COMP1521 26T1

Week 4 Lecture 1

Integers

Announcements

- **Census Date** : Wednesday this week (12th March)
  - Last day to drop T1 courses without financial liability
- Weekly Test 3 Due: Thursday 9pm
- Assignment 1 Due: Week 5 Friday 18:00 (next week)
- See Help Session Schedule
- Lab marks usually come out about 2 weeks after deadline

Today's Lecture

- Invalid C
- Integers

Code Demos: Invalid C

invalid_1.c
invalid_4.c

See more interesting invalid C code examples at

https://cgi.cse.unsw.edu.au/~cs1521/26T1/topic/mips_functions/code/invalid0.c
https://cgi.cse.unsw.edu.au/~cs1521/26T1/topic/mips_functions/code/invalid2.c
https://cgi.cse.unsw.edu.au/~cs1521/26T1/topic/mips_functions/code/invalid3.c

Integers

Why Learn About Integers

- Fundamental topic in computing
  - Understand what you are seeing in mipsy web!
  - Understand limits of types and help you understand and debug code
- Prepare you for the next topic bitwise operators
- Understand the jokes in these slides

There are 10 types of students

Those that understand binary,

And those that don’t.

Numbers

4705

It is equivalent to: $\mathbf{4}*10^3 + \mathbf{7}*10^2 + \mathbf{0}*10^1 + \mathbf{5}*10^0$

$= \mathbf{4}000 + \mathbf{7}00 + \mathbf{0} + \mathbf{5}$

Numbers

4705

It is equivalent to: $\mathbf{4}*10^3 + \mathbf{7}*10^2 + \mathbf{0}*10^1 + \mathbf{5}*10^0$

$= \quad \mathbf{4}000 + \mathbf{7}00 + \mathbf{0} + \mathbf{5}$

If we assume it is base 10!

Base 10: Decimal

- In Base (or radix) 10 we have 10 digits e.g. 0..9
  - Then to get bigger numbers we start combining the digits e.g. 10
- Place Values

10^3	10^2	10^1	10^0
1000_{10}	100_{10}	10_{10}	1_{10}

- Example:

$\mathbf{4705}_{10} = \mathbf{4}*10^3 + \mathbf{7}*10^2 + \mathbf{0}*10^1 + \mathbf{5}*10^0$

$= \mathbf{4}000 + \mathbf{7}00 + \mathbf{0} + \mathbf{5}$

$= 4705_{10}$

Base 10 was an arbitrary choice

- Possibly exists because we have 10 digits (fingers)
- Ancient Egyptians, Brahmi Numerals, Greek Numerals, Hebrew Numerals, Roman Numerals and Chinese Numerals:
  - All base 10!

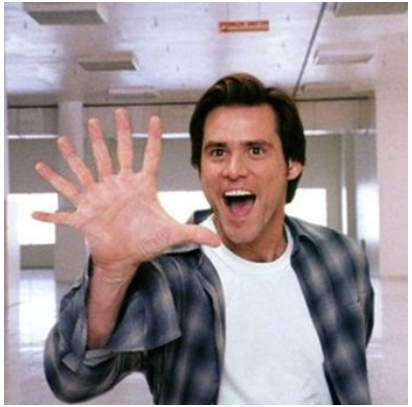
Code Demo: Digits.c

How can we get each digit from a number in decimal, one by one?

E.g. How can we get the 2 from 8192?

What about the 9?

What about some other bases?



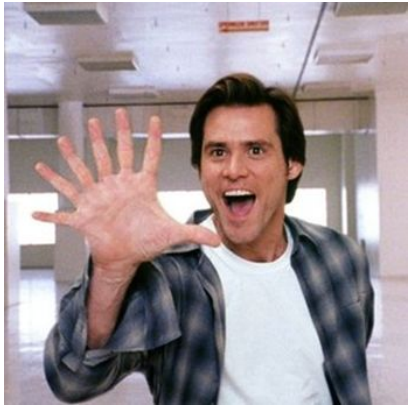
- Let's think about base 7 (not a very useful base)
- We have 7 digits 0..6
  - Then we start combining the digits

    e.g. 10_7 represents 7_{10}
- Place Values

7^3	7^2	7^1	7^0
343_{10}	49_{10}	7_{10}	1_{10}

Here, 1216_7 = ?

What about some other bases?



- Let's think about base 7 (not a very useful base)
- We have 7 digits 0..6
  - Then we start combining the digits

    e.g. 10_7 represents 7_{10}
- Place Values

7^3	7^2	7^1	7^0
343_{10}	49_{10}	7_{10}	1_{10}

Here, 1216_7 = **1** * 7^3 + **2** * 7^2 + **1** * 7^1 + **6** * 7^0

= **1** * 343 + **2** * 47 + **2** * 7 + **6** * 1

= 454_{10}

Base 2: Computers like binary

- In Base (or radix) 2 we have 2 digits (bits) – 0 and 1
  - Easy to represent using “electricity” – Off and On
  - Then we start combining the digits e.g. 10_2 represents 2_{10}
- Place Values

2^3	2^2	2^1	2^0
8_{10}	4_{10}	2_{10}	1_{10}

1011_2 =$?_{10}$

Base 2: Computers like binary

- In Base (or radix) 2 we have 2 digits (bits) – 0 and 1
  - Easy to represent using “electricity” – Off and On
  - Then we start combining the digits e.g. 10_2 represents 2_{10}
- Place Values

