National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Lecture 2-1: Sign-Extension

CS10014 Computer Organization

Department of Computer Science
Tsung Tai Yeh
Thursday: 1:20 pm– 3:10 pm
Classroom: EC-022

Acknowledgements and Disclaimer

- Slides were developed in the reference with
  - CS 61C at UC Berkeley
    - https://inst.eecs.berkeley.edu/~cs61c/sp23/
  - CS 252 at UC Berkeley
    - https://people.eecs.berkeley.edu/~culler/courses/cs252-s05/
  - CSCE 513 at University of South Carolina
    - https://passlab.github.io/CSCE513/

Sign Magnitude Representation

- The Most Significant bit of the number if a sign bit
- The remaining bit represents the magnitude of the number in a binary form

MSB **Magnitude**

0 0 1 0 0 0 1 0

- Example: 8-bit sign-magnitude form

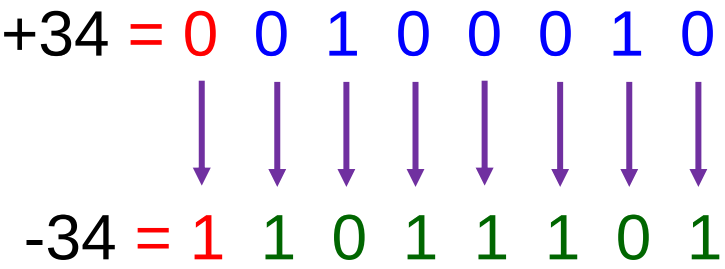
+34 = 0 0 1 0 0 0 1 0

-34 = 1 0 1 0 0 0 1 0

Using n-bits, the range of numbers is from $-(2^{n-1})$ to $(2^{n-1} - 1)$

1’s Complement Representation

- The representation of the negative number is different from the positive number representation
- Example: The represent -34 in 1’s complement form



+34 = 0 0 1 0 0 0 1 0

-34 = 1 1 0 1 1 1 0 1

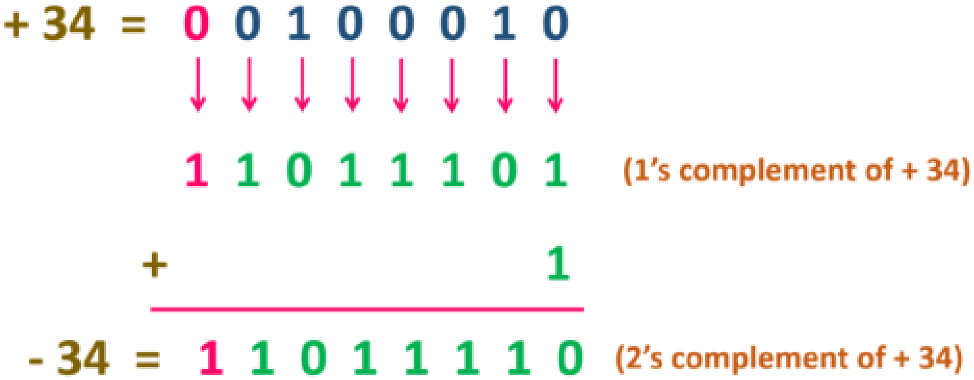
Invert all 1s in that number by 0s and 0s by 1s

2’s Complement Representation

- The representation of the positive number as the 1’s complement form
- Translate negative number from 1’s complement to 2’s complement form
    - Write the number corresponding to +34
    - Find 1’s complement of +34
    - Add 1 to the 1’s complement number

2’s Complement Representation

- Translate negative number from 1’s complement to 2’s complement form
  - Write the number corresponding to +34
  - Find 1’s complement of +34
  - Add 1 to the 1’s complement number



+ 34 = 0 0 1 0 0 0 1 0

1 1 0 1 1 1 0 1 (1’s complement of + 34)

