CSE2312: Computer Organization and Assembly Language Programming

Slide 1: Number Conversion (Integer)

Decimal/Base-10

- 10 digits: 0~9
- The weight of digit n from the right is 10^n
  - e.g., the ‘3’ in 5326 is $3*10^2$

Binary/Base-2 (Unsigned)

- 2 digits: 0 and 1
- The weight is exponent of 2
  - Weight of digit n from the right is 2^n
  - E.g., 10101010 (base-2)

1	0	1	0	1	0	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

MSb (the most significant bit)

LSb (the least significant bit)

Binary/Base-2 (Unsigned)

- 2 digits: 0 and 1
- The weight is exponent of 2
    - Weight of digit n from the right is 2^n
    - E.g., 10101010 (base-2)
    - 10101010 (base-2), 0b10101010, 10101010_2

Conversion: Binary to Decimal (Unsigned)

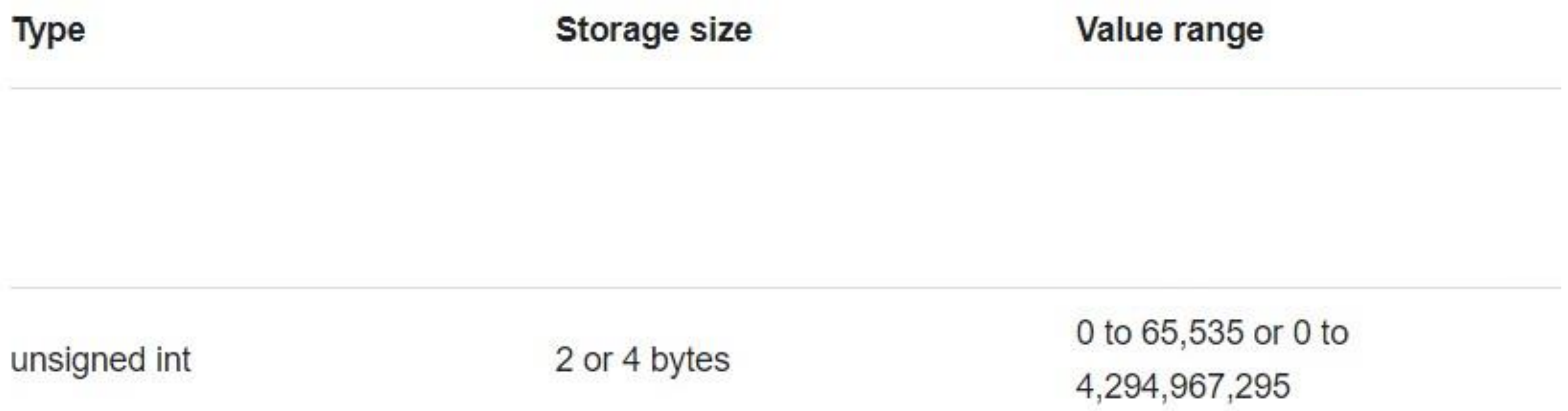
- Step 1: List the weights of all the digits
- Step 2: Multiple digits by their corresponding weights
- Step 3: Calculate the sum of all products
- Practice: Convert 10101010 (base-2) to decimal

1	0	1	0	1	0	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$\begin{aligned} & 10101010(base-2) \\ = & 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ = & 128 + 32 + 8 + 2 \\ = & 170 \end{aligned}$$

Ambiguity of type int in C language

- Type int is store in binary
- C language of different versions use different number of bit for int



Type	Storage size	Value range
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295

Ambiguity of type int in C language

- Type int is store in binary
- C language of different versions use different number of bit for int
- To avoid ambiguity, this course will use type int with precision (C99)
  - uint8_t/uint16_t/uint32_t/uint64_t: unsigned int of 8 bits/16 bits/32 bits/64 bits
  - int8_t/int16_t/int32_t/int64_t: signed int of 8 bits/16 bits/32 bits/64 bits

Range of uintN_t (Unsigned)