2^3	2^2	2^1	2^0
8_{10}	4_{10}	2_{10}	1_{10}

$$\mathbf{1011_2} = \mathbf{1} * 2^3 + \mathbf{0} * 2^2 + \mathbf{1} * 2^1 + \mathbf{1} * 2^0$$
$$= \mathbf{1} \times 8 + \mathbf{0} \times 4 + \mathbf{1} \times 2 + \mathbf{1} \times 1$$
$$= 11_{\mathbf{10}}$$

More examples

Question: Convert 1101_2 to decimal

Question: Convert 25_{10} to binary

More examples

Question: Convert 1101_2 to decimal

Answer: $1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

$= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$

$= 13$

Question: Convert 25_{10} to binary

More examples

Question: Convert 1101_2 to decimal

Answer: $1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

$= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$

$= 13$

Question: Convert 25_{10} to binary = 11001

- 25/2 = 12 R 1 (this is the least significant bit)
- 12/2 = 6 R 0
- 6/2 = 3 R 0
- 3/2 = 1 R 1
- 1/2 = 0 R 1 (this is the most significant bit)

Binary numbers are hard to read!

- They get very long, very fast
- E.g. $12345678_{10} = 101111000110000101001110_2$

Binary numbers are hard to read!

- They get very long, very fast
- E.g. $12345678_{10} = 101111000110000101001110_2$
- Solution: Write numbers in hexadecimal!
  - More compact than binary
    - 4 binary digits replaced by 1 hexadecimal digit
    - Bit patterns remain more obvious than in decimal

Base 16: Hexadecimal

- In Base (or radix) 16 we have 16 digits (0..15)
  - 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - Then we start combining the digits e.g. 10 represents 16_{10}
- Place Values

16^3	16^2	16^1	16^0
4096_{10}	256_{10}	16_{10}	1_{10}

- $3AF1_{16}$ = $?_{10}$

Base 16: Hexadecimal

- In Base (or radix) 16 we have 16 digits
  - 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - Then we start combining the digits e.g. 10 represents 16_{10}
- Place Values

16^3	16^2	16^1	16^0
4096_{10}	256_{10}	16_{10}	1_{10}

- $\mathbf{3AF1}_{16} = \mathbf{3} \times 16^3 + \mathbf{10} \times 16^2 + \mathbf{15} \times 16^1 + \mathbf{1} \times 16^0$

  $= 15089_{10}$

More hexadecimal examples

Question: Convert $1FF_{16}$ to decimal

Question: Convert 13_{10} to hexadecimal

More hexadecimal examples

Question: Convert $1FF_{16}$ to decimal

Answer: $1 \times 16^2 + 15 \times 16^1 + 15 \times 16^0 = 511_{10}$

Question: Convert 13_{10} to hexadecimal

More hexadecimal examples

Question: Convert $1FF_{16}$ to decimal?

Answer: $1 \times 16^2 + 15 \times 16^1 + 15 \times 16^0 = 511_{10}$

Question: Convert 13_{10} to hexadecimal?

Answer: D_{16}

Binary -> Hexadecimal

- Binary gets very long very quick
  - e.g. $12345678_{10} = 101111000110000101001110_2$
- Solution: Write numbers in hexadecimal!
- Place Values

16^3	16^2	16^1	16^0
4096_{10}	256_{10}	16_{10}	1_{10}

- Since **16** == $\mathbf{2^4}$
  - Each group of 4 bits to represents 1 hex digit

Binary -> Hexadecimal

$$10111100011000010100111 0_2$$

$$= 1011\ 1100\ 0110\ 0001\ 0100\ 1110_2$$

Binary -> Hexadecimal

$$101111000110000101001110_2$$
$$= 1011\ 1100\ 0110\ 0001\ 0100\ 1110_2$$

Each 4 bit group can be represented by **one** hexadecimal digit!

Base 10	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Base 16	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Base 2	1111	1110	1101	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001	0000

Binary -> Hexadecimal

$$10111100011000010100111 0_2$$
$$= 1011\ 1100\ 0110\ 0001\ 0100\ 1110_2$$
$$= \quad B \quad C \quad 6 \quad 1 \quad 4 \quad E$$

Each 4 bit group can be represented by **one** hexadecimal digit!

Base 10	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Base 16	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Base 2	1111	1110	1101	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001	0000

More examples

Binary $01101111_2 = ?_{16}$

Hexadecimal $BAD2_{16} = ?_2$

More examples

Binary $01101111_2 = 6F_{16}$

Hexadecimal $BAD2_{16} = ?_2$

More examples

Binary $01101111_2 = 6F_{16}$

Hexadecimal $BAD2_{16} = 10111010\ 11010010_2$

Base 8: Octal

- In Base (or radix) 8 we have 8 digits
  - 0 1 2 3 4 5 6 7
  - Then we start combining the digits e.g. 10 represents 8_{10}
- Similar advantages to hexadecimal
  - $8 = 2^3$ so group bits into 3:
  - Example: $72_8 = 111\ 010_2 = 3A_{16} = 58_{10}$

Base 10	7	6	5	4	3	2	1	0
Base 8	7	6	5	4	3	2	1	0
Base 2	111	110	101	100	011	010	001	000

Binary, Octal, Hexadecimal Summary

- In **binary**, (base 2), each digit represents **1** bit:
  - $01001000111110101011110010010111_2$
- In **octal**, (base 8), each digit represents **3** bits
  - $01\ 001\ 000\ 111\ 110\ 101\ 011\ 110\ 010\ 010\ 111_2$
  - $1\quad 1\quad 0\quad 7\quad 6\quad 5\quad 3\quad 6\quad 2\quad 2\quad 7_8$
- In **hexadecimal**, (base 16), each digit represents **4** bits:
  - $0100\ 1000\ 1111\ 1010\ 1011\ 1100\ 1001\ 0111_2$
  - $4\quad 8\quad F\quad A\quad B\quad C\quad 9\quad 7_{16}$

