

COMP 3221

Microprocessors and Embedded Systems

Lecture 5: Number Systems – I

<http://www.cse.unsw.edu.au/~cs3221>

August, 2003

Saeid Nooshabadi

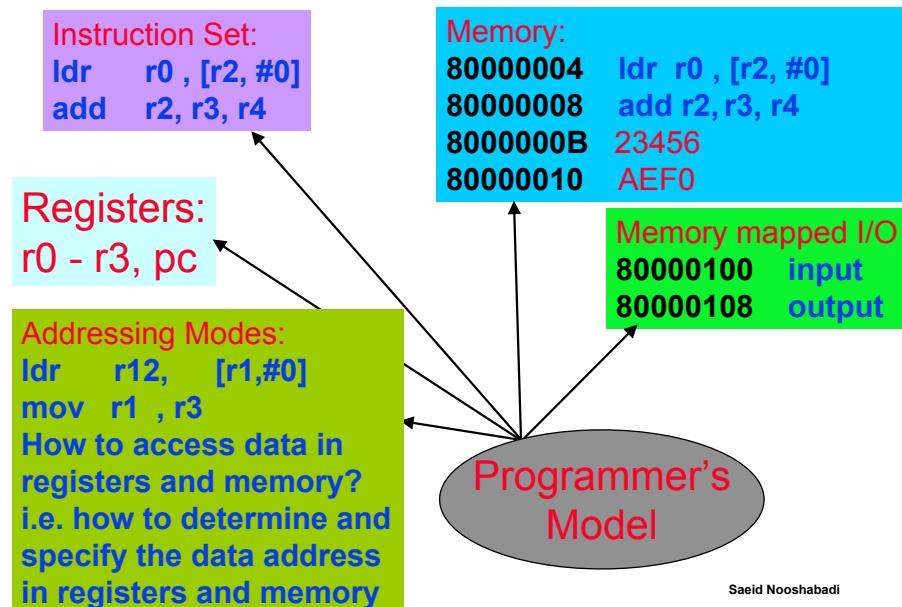
Saeid@unsw.edu.au

COMP3221 lec05-numbers-I.1

Saeid Nooshabadi

Saeid Nooshabadi

Review: The Programmer's Model of a Microcomputer



Saeid Nooshabadi

Overview

- ° Computer representation of “things”
- ° Unsigned Numbers
- ° Signed Numbers: search for a good representation
- ° Shortcuts
- ° In Conclusion

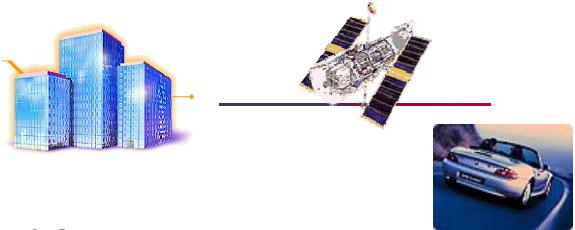
Review: Compilation

- ° How to turn notation programmers prefer into notation computer understands?
- ° Program to translate C statements into Assembly Language instructions; called a **compiler**
- ° Example: compile by hand this C code:
 $a = b + c;$
 $d = a - e;$
- ° Easy: add r1, r2, r3
sub r4, r5, r6
- ° Big Idea: compiler translates notation from 1 level of abstraction to lower level

COMP3221 lec05-numbers-I.4

Saeid Nooshabadi

What do computers



- ° Computers **manipulate representations of things!**

- ° What can you represent with N bits?

- 2^N things!

- ° Which things?

- Numbers! Characters! Pixels! Dollars! Position! Instructions!
...
 - Depends on what operations you do on them

COMP3221 lec05-numbers-I.5

Saeid Nooshabadi

Numbers: positional notation

- ° Number Base B => B symbols per digit:

- Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - Base 2 (Binary): 0, 1

- ° Number representation:

- $d_{31}d_{30} \dots d_2d_1d_0$ is a 32 digit number
 - value = $d_{31} \times B^{31} + d_{30} \times B^{30} + \dots + d_2 \times B^2 + d_1 \times B^1 + d_0 \times B^0$

- ° Binary: 0,1

- $1011010 = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 0 \times 1 = 64 + 16 + 8 + 2 = 90$
 - Notice that 7 digit binary number turns into a 2 digit decimal number
 - A base that converts to binary easily?