+ 1

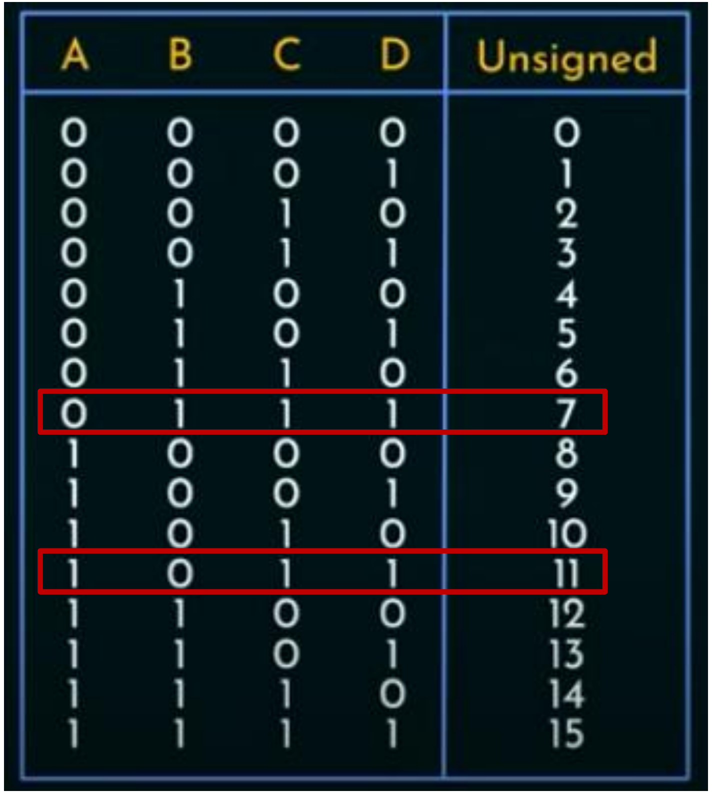
- 34 = 1 1 0 1 1 1 1 0 (2’s complement of + 34)

Understanding of overflow

- Carry indicates overflow

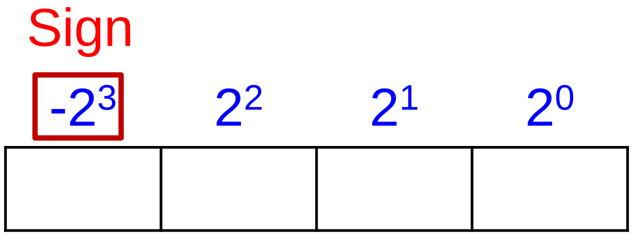
$$\begin{array}{rcccc l}
 & 1 & 0 & 1 & 1 & 7(\text{DEC}) \\
+ & 0 & 1 & 1 & 1 & 11(\text{DEC}) \\
\hline
\boxed{1} & 0 & 0 & 1 & 0 & 19(\text{DEC})
\end{array}$$

Overflow -> 19 is out of the range of the 4-bit value representation (0-15)



A	B	C	D	Unsigned
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Overflow in Signed numbers (2’s Complement)

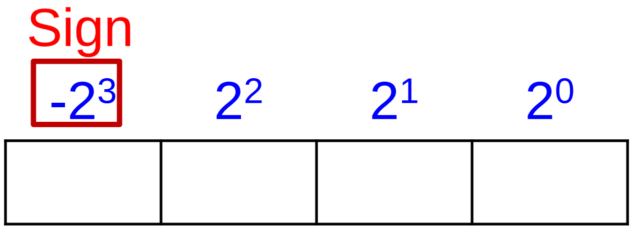


The range of 4-bits signed number
-2^{n-1} <-> $(2^{n-1} - 1)$ ==> -8 <-> 7

$$\begin{array}{rrrrrl} & 1 & 0 & 0 & 1 & -7(\text{DEC}) \\ + & 1 & 1 & 0 & 1 & -3(\text{DEC}) \\ \hline \boxed{1} & 0 & 1 & 1 & 0 & -10(\text{DEC}) \end{array}$$

Overflow !