Doing maths in binary

4 = 00000100 +

1 = 00000001

---

6 = 00000110 +

7 = 00000111

Doing maths in binary

```
4 = 00000100   +
1 = 00000001
--------------
5 = 00000101
```

```
6 = 00000110   +
7 = 00000111
--------------
```

Doing maths in binary

```
4 = 00000100   +
1 = 00000001
--------------
5 = 00000101
```

```
6 = 00000110   +
7 = 00000111
--------------
13= 00001101
```

Bits, Bytes and Bases in C

Constants in C and MIPS assembly

- A number beginning with `0x` is hexadecimal
- A number beginning with `0` is octal
- A number beginning with `0b` is binary
- Otherwise, it is decimal
- Note that %d format specifier means decimal

```
printf("%d", 0x2A);      // prints 42
printf("%d", 052);       // prints 42
printf("%d", 0b101010);  // prints 42
printf("%d", 42);        // prints 42
```

Easy Base Conversions in C

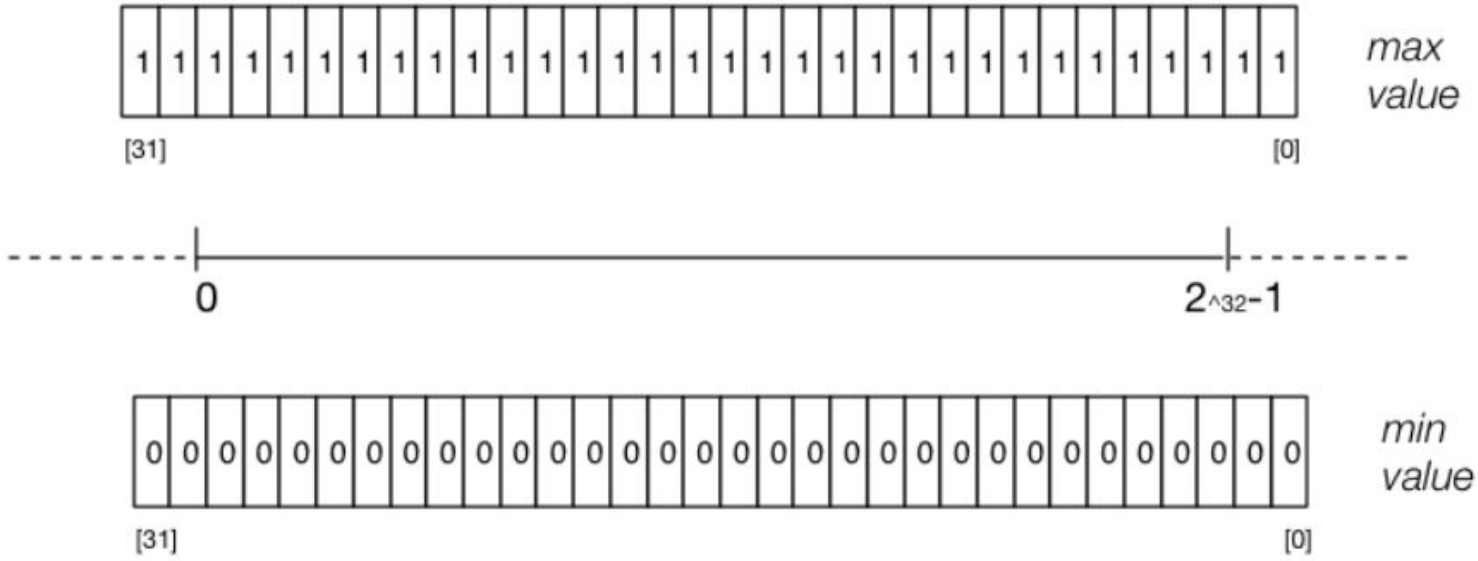
integer_prefixes.c

integer_prefixes_args.c

Integer Representations

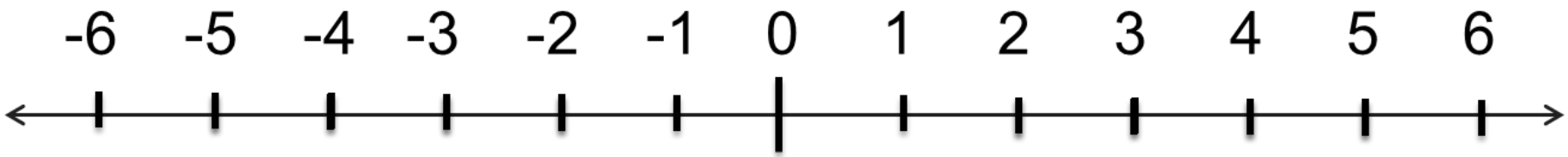
Unsigned integers

- In C the `unsigned int` data type is 4 bytes on our system
  - means we can store values from the range 0 .. 2^{32}-1

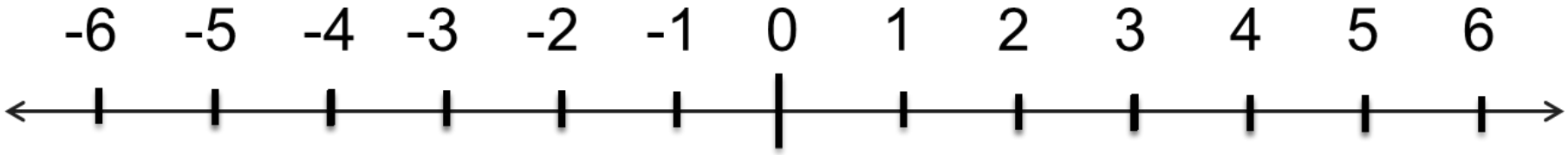


How do we store signed integers?