- Range of uint8_t: 0 (0b00000000) ~255 (0b11111111)
- Range of uint16_t: 0 (0b0000000000000000) ~65,535 (0b1111111111111111)
- Range of uintN_t: 0 (all 0s) ~ 2^N-1 (all 1s)

Conversion: Decimal to Binary (Unsigned)

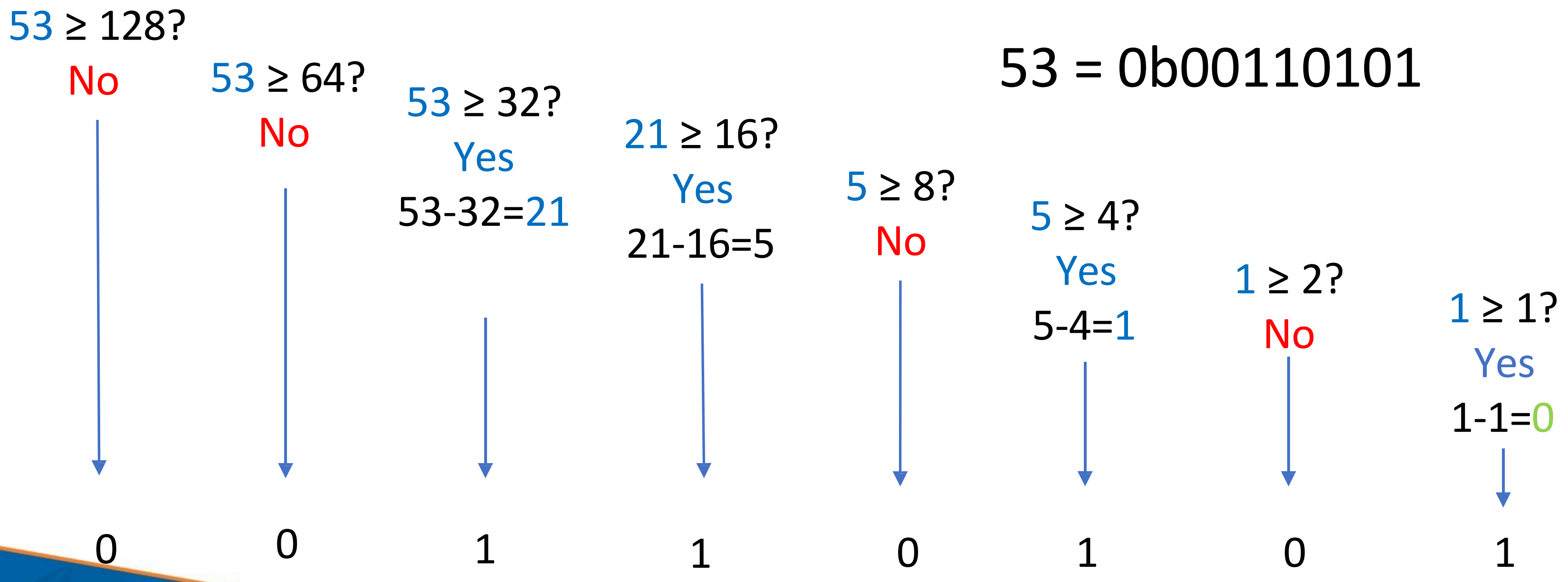
- Step 0: Check whether the decimal number is within the range
- Step 1: List the weights of bits at all the positions
- Step 2: Compare the decimal number with weights along the direction from MSb to LSb. If the number is larger than or equal to the weight, subtract the weight from the number, put ‘1’ as the binary digit in this position, and continue with the difference. Otherwise, put ‘0’ and move on to the next weight.
- Practice: Convert 53 to uint8_t

Conversion: Decimal to Binary

- 53 is within the range of uint8_t (0~255)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

53 = 0b00110101



Conversion: Decimal to Binary

- Practice: Convert 1000 to uint8_t
- Answer: 1000 is larger than 255, it’s impossible