COMP3221 lec05-numbers-I.7

Saeid Nooshabadi

Decimal Numbers: Base 10

- ° Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- ° Example:

$$3271 =$$

$$(3 \times 10^3) + (2 \times 10^2) + (7 \times 10^1) + (1 \times 10^0)$$

COMP3221 lec05-numbers-I.6

Saeid Nooshabadi

Hexadecimal Numbers: Base 16 (#1/2)

- ° Digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

- ° Normal digits have expected values

- ° In addition:

- A → 10
 - B → 11
 - C → 12
 - D → 13
 - E → 14
 - F → 15

COMP3221 lec05-numbers-I.8

Saeid Nooshabadi

Hexadecimal Numbers: Base 16 (#2/2)

- ° Example (convert hex to decimal):

$$\begin{aligned} \text{B28F0DD} &= (\text{B} \times 16^6) + (2 \times 16^5) + (8 \times 16^4) + (\text{F} \times 16^3) + (0 \times 16^2) + \\ &\quad (\text{D} \times 16^1) + (\text{D} \times 16^0) \\ &= (11 \times 16^6) + (2 \times 16^5) + (8 \times 16^4) + (15 \times 16^3) + \\ &\quad (0 \times 16^2) + (13 \times 16^1) + (13 \times 16^0) \\ &= 187232477 \text{ decimal} \end{aligned}$$

- ° Notice that a 7 digit hex number turns out to be a 9 digit decimal number

COMP3221 lec05-numbers-I.9

Saeid Nooshabadi

Hex to Binary Conversion

- ° HEX is a more compact representation of Binary!
- ° Each hex digit represents 16 decimal values.
- ° Four binary digits represent 16 decimal values.
- ° Therefore, each hex digit can replace four binary digits.
- ° Example:

0011 1011 1001 1010 1100 1010 0000 0000_{two}
3 b 9 a c a 0 0_{hex}
C uses notation 0x3b9aca00

COMP3221 lec05-numbers-I.11

Saeid Nooshabadi

Decimal vs. Hexadecimal vs. Binary

- ° Examples:

$$\begin{aligned} &00\ 0\ 0000 \\ &01\ 1\ 0001 \\ &02\ 2\ 0010 \\ &03\ 3\ 0011 \\ &04\ 4\ 0100 \\ &05\ 5\ 0101 \\ &06\ 6\ 0110 \\ &07\ 7\ 0111 \\ &08\ 8\ 1000 \\ &09\ 9\ 1001 \\ &10\ A\ 1010 \\ &11\ B\ 1011 \\ &12\ C\ 1100 \\ &13\ D\ 1101 \\ &14\ E\ 1110 \\ &15\ F\ 1111 \end{aligned}$$

° 10111 (binary)

= 0001 0111 (binary)

= ? (hex)

° 3F9(hex)

= ? (binary)

COMP3221 lec05-numbers-I.10

Saeid Nooshabadi

Which Base Should We Use?

- ° Decimal: Great for humans; most arithmetic is done with these.
- ° Binary: This is what computers use, so get used to them. Become familiar with how to do basic arithmetic with them (+, -, *, /).
- ° Hex: Terrible for arithmetic; but if we are looking at long strings of binary numbers, it's much easier to convert them to hex in order to look at four bits at a time.

COMP3221 lec05-numbers-I.12

Saeid Nooshabadi

How Do We Tell the Difference?

- In general, append a subscript at the end of a number stating the base:

- 10_{10} is in decimal
- 10_2 is binary ($= 2_{10}$)
- 10_{16} is hex ($= 16_{10}$)

- When dealing with ARM computer:

- Hex numbers are preceded with “&” or “0x”
 - $\&10 == 0x10 == 10_{16} == 16_{10}$
 - Note: Lab software environment only supports “0x”
- Binary numbers are preceded with “0b”
- Octal numbers are preceded with “0”
- Everything else by default is Decimal

COMP3221 lec05-numbers-I.13

Saeid Nooshabadi

COMP3221 lec05-numbers-I.14

Saeid Nooshabadi

What to do with representations of numbers?

- Just what we do with numbers!