We COULD try to put a leading 0 for positive and a leading 1 for negative numbers



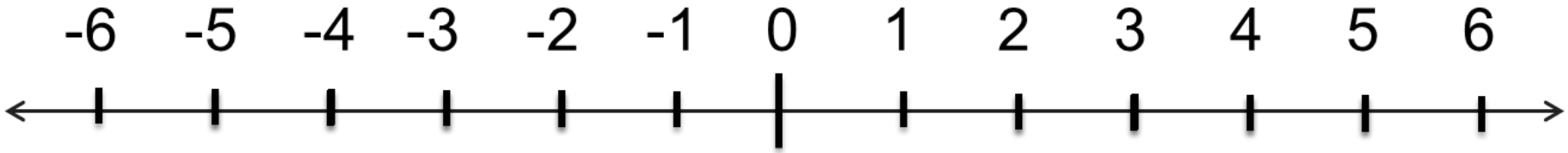
How do we store signed integers?



0000 0001 0010 0011 0100 0101 0110

1110 1101 1100 1011 1010 1001 1000

How do we store signed integers?



- Two representations for 0?
- Okay, but what algorithm for adding/subtracting numbers?

What do signed binary numbers look like?

4 = 00000100 -1
3 = 00000011 -1
2 = 00000010 -1
1 = 00000001 -1
0 = 00000000

What do signed binary numbers look like?

4 = 00000100 -1
3 = 00000011 -1
2 = 00000010 -1
1 = 00000001 -1
0 = 00000000

Let’s keep subtracting

What do signed binary numbers look like?

4 = 00000100 -1
3 = 00000011 -1
2 = 00000010 -1
1 = 00000001 -1
0 = 00000000 -1
-1= 11111111 -1
-2= 11111110 -1
-3= 11111101

What do signed binary numbers look like?

4 = 00000100 -1
3 = 00000011 -1
2 = 00000010 -1
1 = 00000001 -1
0 = 00000000 -1
-1= 11111111 -1
-2= 11111110 -1
-3= 11111101 -1

This representation is called

two's complement

- We don't have 2 zeros
- Subtraction works in the same way and so does addition. (Try it)

Note: For a binary signed number

Positive numbers start with **0**

Negative numbers start with **1**

What do signed binary numbers look like?

- Modern computers use **two's complement** for integers
- Positive integers and zero represented as normal
- Negative integers represented in a way to make maths ✨easy✨ for the computer (not humans)
  - For an *n*-bit binary number, the number -*b* is $2^n - b$
  - E.g. 8-bit number "-5" is represented as $2^8 - 5 = 251 = 1111\ 1011_2$
  - Conversely, $1111\ 1011_2$ represents "−5" because $251 - 2^8 = -5$.

Two's Complement Tips and Tricks

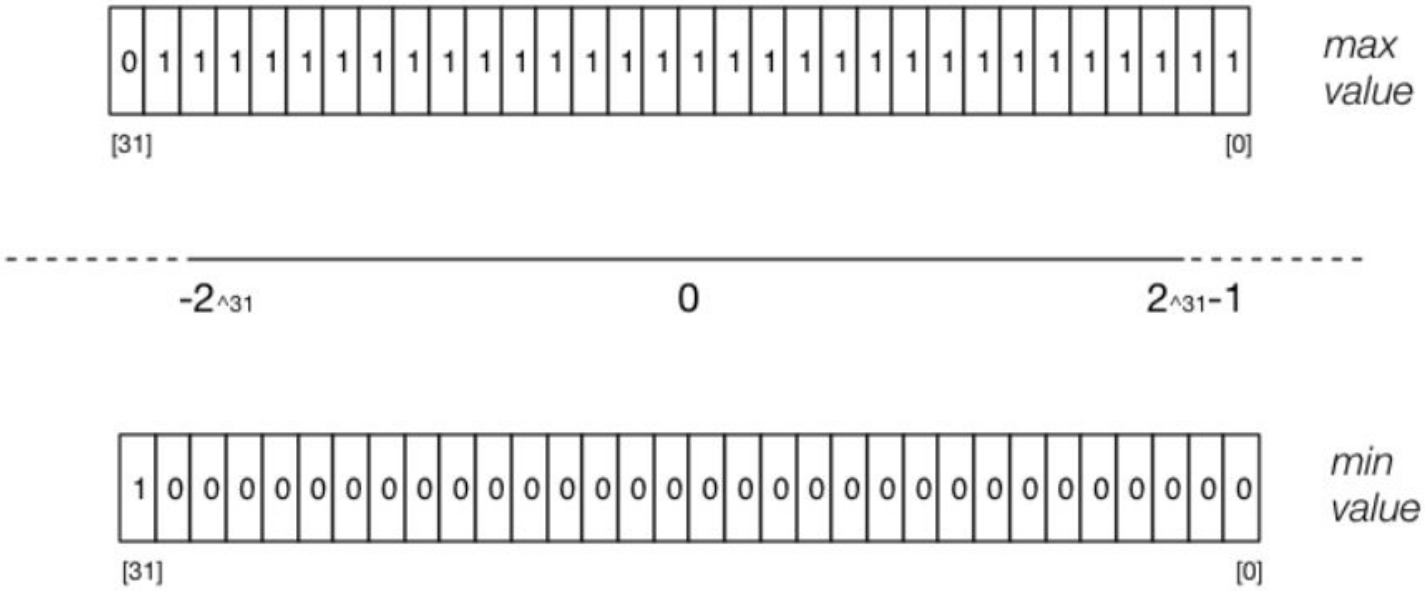
- Another way of thinking about 2's complement
  - If you are trying to represent -5 in 8 bits

Take the +5 representation:	0000 0101
invert all the bits:	1111 1010
add 1:	1111 1011

- Repeat the process to go from -5 back to 5 again!