Hex/Base-16

- 16 digits: 0~9, A (10), B (11), C(12), D(13), E(14), F(15)
- The weight is exponent of 16
  - Weight of digit n from the right is 16^n
  - E.g., the ‘A’ in ABCD (Base-16) stands for $10*16^3$
- ABCD (Base-16), 0xABCD, $ABCD_{16}$

Conversion: Hex to Decimal

- Step 1: List the weights of all the digits
- Step 2: Multiple digits by their corresponding weights
- Step 3: Calculate the sum of all products
- Practice: Convert ABCD (base-16) to decimal

	0x	A	B	C	D
(weight)		16^3	16^2	16^1	16^0

$$
\begin{aligned}
&0xABCD \\
&= A \times 4096 + B \times 256 + C \times 16 + D \times 1 \\
&= 10 \times 4096 + 11 \times 256 + 12 \times 16 + 13 \times 1 \\
&= 43981
\end{aligned}
$$

Conversion: Hex to Binary

Solution 1

- Step 1: Convert Hex to Decimal
- Step 2: Convert Decimal to Binary

Conversion: Hex to Binary

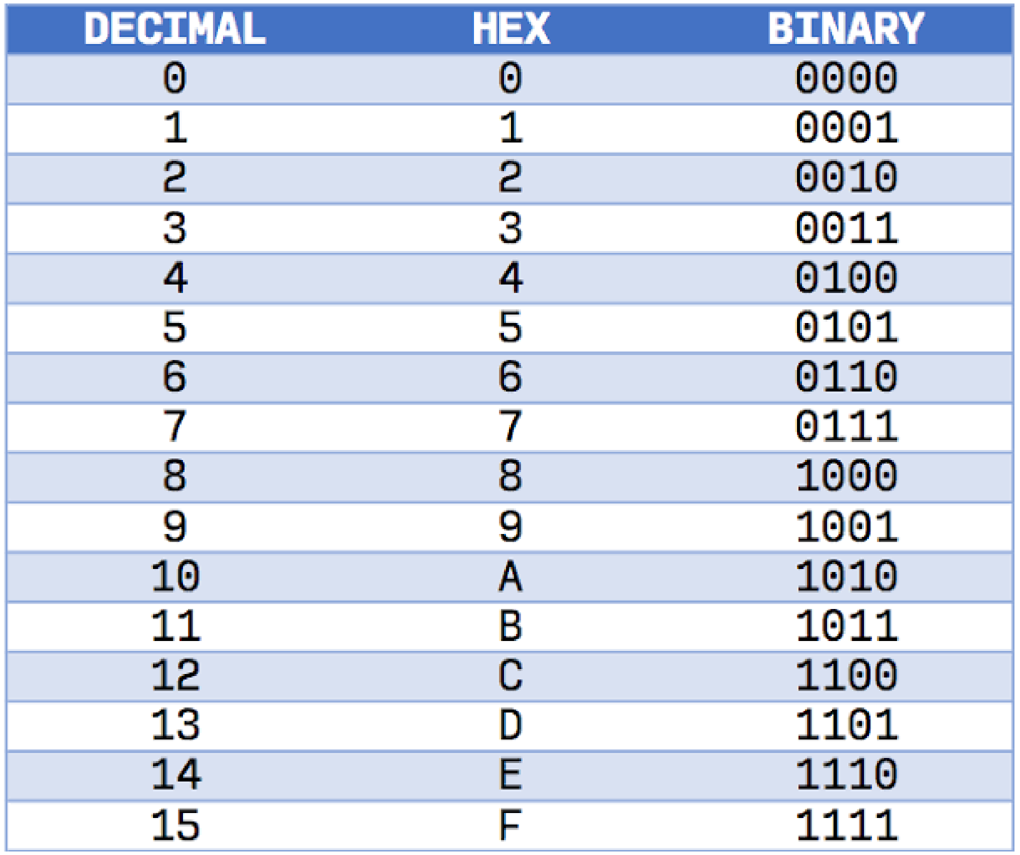
- 0x0 = 0 = 0b0000
- 0x8 = 8 = 0b1000
- 0xF = 15 = 0b1111
- 0x8F = 143 = 0b10001111
- 0x8F8F = 36751 = 0b1000111110001111