Overflow in Signed numbers (2’s Complement)



The range of 4-bits signed number
-2^{n-1} <-> $(2^{n-1} - 1)$ ==> -8 <-> 7

$$\begin{array}{rccccl} & 0 & 1 & 1 & 1 & \quad 7(\text{DEC}) \\ + & 0 & 0 & 0 & 1 & \quad 1(\text{DEC}) \\ \hline & \boxed{1} & 0 & 0 & 0 & \quad -8(\text{DEC}) \end{array}$$

Overflow !

Overflow in Signed numbers (2’s Complement)

-2^3	2^2	2^1	2^0

The range of 4-bits signed number
-2^{n-1} <-> $(2^{n-1} - 1)$ ==> -8 <-> 7

$$
\begin{array}{rccccl}
 & 0 & 1 & 1 & 1 & \quad 7(\text{DEC}) \\
+ & 0 & 0 & 0 & 1 & \quad 1(\text{DEC}) \\
\hline
 & \boxed{1} & 0 & 0 & 0 & \quad -8(\text{DEC})
\end{array}
$$

Overflow ! How to fix this problem?

Overflow in Signed numbers (2’s Complement)

Sign

-2^4	2^3	2^2	2^1	2^0

The range of 5-bits signed number

-2^{n-1} <-> $(2^{n-1} - 1)$ ==> -16 <-> 15

$$\begin{array}{rcccccl} & 0 & 0 & 1 & 1 & 1 & 7(\text{DEC}) \\ + & 0 & 0 & 0 & 0 & 1 & 1(\text{DEC}) \\ \hline & 0 & 1 & 0 & 0 & 0 & 8(\text{DEC}) \end{array}$$

Extend 4-bit value to 5 bits to hold the correct result

What is sign extension?

- Sign-extension
  - Copying the sign bit of the un-extended value to all bits on the left side of the larger-size value
  - **SEXT** instruction widens the data while maintaining its sign and value.
  - e.g. widen the data while maintaining its sign and value
  - Unsigned number, converts positive values, provided the sign bit is zero

01001000 <- 8-bit value of 72
00000000 01001000 <- extended to 16-bit value
00000000 00000000 00000000 01001000 <- extended 32-bit value

What is sign extension?

- 8-bit encoding of decimal signed number -56 can be sign-extended as follows:

Sign

00111000 <- 8-bit value of 56

11000111 <- 8-bit value of -56 (1’s complement)

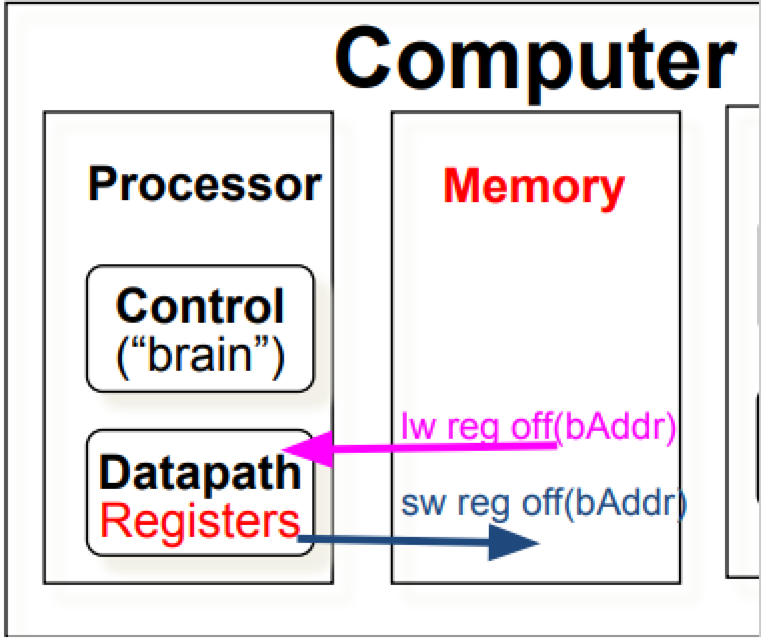
11001000 <- 8-bit value of -56 (2’s complement)