Signed integers

- In C the `int` data type is 4 bytes on our system
  - we can store values from the range $-2^{31} .. 2^{31} -1$



Example: 2's Complement Example

- Some simple code to examine 8-bit 2's complement numbers:

```
for (int i = -128; i < 128; i++) {
    printf("%4d ", i);
    print_bits(i, 8);
    printf("\n");
}
```

- `dcc 8_bit_twos_complement.c print_bits.c -o 8_bit_twos_complement`

Example: Printing all 8-bit 2's complement

```
$ ./8_bit_twos_complement
-128 10000000
-127 10000001
-126 10000010
...
-3 11111101
-2 11111110
-1 11111111
0 00000000
1 00000001
2 00000010
3 00000011
...
125 01111101
126 01111110
127 01111111
```

Exercise 8-bit numbers

What do the following bit patterns represent:
11111111
10000000

Assume they are unsigned 8 bit integers:

Now assume they are signed 8 bit integers:

Exercise 8-bit numbers

What do the following bit patterns represent:
11111111
10000000

Assume they are unsigned 8 bit integers:
255
128
Now assume they are signed 8 bit integers:

Exercise 8-bit numbers

What do the following bit patterns represent:
11111111
10000000

Assume they are unsigned 8 bit integers:
255
128
Now assume they are signed 8 bit integers: (Note 2^8 == 256)
-1 (255 - 256)
-128 (128 - 256)

Example: print_bits_of_int.c

```
$ ./print_bits_of_int
Enter an int: 0
00000000000000000000000000000000
$ ./print_bits_of_int
Enter an int: 1
00000000000000000000000000000001
$ ./print_bits_of_int
Enter an int: -1
11111111111111111111111111111111
$ ./print_bits_of_int
Enter an int: 2147483647
01111111111111111111111111111111
$ ./print_bits_of_int
Enter an int: -2147483648
10000000000000000000000000000000
$
```

Bits and Bytes on cse Servers

- On CSE servers, C types have these sizes
  - **char** = 1 byte = 8 bits
    - 42 is 00101010
  - **short** = 2 bytes = 16 bits,
    - 42 is 0000000000101010
  - **int** = 4 bytes = 32 bits,
    - 42 is 00000000000000000000000000101010
  - **double** = 8 bytes = 64 bits,
    - 42 = ?
- above are common sizes but not universal
- sizeof (int) might be 2 (bytes) on a small embedded CPU

integer_types.c - exploring integer types

Type	Bytes	Bits
char	1	8
signed char	1	8
unsigned char	1	8
short	2	16
unsigned short	2	16
int	4	32
unsigned int	4	32
long	8	64
unsigned long	8	64
long long	8	64
unsigned long long	8	64

Exploring integer types and <limits.h>

Type	Min	Max
char	-128	127
signed char	-128	127
unsigned char	0	255
short	-32768	32767
unsigned short	0	65535
int	-2147483648	2147483647
unsigned int	0	4294967295
long	-9223372036854775808	9223372036854775807
unsigned long	0	18446744073709551615
long long	-9223372036854775808	9223372036854775807
unsigned long long	0	18446744073709551615

stdint.h - guaranteed size integer types

- `#include <stdint.h>` to get below int types (and more) with known sizes
- We use these a lot in COMP1521!

```
                // range of values for type
                //                minimum                  maximum
int8_t    i1; //                   -128                      127
uint8_t   i2; //                      0                      255
int16_t   i3; //                 -32768                    32767
uint16_t  i4; //                      0                    65535
int32_t   i5; //            -2147483648               2147483647
uint32_t  i6; //                      0               4294967295
int64_t   i7; // -9223372036854775808  9223372036854775807
uint64_t  i8; //                      0 18446744073709551615
```