Conversion: Hex to Binary

- 0x0 = 0 = 0b0000
- 0x8 = 8 = 0b1000
- 0xF = 15 = 0b1111
- 0x8F = 143 = 0b 1000 1111
- 0x8F8F = 36751 = 0b 1000 1111 1000 1111

Conversion: Hex to Binary

- Convert each hex digit to four binary digit and put them together
- Practice: Convert ABCD to binary
  - 0x ABCD (43981)
  - = 0b 1010 1011 1100 1101 (43981)

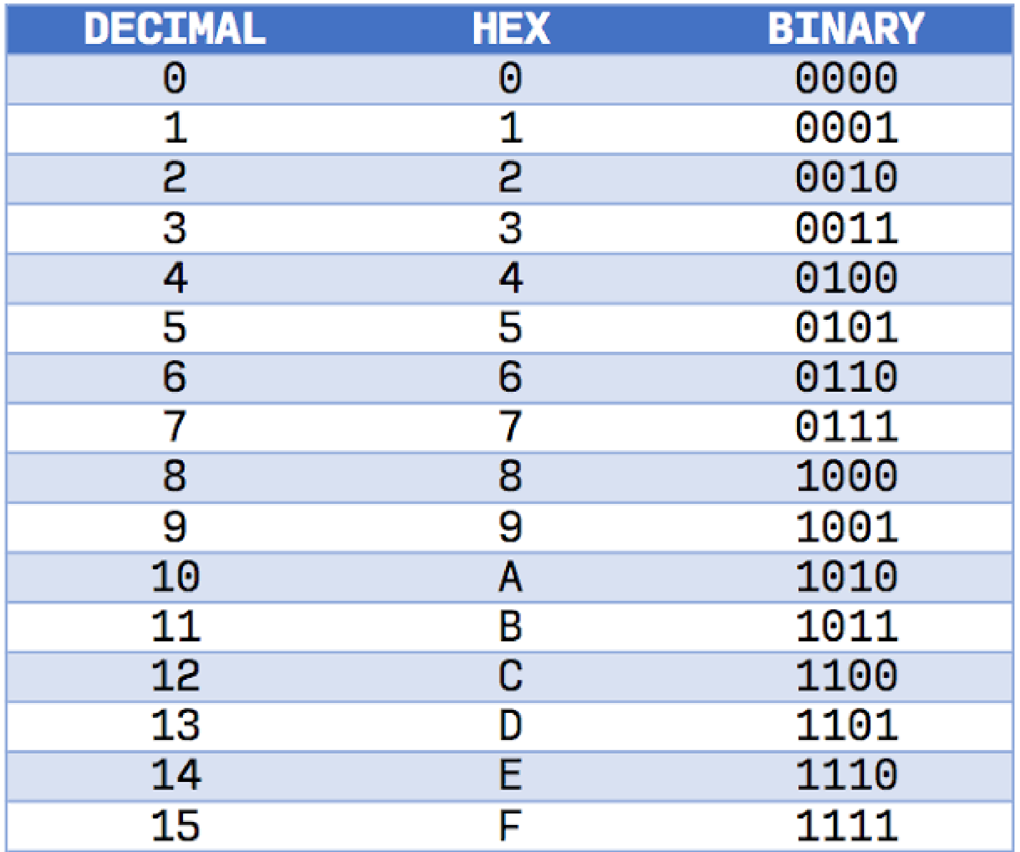


DECIMAL	HEX	BINARY
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversion: Binary to Hex

- Step 1: Divide the binary digits to segments of four digits
- Step 2: Convert every four binary digits to a hex digits, and put them together
- Practice: convert 0b1000010010101001 to hex
  - 0b 1000 0100 1010 1001

  = 0x 8 4 A 9



DECIMAL	HEX	BINARY
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversion: Decimal to Hex

- Step 1: Convert Decimal to Binary
- Step 2: Convert Binary to Hex

Homework Examples

- Convert the following numbers between bases:

  10111011 (base-2) = ____________ (base-10)

  10111011 (base-2) = ____________ (base-16)

