower address
- Little-endian means MSB on higher address

Side note: take number 123 in decimal as an example:
123 =
1 x 100 +
2 x 10 +
3 x 1
The digit 1 is most significant digit, and 3 is least significant digit, obviously.

Question: Instruction Encoding

- Given part of the LC3 ISA, encode the following instructions into binary format
  - ADD R0, R1, R2
  - ADD R5, R0, #7
  - JMP R5

- Answer:
- 0b 0001 000 001 0 00 010
- 0b 0001 101 000 1 00 111
- 0b 1100 000 101 0 00 000



Question: Instruction Decoding

- Given part of the LC3 ISA, decode the following instructions into assembly code, and explain what does each line do.
  - 0001 010 001 0 00 011
  - 0101 011 100 1 00101
  - 1001 001 010 111111

Answer:

0001 010 001 0 00 011 -> ADD R2, R1, R3 -> R2 = R1+R3=0x0001+0x1000=0x0101

0101 011 100 1 00101 -> ADD R3, R4, #5 -> R3 = R4&0x5=0x0010 & 0x0005=0x0000

1001 001 010 111111 -> NOT R1, R2 -> R1 = ~R2 = ~0x0101 = 0xFEFE

Regnum	Init Value	Final Value
R0	0x1234	0x1234
R1	0x0001	0xFEFE
R2	0x0010	0x0101
R3	0x0100	0x0000
R4	0x0010	0x0010
R5	0x0000	0x0000
R6	0x0000	0x0000
R7	0xABCD	0xABCD