- Add them
- Subtract them
- Multiply them
- Divide them
- Compare them

1 1

1 0 1 0

+ 0 1 1 1

- Example: $10 + 7 = 17$

1 0 0 0 1

- so simple to add in binary that we can build circuits to do it
- subtraction also just as you would in decimal

COMP3221 lec05-numbers-I.15

Saeid Nooshabadi

COMP3221 lec05-numbers-I.16

Saeid Nooshabadi

Inside the Computer

- To a computer, numbers are always in binary; all that matters is how they are printed out: binary, decimal, hex, etc.

- As a result, it doesn’t matter what base a number in C is in...

- $32_{10} == 0x20 == 100000_2$

- ... only the value of the number matters.

Addition Table

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Addition Table (binary)

$$\begin{array}{r}
 + \quad 0 \quad 1 \\
 \hline
 0 \quad | \quad 0 \quad 1 \\
 1 \quad | \quad 1 \quad 10
 \end{array}$$

COMP3221 lec05-numbers-I.17

Saeid Nooshabadi

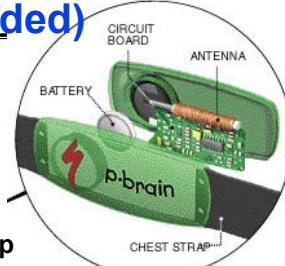
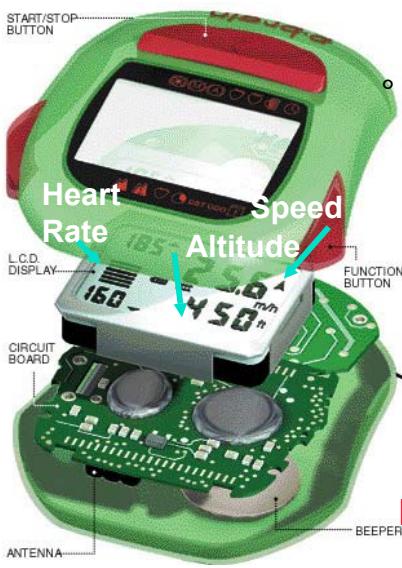
Addition Table (Hex)

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

COMP3221 lec05-numbers-I.18

Saeid Nooshabadi

Bicycle Computer (Embedded)



° P. Brain

- wireless heart monitor strap
- record 5 measures: speed, time, current distance, elevation and heart rate
- Every 10 to 60 sec.
- 8KB data => 33 hours
- Stores information so can be uploaded through a serial port into PC to be analyzed

<http://www.specialized.com>

COMP3221 lec05-numbers-I.19

Saeid Nooshabadi

Limits of Computer Numbers

° Bits can represent anything!

° Characters?

- 26 letter => 5 bits
- upper/lower case + punctuation => 7 bits (in 8) (ASCII)
- rest of the world's languages => 16 bits (unicode)

° Logical values?

- 0 -> False, 1 => True

° colors ?

° locations / addresses? commands?

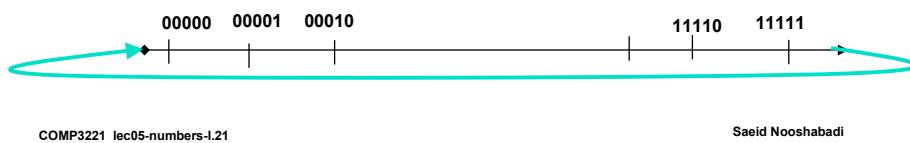
° but N bits => only 2^N things

COMP3221 lec05-numbers-I.20

Saeid Nooshabadi

What if too big?

- Binary bit patterns above are simply representatives of numbers
- Numbers really have an infinite number of digits
 - with almost all being zero except for a few of the rightmost digits: e.g: **0000000 ... 000098 == 98**
 - Just don't normally show leading zeros
- Computers have fixed number of digits
 - In general, adding two n-bit numbers can produce an $(n+1)$ -bit result.
 - Since computers use fixed, 32-bit integers, this is a problem.
 - If result of add (or any other arithmetic operation), cannot be represented by these rightmost hardware bits, overflow is said to have occurred



COMP3221 lec05-numbers-I.21

Saeid Nooshabadi

How avoid overflow, allow it sometimes?