Code Examples

overflow_int.c

wrap_around_uint.c

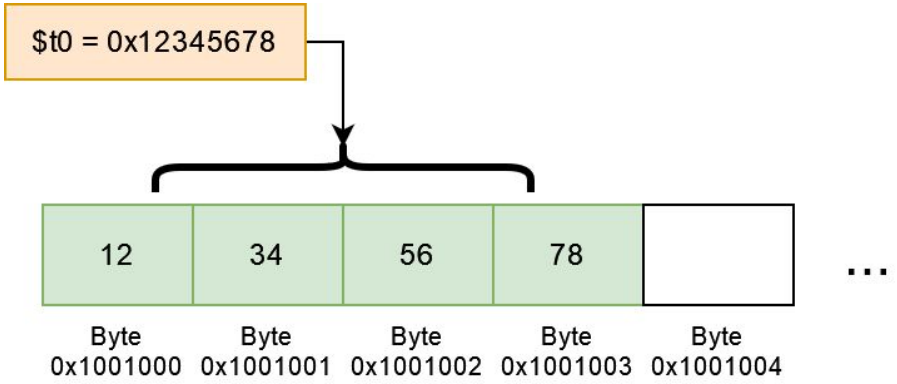
char_bug.c

unsigned_arithmetic.s

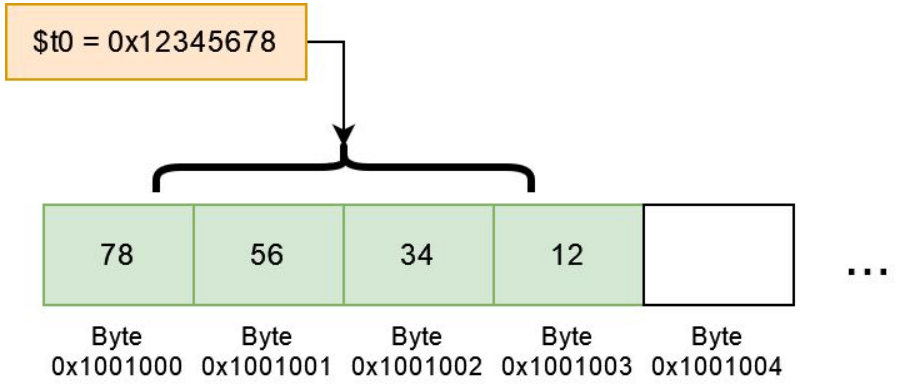
New concept: Endian-ness

- “What order to put things in” is a hard question to answer
- Two schools of thought:
  - **Big**-endian: MSB at the “low address” - big bits “first!”
  - **Little**-endian: LSB at the “low address” - little bits “first!”

BIG:



LITTLE:

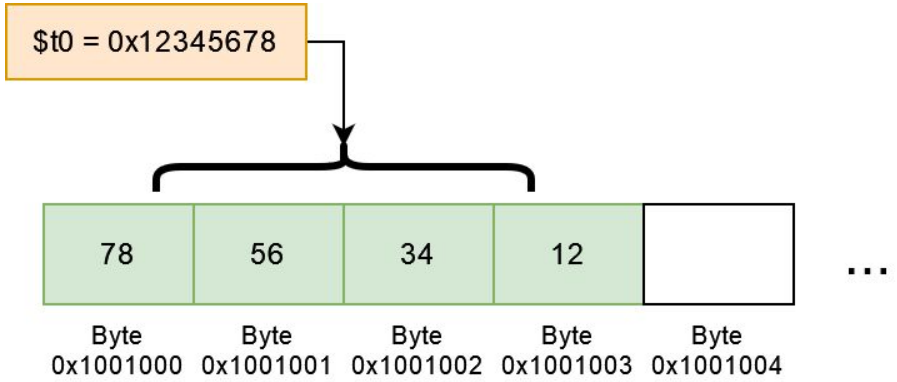


Code example

- Mipsy-web is **little-endian**

```
        .text
main:
        li $t0, 0x12345678
        sw $t0, my_word

        .data
my_word:
        .space 4
```



Loading bytes, half-words

The results of these will depend on endianness:

- `lh/lb` assume the loaded byte/halfword is signed
- `lhu/lbu` for doing the same thing, but unsigned

- What might we need to do differently if we are loading a signed vs unsigned number?
- Why did we not need 2 versions for lw?

Loading bytes, half-words

- Consider storing 0x80 in a 4 byte (32 bit register) register
- This only takes up 1 byte (8 bits)
- What do we store in the rest of the register?

??	??	??	10000000 (0x80)

Loading bytes, half-words

- Consider storing 0x80 in a 4 byte (32 bit register) register
- This only takes up 1 byte (8 bits)
- What do we store in the rest of the register?

00000000 (0x00)	00000000 (0x00)	00000000 (0x00)	10000000 (0x80)

If we set them all the 0s, what happens if 0x80 is meant to be treated as a signed value?

Loading bytes, half-words

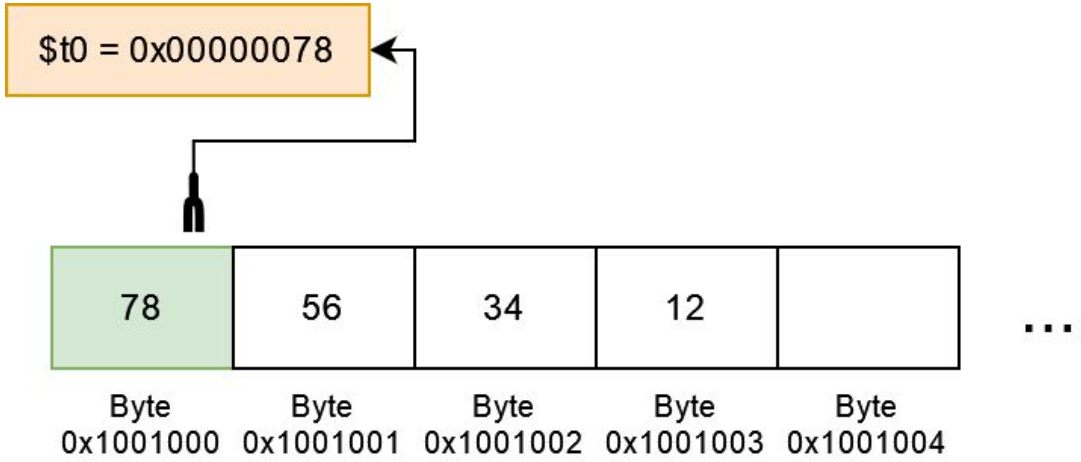
- Consider storing 0x80 in a 4 byte (32 bit register) register
- This only takes up 1 byte (8 bits)
- What do we store in the rest of the register?