- What is the range of the following C99 variable types (assuming the processor uses two's compliment arithmetic for signed number representation)?

  uint8_t ______________ to _______________

  uint16_t ______________ to _______________

Binary/Base-2 (signed)

- int8_t, int16_t, int32_t, int64_t
- **2's complement coding: The weight of the MSb is negative**
  - E.g., 10101010 (base-2)

1	0	1	0	1	0	1	0
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

MSb (the most significant bit)

LSb (the least significant bit)

Conversion: Binary to Decimal (signed)

- Step 1: List weights of all digits (MSb is negative)
- Step 2: Multiply each digit by its weight
- Step 3: Calculate the sum of all products
- Practice: Convert 10101010 (int8_t) to decimal

$$\begin{aligned}&10101010(int8\_t)\\&= 1 \times (-2^7) + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0\\&= -128 + 32 + 8 + 2\\&= -86\end{aligned}$$

Why both Unsigned and Signed?

Resource consideration example:

- Range of int8_t: 0b10000000 (-128) ~ 0b01111111 (127)
- Range of int16_t: -32768 ~ 32767
- Range of intN_t: $-2^{N-1} \sim 2^{N-1} - 1$

Performance consideration example:

- Operations on unsigned integers can be optimized differently by the hardware.

Reliability consideration example:

- Using unsigned integers where only non-negative values make sense can help catch logical errors.

Sign Bit (MSb)

- int8_t numbers and their corresponding decimal

0b0 1 1 1 1 1 1 1	127
0b0 1 1 1 1 1 1 0	126
… …	… …
0b0 0 0 0 0 0 0 1	1
0b0 0 0 0 0 0 0 0	0
0b1 1 1 1 1 1 1 1	-1
0b1 1 1 1 1 1 1 0	-2
… …	… …
0b1 0 0 0 0 0 0 1	-127
0b1 0 0 0 0 0 0 0	-128

MSB is **Sign Bit**

Sign Bit is 0 for non-negative values
Sign Bit is 1 for negative values

Conversion: Decimal to Binary (signed)

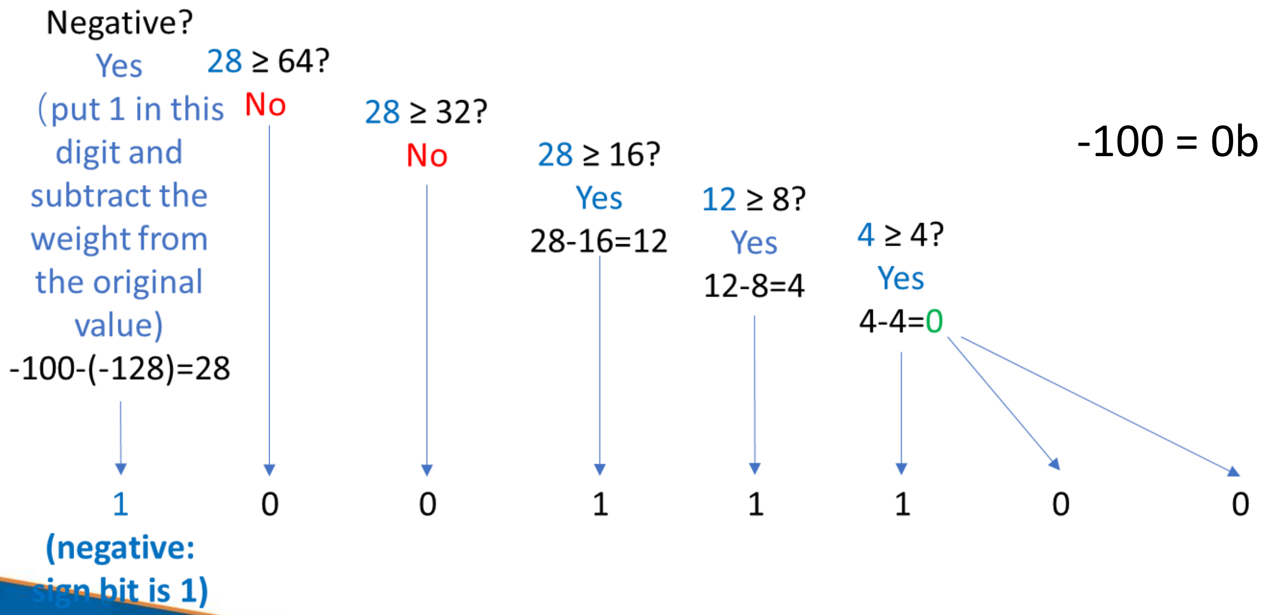
- Step 0: Check whether the decimal value is within the range of the type
- Step 1: List the weights of bits at all the positions (MSb is negative)
- Step 2: If the decimal number is negative, put ‘1’ at the MSb, subtract the weight of the MSb (negative) from the decimal number, and move forward to step 3. Otherwise, put ‘0’ at the MSb, and move forward to step 3.
- Step 3: Compare the decimal value yielded from step 2 with the weights of the remaining digits along the direction from the second most significant bit to the LSb. If the decimal value is larger than the weight, put ‘1’ at this digit, subtract the weight from the decimal value, and move on with the difference. Otherwise, move on to the next digit.
- Practice: Convert -100 to int8_t