11111111 11001000 <- extended to 16-bit value

11111111 11111111 11111111 11001000 <- extended 32-bit value

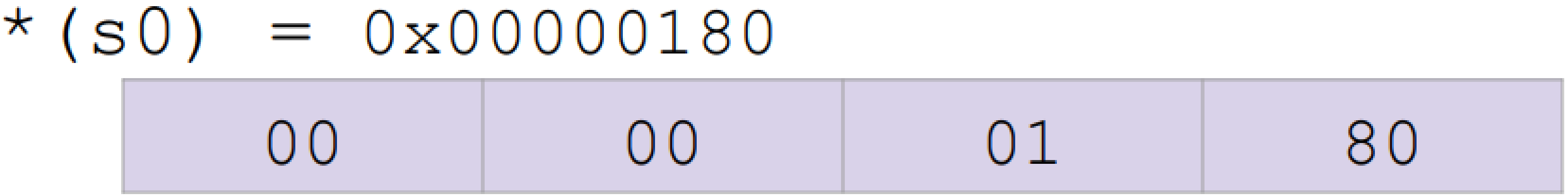
Memory and Variable Size

- So Far
  - lw reg, off(bAddr)
  - sw reg, off(bAddr)
- How to interact with memory values smaller than a word?
  - E.g. Characters (1B)
  - E.g. Shorts (sometimes 2B)



Trading Bytes with Memory

- **Method 1**: Move words in and out of memory using bit-masking and shifting
  - lw s0, 0(s1)
  - andi s0, s0, 0xFF # lowest byte
- **Method 2:** Load/store byte instructions
  - lb s1, 1(s0)
  - sb s1, 0(s0)



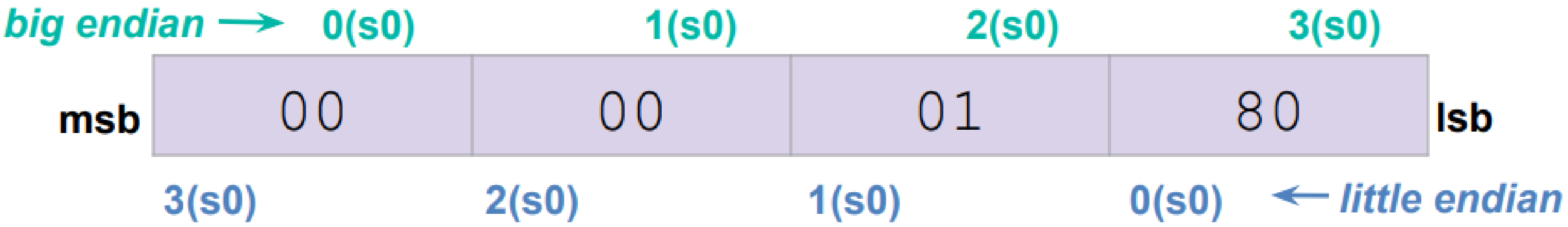
*(s0) = 0x00000180

00	00	01	80

Endianess

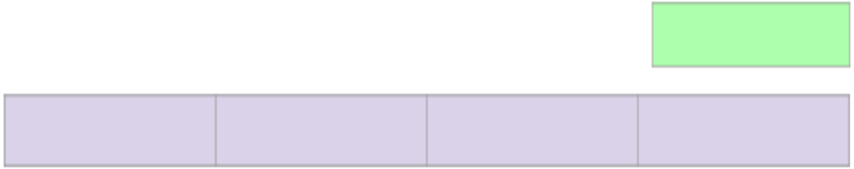
- **Big Endian**
  - Most significant byte at least address of word
  - Word address = address of the most significant byte
- **Little Endian**
  - Word address = address of the least significant byte
  - RISC-V is Little Endian

```
*(s0) = 0x00000180
```