11111111 (0xFF)	11111111 (0xFF)	11111111 (0xFF)	10000000 (0x80)

If our value is meant to be signed we extend the sign bit. For a negative value this means extending with 1

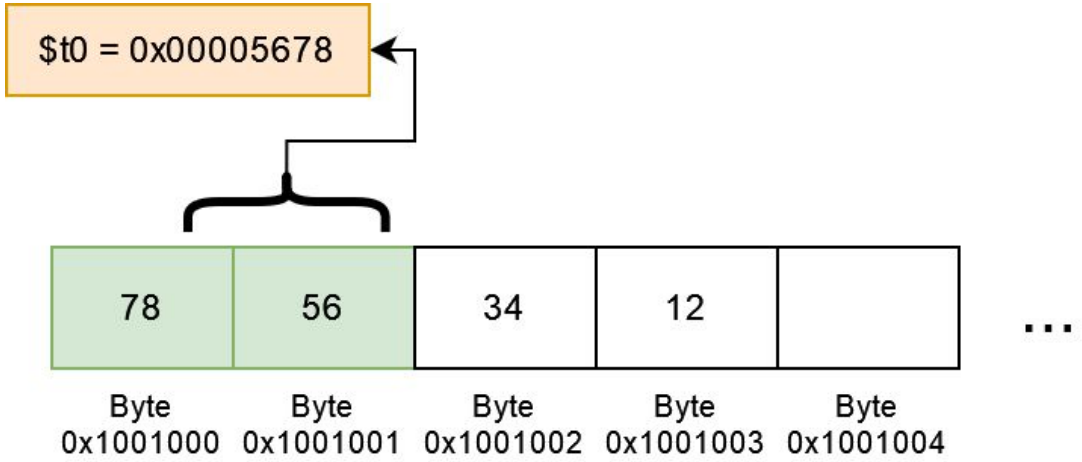
Loading Examples: lb

```
.text
main:
        lb $t0, my_label
.data
my_label:
        .word 0x12345678
```



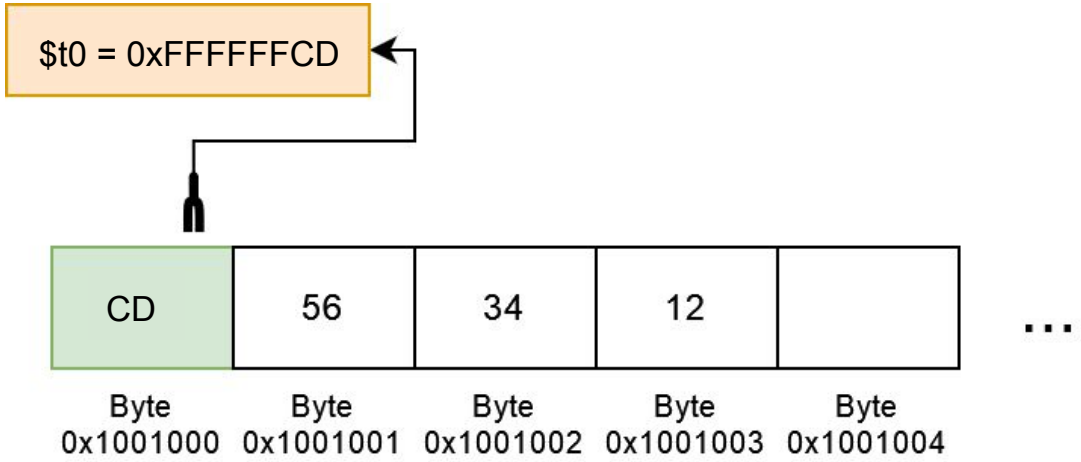
Loading Examples: lh

```
.text
main:
        lh $t0, my_label
.data
my_label:
        .word 0x12345678
```



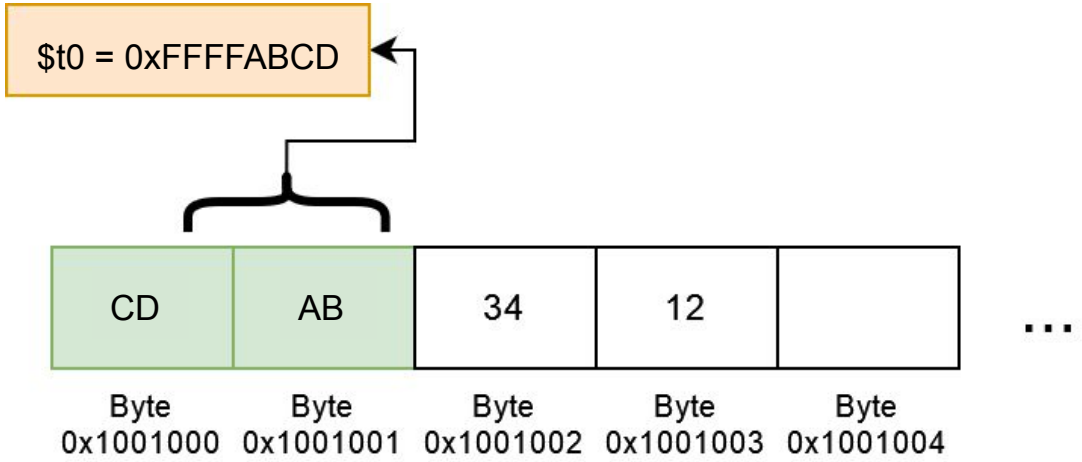
Loading Examples Negative: lb

```
.text
main:
        lb $t0, my_label
.data
my_label:
        .word 0x1234ABCD
```