- Some languages detect overflow (Ada), some don't (C and JAVA)
- ARM has N, Z, C and V flags to keep track overflow
 - Refer Book!
 - Will cover details later



COMP3221 lec05-numbers-I.23

Saeid Nooshabadi

Overflow Example

- Example (using 4-bit numbers):

$$\begin{array}{r} +15 & 1111 \\ +3 & \underline{0011} \\ \hline +18 & 10010 \end{array}$$

- But we don't have room for 5-bit solution, so the solution would be 0010, which is +2, which is wrong.

COMP3221 lec05-numbers-I.22

Saeid Nooshabadi

Comparison

- How do you tell if $X > Y$?
 - See if $X - Y > 0$
- We need representation for both +ve and -ve numbers

COMP3221 lec05-numbers-I.24

Saeid Nooshabadi

How to Represent Negative Numbers?

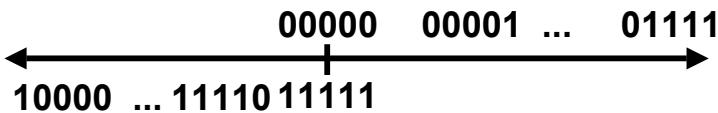
- So far, **unsigned numbers**
- Obvious solution: define leftmost bit to be sign!
 - $0 \Rightarrow +$, $1 \Rightarrow -$
 - Rest of bits can be numerical value of number
- Representation called **sign and magnitude**
- ARM uses 32-bit integers. $+1_{\text{ten}}$ would be:
0000 0000 0000 0000 0000 0000 0001
- And -1_{ten} in sign and magnitude would be:
1000 0000 0000 0000 0000 0000 0001

COMP3221 lec05-numbers-I.25

Saeid Nooshabadi

Another try: complement the bits

- Example: $7_{10} = 00111_2$ $-7_{10} = 11000_2$
- Called **one's Complement**
- Note: positive numbers have leading 0s, negative numbers have leadings 1s.



- What is -00000 ?
- How many positive numbers in N bits?
- How many negative ones?

COMP3221 lec05-numbers-I.27

Saeid Nooshabadi

Shortcomings of sign and magnitude?

- Arithmetic circuit more complicated
 - Special steps depending whether signs are the same or not
- Also, Two zeros
 - $0x00000000 = +0_{\text{ten}}$
 - $0x80000000 = -0_{\text{ten}}$
 - What would it mean for programming?
- Sign and magnitude abandoned because another solution was better

COMP3221 lec05-numbers-I.26

Saeid Nooshabadi

Shortcomings of ones complement?

- Arithmetic not too hard
- Still two zeros
 - $0x00000000 = +0_{\text{ten}}$
 - $0xFFFFFFF = -0_{\text{ten}}$
 - What would it mean for programming?
- One's complement eventually abandoned because another solution was better

COMP3221 lec05-numbers-I.28

Saeid Nooshabadi

Search for Negative Number Representation

- Obvious solution didn't work, find another
 - What is result for unsigned numbers if tried to subtract large number from a small one?
 - Would try to borrow from string of leading 0s, so result would have a string of leading 1s
- $3 - 7 = -4$
- $\begin{array}{r} 111 \\ 000011 \\ -000111 \\ \hline 11100 \end{array}$
- With no obvious better alternative, pick representation that made the hardware simple: leading 0s => positive, leading 1s => negative
 - 000000...xxx is ≥ 0 , 111111...xxx is < 0