Conversion: Decimal to Binary (signed)

- Practice: Convert -100 to int8_t

sign bit is 0 for non-negative values
sign bit is 1 for negative values

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128	64	32	16	8	4	2	1

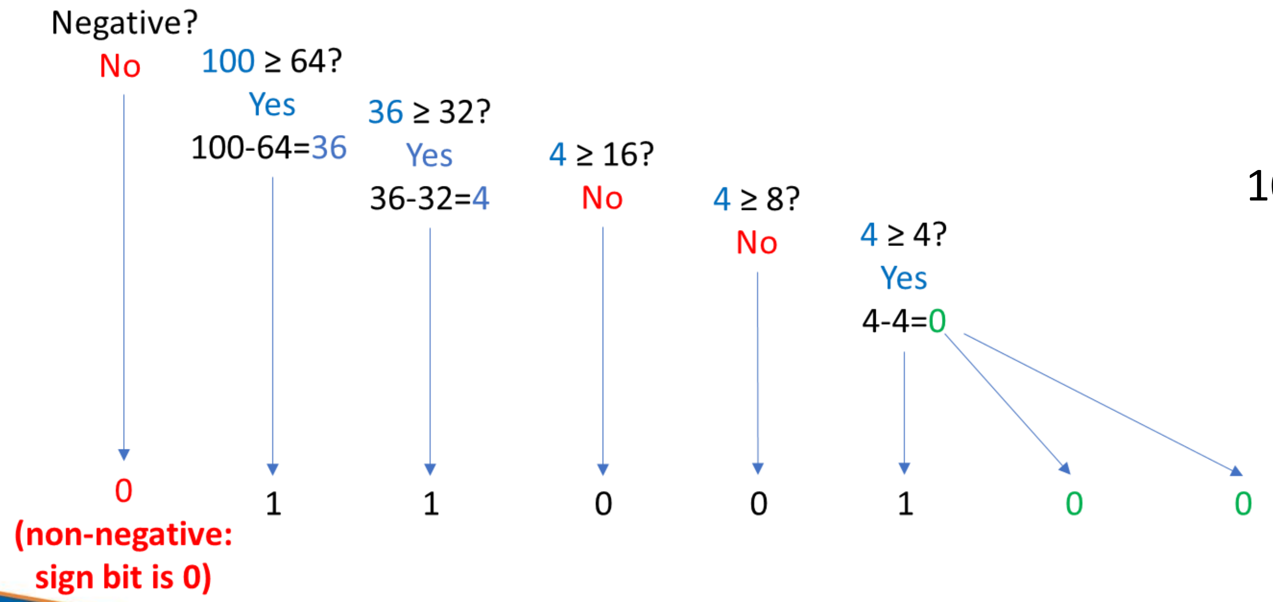


-100 = 0b10011100

Conversion: Decimal to Binary (signed)

- Practice: Convert 100 to int8_t

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128	64	32	16	8	4	2	1



100 = 0b01100100

Benefits of 2’s Complement Coding

- Given a binary number, the computer hardware does not need to know whether it’s signed or unsigned
- With 2’s complement coding, computer can calculate correctly anyway.