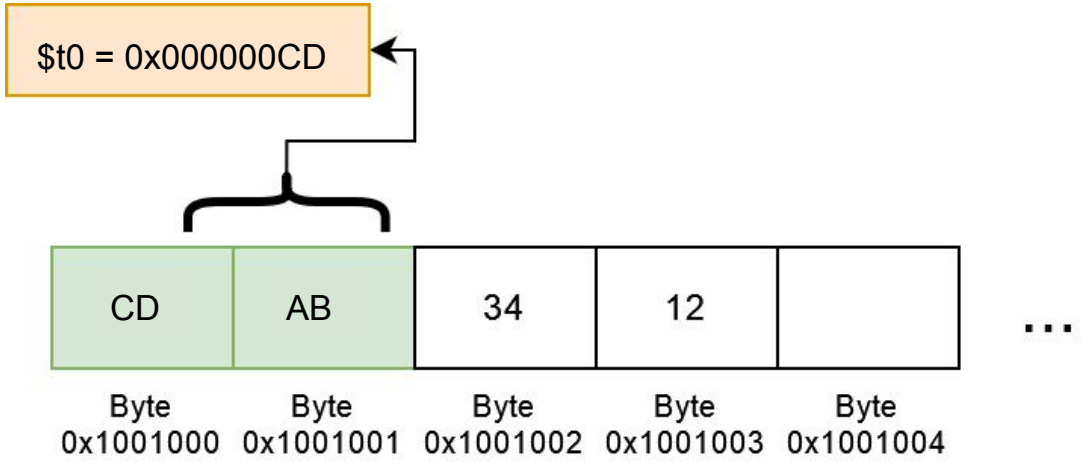
Loading Examples Negative: lh

```
.text
main:
        lh $t0, my_label
.data
my_label:
        .word 0x1234ABCD
```



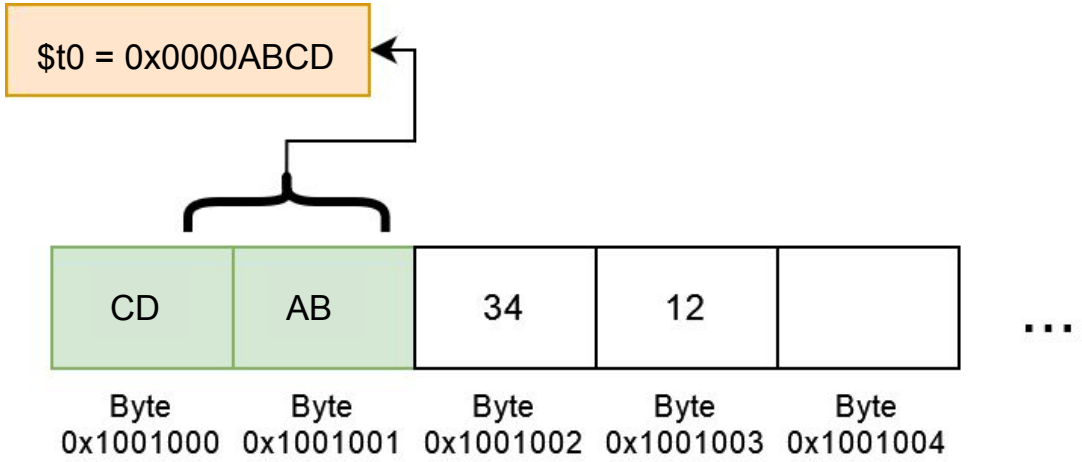
Loading Examples: lbu

```
.text
main:
        lbu $t0, my_label
.data
my_label:
        .word 0x1234ABCD
```



Loading Examples Negative: lhu

```
.text
main:
        lhu $t0, my_label
.data
my_label:
        .word 0x1234ABCD
```



Endianness in C

endianness.c

What did we learn today?

- Invalid C
- Integers
  - Endianness
  - Unsigned instructions in MIPS
- Next lecture:
  - Bitwise operators

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.

https://forms.office.com/r/GXGX6vKDG6

Reach Out

Content Related Questions:

Forum

Admin related Questions email:

cs1521@cse.unsw.edu.au

Student Support | I Need Help With...

My Feelings and Mental Health
Managing Low Mood, Unusual Feelings & Depression

Mental Health Connect
student.unsw.edu.au/**counselling**
Telehealth

Mind HUB
student.unsw.edu.au/**mind-hub**
Online Self-Help Resources

In Australia Call Afterhours UNSW Mental Health Support Line
1300 787 026
5pm-9am

Outside Australia Afterhours 24-hour Medibank Hotline
+61 (2) 8905 0307

Uni and Life Pressures
Stress, Financial, Visas, Accommodation & More

Student Support
Indigenous Student Support
student.unsw.edu.au/**advisors**

Reporting Sexual Assault/Harassment

Equity Diversity and Inclusion (EDI)
edi.unsw.edu.au/**sexual-misconduct**

Educational Adjustments
To Manage my Studies and Disability / Health Condition

Equitable Learning Service (ELS)
student.unsw.edu.au/**els**

Academic and Study Skills

Academic Language Skills
student.unsw.edu.au/**skills**

Special Consideration
Because Life Impacts our Studies and Exams

Special Consideration
student.unsw.edu.au/**special-consideration**