◦ This representation called **two's complement**

COMP3221 lec05-numbers-I.29

Saeid Nooshabadi

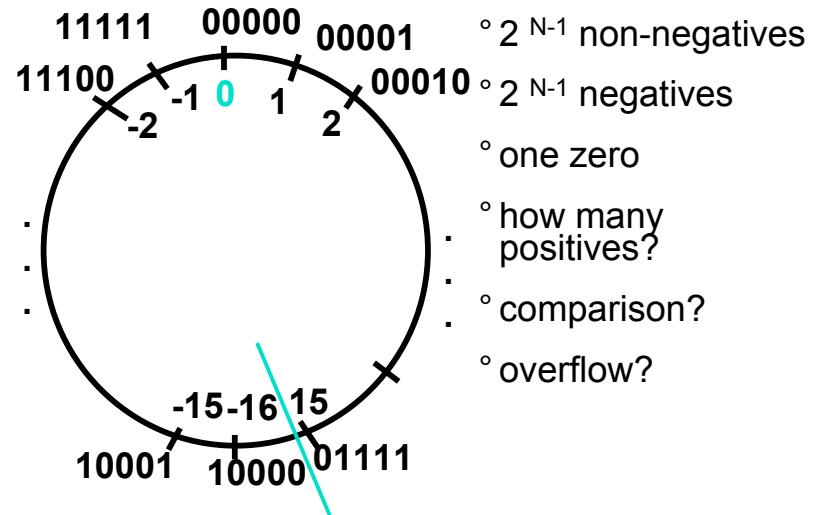
Two's Complement

0000 ... 0000 0000 0000 0000 _{two} =	0 _{ten}
0000 ... 0000 0000 0001 _{two} =	1 _{ten}
0000 ... 0000 0000 0010 _{two} =	2 _{ten}
⋮	
0111 ... 1111 1111 1111 1101 _{two} =	2,147,483,645 _{ten}
0111 ... 1111 1111 1111 1110 _{two} =	2,147,483,646 _{ten}
0111 ... 1111 1111 1111 1111 _{two} =	2,147,483,647 _{ten}
1000 ... 0000 0000 0000 0000 _{two} =	-2,147,483,648 _{ten}
1000 ... 0000 0000 0000 0001 _{two} =	-2,147,483,647 _{ten}
1000 ... 0000 0000 0000 0010 _{two} =	-2,147,483,646 _{ten}
⋮	
1111 ... 1111 1111 1111 1101 _{two} =	-3 _{ten}
1111 ... 1111 1111 1111 1110 _{two} =	-2 _{ten}
1111 ... 1111 1111 1111 1111 _{two} =	-1 _{ten}

- One zero, 31st bit => ≥ 0 or < 0 , called **sign bit**

- but one negative with no positive $-2,147,483,648_{ten}$

2's Complement Number line



COMP3221 lec05-numbers-I.30

Saeid Nooshabadi

Two's Complement Formula, Example

- Recognizing role of sign bit, can represent positive **and negative** numbers in terms of the bit value times a power of 2:
 - $d_{31} \times -2^{31} + d_{30} \times 2^{30} + \dots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$
- Example

$$\begin{aligned}
 &1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two} \\
 &= 1 \times -2^{31} + 1 \times 2^{30} + 1 \times 2^{29} + \dots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\
 &= -2^{31} + 2^{30} + 2^{29} + \dots + 2^2 + 0 + 0 \\
 &= -2,147,483,648_{ten} + 2,147,483,644_{ten} \\
 &= -4_{ten}
 \end{aligned}$$

COMP3221 lec05-numbers-I.32

Saeid Nooshabadi

COMP3221 lec05-numbers-I.31

Saeid Nooshabadi

Two's complement shortcut: Negation

- Invert every 0 to 1 and every 1 to 0, then add 1 to the result

- Sum of number and its inverted representation must be
 $111\dots111_{\text{two}} + 000011_{\text{two}} = -1_{\text{ten}}$
- $111\dots111_{\text{two}} = -1_{\text{ten}}$
- Let x' mean the inverted representation of x
- Then $x + x' = -1 \Rightarrow x + x' + 1 = 0 \Rightarrow x' + 1 = -x$

- Example: -4 to +4 to -4

X :	two
X' :	two
+1 :	two
(-) :	two
+1 :	two

COMP3221 lec05-numbers-I.33

Saeid Nooshabadi

Two's complement shortcut: Negation

- Another Example: 20 to -20 to +20

X :	two
X' :	two
+1 :	two
(-) :	two
+1 :	two

COMP3221 lec05-numbers-I.34

Saeid Nooshabadi

And in Conclusion...

- We represent “things” in computers as particular bit patterns: N bits $\Rightarrow 2^N$
 - numbers, characters, ... (data)
- Decimal for human calculations, binary to understand computers, hex to understand binary
- 2's complement universal in computing: cannot avoid, so learn
- Computer operations on the representation correspond to real operations on the real thing
- Overflow: numbers infinite but computers finite, so errors occur

COMP3221 lec05-numbers-I.35

Saeid Nooshabadi