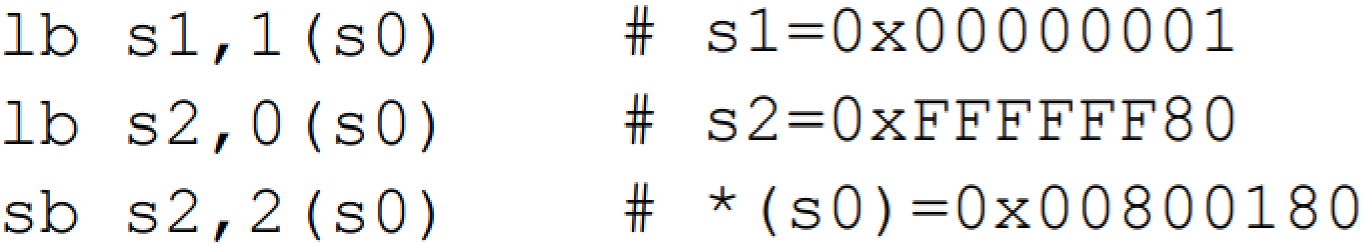


big endian →	0(s0)	1(s0)	2(s0)	3(s0)	
msb	00	00	01	80	lsb
	3(s0)	2(s0)	1(s0)	0(s0)	← *little endian*

Byte Instructions

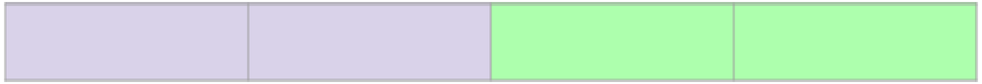


- lb/sb utilize the least significant bytes of the register
  - On sb, upper 24 bits are ignored
  - On lb, upper 24 bits are filled by sign-extension
- For example, let *(s0) = 0x00000180



```
lb s1,1(s0)      # s1=0x00000001
lb s2,0(s0)      # s2=0xFFFFFF80
sb s2,2(s0)      # *(s0)=0x00800180
```

Byte Instructions



- lh reg, off(bAddr) ‘load half’
- sh reg, off(bAddr) ‘store half’
  - On sh, upper 16 bits are ignored
  - On lh, upper 16 bits are filled by sign-extension
- Unsigned Instructions
  - lhu reg, off(bAddr) ‘load half unsigned’
  - lbu reg, off(bAddr) ‘load byte unsigned’
  - On l(b/h)u, upper bits are filled zero-extension

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12. 1(x5)
```

The range of the 12-bit signed immediate is -2^{12} <-> 2^{12} - 1

Sign

$\boxed{1}$000 0000 0000 ⇔ 1111 1111 1111

-2048(DEC) ⇔.   2047(DEC)

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12. 1(x5)
```

Sign

0x8f5 <=> 1000 1111 0101 (2’ complement) <=> -779(DEC)

1000 1111 0101 (2’complement) -> -779
1000 1111 0100 (1’ complement)
0111 0000 1011 (unsigned 779)

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12. 1(x5)
```

Sign

0x8f5 <=> 1000 1111 0101 (2’ complement) <=> -779(DEC)

1111 1111 1111 1111 1111 1000 1111 0101 (Signed extend 0x8f5 to 32-bits) => 0xfffff8f5

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12. 1(x5)
```

- **addi x11, x0, 0x8f5**
- The immediate got sign extended, x11 is 0xfffff8f5 because x11 is signed 32-bit register
- **sw x11, 0(x5)**
- the value of x11 is copied to x5 = 0xfffff8f5

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12, 1(x5)
```

- **lb x12, 1(x5)**
- Load byte sign extend to the register
- 0(x5) = 0xf5
- 1(x5) = 0xfffffff8