0b1 0 0 0 0 0 0 1 + 0b0 0 0 0 0 0 0 1

uint8_t
129 + 1 = 130

uint8_t
130

0b1 0 0 0 0 0 1 0

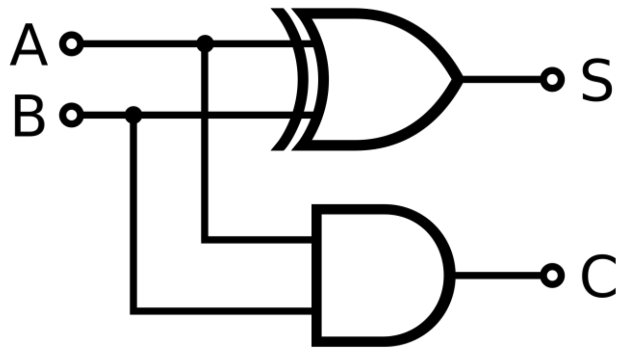
Int8_t
-127+ 1 = -126

uint8_t
-126

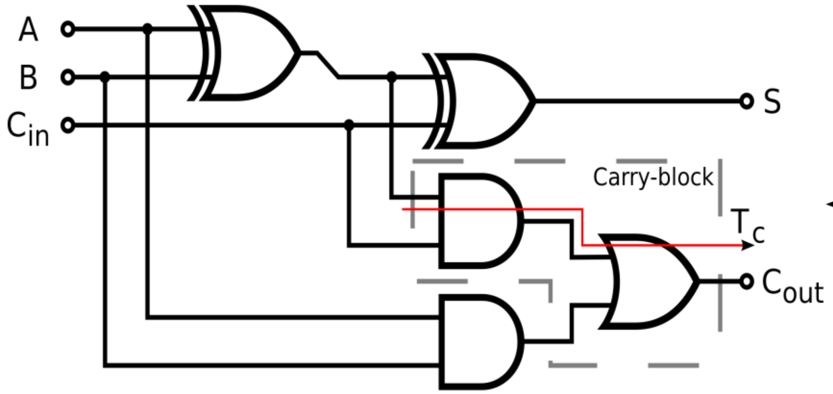
Benefits of 2’s Complement Coding

- 2’s Complement Coding simplifies ALU significantly.
- Example of a one-bit adder:

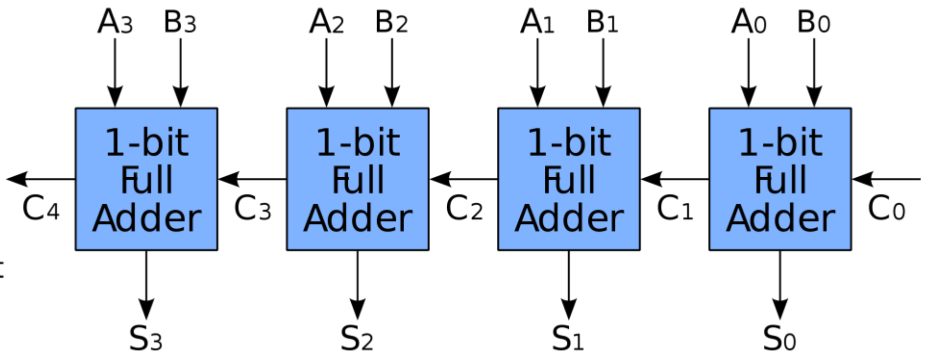
Half Adder



Full Adder



Four bits



Homework Examples

- Range of the following C99 variable types assuming the processor uses two's compliment arithmetic?

  int8_t ________________ to _________________

  int16_t ________________ to _________________

- Binary representation of the C99 variables given below?

  int8_t x = -40:

  int8_t x = -103: