
ELEC2041
Microprocessors and Interfacing

Lecture 7: Number Systems - II

<http://webct.edtec.unsw.edu.au/>

March 2006

Saeid Nooshabadi

Saeid@unsw.edu.au

Overview

- Signed Numbers: 2' Complement representation
- Addition, Subtraction and Comparison in 2's Complement
- Comparison in signed and unsigned numbers
- Condition code flags
- Characters and Strings

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

Review: Two's Complement Formula

◦ Recognizing role of sign bit, can represent positive and negative numbers in terms of the bit value times a power of 2:

$$\bullet d_{31} \times (-2^{31}) + d_{30} \times 2^{30} + \dots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$$

◦ Example

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

Review: 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

($111\dots111_{\text{two}} = -1_{\text{ten}}$)  000011

- Let x' mean the inverted representation of x

+111100

- Then $x + x' = -1 \Rightarrow x + x' + 1 = 0 \Rightarrow x' + 1 = -x$

111111

◦ Example: -4 to +4 to -4

x : 1111 1111 1111 1111 1111 1111 1111 1100_{two}

x' : 0000 0000 0000 0000 0000 0000 0000 0011_{two}

+1: 0000 0000 0000 0000 0000 0000 0000 0100_{two}

($\)'$: 1111 1111 1111 1111 1111 1111 1111 1011_{two}

+1: 1111 1111 1111 1111 1111 1111 1111 1100_{two}

Two's Complement's Arithmetic Example

◦ Example: $20 - 4 = 16$

◦ $20 - 4 == 20 + (-4)$

x : 0000 0000 0000 0000 0000 0000 0001 0100_{two} -

Y : 0000 0000 0000 0000 0000 0000 0000 0100_{two}

x : 0000 0000 0000 0000 0000 0000 0001 0100_{two} +

- Y : 1111 1111 1111 1111 1111 1111 1111 1100_{two} -

$X + (-Y)$: 0000 0000 0000 0000 0000 0000 0001 0000_{two}

1

Ignore Carry out

Two's Complement's Arithmetic Example

◦ Example: $-2,147,483,648 - 2 = -2,147,483,650$?

x : 1000 0000 0000 0000 0000 0000 0000 0000_{two} -

Y : 0000 0000 0000 0000 0000 0000 0000 0010_{two}

x : 1000 0000 0000 0000 0000 0000 0000 0000_{two} +

- Y : 1111 1111 1111 1111 1111 1111 1111 1110_{two} -

$X + (-Y)$: 0111 1111 1111 1111 1111 1111 1111 1110_{two}

1

OVERFLOW

0111 1111 1111 1111 1111 1111 1111 1110_{two}

= 2,147,483,646_{ten}

≠ -2,147,483,650_{ten}

Signed vs. Unsigned Numbers

◦ C declaration `int`

- Declares a signed number
- Uses two's complement

◦ C declaration `unsigned int`

- Declares an unsigned number
- Treats 32-bit number as unsigned integer, so most significant bit is part of the number, not a sign bit

◦ NOTE:

- Hardware does all arithmetic in 2's complement.
- It is up to programmer to interpret numbers as signed or unsigned.
- Hardware provide some information to interpret numbers as signed or unsigned (check Slides 11-13!)

Overflow for Two's Complement Numbers?

- Adding (or subtracting) 2 32-bit numbers can yield a result that needs 33 bits
 - sign bit set with **value** of result instead of proper **sign** of result
 - since need just 1 extra bit, only sign bit can be wrong

Overflow Conditions

Op	A	B	Result
A + B	>=0	>=0	<0
A + B	<0	<0	>=0
A - B	>=0	<0	<0
A - B	<0	>=0	>=0

- When adding operands with different signs (subtracting with same signs) Overflow cannot occur

Signed vs. Unsigned Comparisons

- X = 1111 1111 1111 1111 1111 1111 1111 1100_{two}
- Y = 0011 1011 1001 1010 1000 1010 0000 0000_{two}

◦ Is X > Y?

- unsigned: YES
- signed: NO

◦ Converting to decimal to check

- Signed comparison:
-4_{ten} < 1,000,000,000_{ten}?
- Unsigned comparison:
4,294,967,292_{ten} > 1,000,000,000_{ten}?

Signed v. Unsigned Comparisons (Hardware Help)

- X = 1111 1111 1111 1111 1111 1111 1111 1100_{two}
- Y = 0011 1011 1001 1010 1000 1010 0000 0000_{two}
- Is X > Y? Do the Subtraction X - Y and check result

$$\begin{array}{r}
 X = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two} \quad - \\
 Y = 0011\ 1011\ 1001\ 1010\ 1000\ 1010\ 0000\ 0000_{two} \\
 \hline
 X = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two} \quad + \\
 -Y = 1100\ 0100\ 0110\ 0101\ 0111\ 0110\ 0000\ 0000_{two} \quad \text{-Y in 2's complement} \\
 \hline
 R = 1100\ 0100\ 0110\ 0101\ 0111\ 0101\ 1111\ 1100_{two} \quad \text{Result in 2's complement}
 \end{array}$$

Carry out

1 -ve sign indicates X is NOT greater than Y

1 carry out indicates X is greater than Y if numbers were unsigned.

Status Flags in Program Status Register CPSR



Copies of the ALU status flags (latched for some instructions).

* **Condition Code Flags**

- N = Negative result from ALU flag.
- Z = Zero result from ALU flag.
- C = ALU operation Carried out
- V = ALU operation oVerflowed (carry into the msb ≠ carry out of msb)

ARM Terminology:

- GT (Greater) X > Y (signed Arithmetic)
- HI (Higher) X > Y (unsigned Arithmetic)

Condition Flags

Flags	Arithmetic Instruction
Negative (N='1')	Bit 31 of the result has been set Indicates a negative number in signed operations
Zero (Z='1')	Result of operation was zero
Carry (C='1')	Result was greater than 32 bits
Overflow (V='1')	Result was greater than 31 bits Indicates a possible corruption of the sign bit in signed numbers

ELEC2041 lec07-numbers-II.13

“What’s This Stuff Good For?”

- **Bow-Lingual Dog Translator coming:** Dog lovers can finally discover what their dogs were saying with a US\$120 gadget called **Bowlingual**, a device that translates a dog's barks into words.



Home Alone Mode records your dog's emotions while you're out at work all day. Also includes a Body Language Translation Mode, a Training Mode and a Medical Reference Mode. Uses 5 "AAA" batteries, included.



Although inventions such as this may seem unnecessary and frivolous, they're often based on serious science. For example, **Bowlingual** uses voiceprint technology to create digital representations of sounds.

What inventions, whether serious or silly, would you like to see in the next 10 years?

<http://www.cnn.com/2003/TECH/biztech/03/24/tech.dogs.language.reut/>

ELEC2041 lec07-numbers-II.14

Saeid Nooshabadi

Kilo, Mega, Giga, Tera, Peta, Exa, Zetta, Yotta

physics.nist.gov/cuu/Units/binary.html

- **Common use prefixes (all SI, except K [= k in SI])**

Name	Abbr	Factor	SI size
Kilo	K	2 ¹⁰ = 1,024	10 ³ = 1,000
Mega	M	2 ²⁰ = 1,048,576	10 ⁶ = 1,000,000
Giga	G	2 ³⁰ = 1,073,741,824	10 ⁹ = 1,000,000,000
Tera	T	2 ⁴⁰ = 1,099,511,627,776	10 ¹² = 1,000,000,000,000
Peta	P	2 ⁵⁰ = 1,125,899,906,842,624	10 ¹⁵ = 1,000,000,000,000,000
Exa	E	2 ⁶⁰ = 1,152,921,504,606,846,976	10 ¹⁸ = 1,000,000,000,000,000,000
Zetta	Z	2 ⁷⁰ = 1,180,591,620,717,411,303,424	10 ²¹ = 1,000,000,000,000,000,000,000
Yotta	Y	2 ⁸⁰ = 1,208,925,819,614,629,174,706,176	10 ²⁴ = 1,000,000,000,000,000,000,000,000

- **Confusing!** Common usage of “kilobyte” means 1024 bytes, but the “correct” SI value is 1000 bytes
- **Hard Disk manufacturers & Telecommunications** are the only computing groups that use SI factors, so what is advertised as a 30 GB drive will actually only hold about 28 x 2³⁰ bytes, and a 1 Mbit/s connection transfers 10⁶ bps.

ELEC2041 lec07-numbers-II.15

Saeid Nooshabadi

kibi, mebi, gibi, tebi, pebi, exbi, zebe, yobi

en.wikipedia.org/wiki/Binary_prefix

- **New IEC Standard Prefixes [only to exbi officially]**

Name	Abbr	Factor
kibi	Ki	2 ¹⁰ = 1,024
mebi	Mi	2 ²⁰ = 1,048,576
gibi	Gi	2 ³⁰ = 1,073,741,824
tebi	Ti	2 ⁴⁰ = 1,099,511,627,776
pebi	Pi	2 ⁵⁰ = 1,125,899,906,842,624
exbi	Ei	2 ⁶⁰ = 1,152,921,504,606,846,976
zebi	Zi	2 ⁷⁰ = 1,180,591,620,717,411,303,424
yobi	Yi	2 ⁸⁰ = 1,208,925,819,614,629,174,706,176

As of this writing, this proposal has yet to gain widespread use...

- **International Electrotechnical Commission (IEC) in 1999 introduced these to specify binary quantities.**
 - Names come from shortened versions of the original SI prefixes (same pronunciation) and *bi* is short for “binary”, but pronounced “bee” :-)
 - Now SI prefixes only have their base-10 meaning and never have a base-2 meaning.

ELEC2041 lec07-numbers-II.16

Saeid Nooshabadi

The way to remember #s

How much is 2^{34} Bytes? Answer! 2^{XY} means... $2^Y \times 2^X$

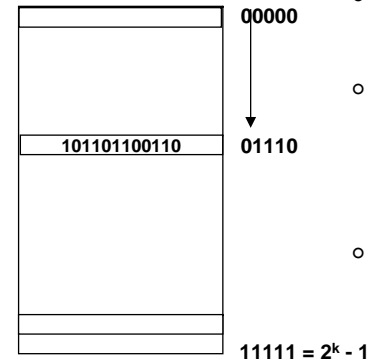
X=0 ⇒ ---	Y=0 ⇒ 1
X=1 ⇒ kibi = $2^{10} \sim 10^3$	Y=1 ⇒ 2
X=2 ⇒ mebi = $2^{20} \sim 10^6$	Y=2 ⇒ 4
X=3 ⇒ gibi = $2^{30} \sim 10^9$	Y=3 ⇒ 8
X=4 ⇒ tebi = $2^{40} \sim 10^{12}$	Y=4 ⇒ 16
X=5 ⇒ pebi = $2^{50} \sim 10^{15}$	Y=5 ⇒ 32
X=6 ⇒ exbi = $2^{60} \sim 10^{18}$	Y=6 ⇒ 64
X=7 ⇒ zebi = $2^{70} \sim 10^{21}$	Y=7 ⇒ 128
X=8 ⇒ yobi = $2^{80} \sim 10^{24}$	Y=8 ⇒ 256
	Y=9 ⇒ 512

2^{34} Bytes = $2^4 \times$ gibi Bytes = 16Gi Bytes

How many bits do we need for 2.5 Ti Bytes? (i.e., what's $\text{ceil } \log_2 = \lg$ of 2.5 Ti Bytes)

2.5 Ti Bytes → $\lg(2.5 \times \text{TI Bytes}) = \lg(2.5) + \lg(\text{TI})$
 = 2 + 40 = 42 bits

Numbers are stored at addresses



- Memory is a place to store bits
- A word is a fixed number of bits (eg, 32) at an address
 - also fixed no. of bits
- Addresses are naturally represented as unsigned numbers

Sign Extension

Consider:

1111 = -1 in 4-bit representation

1111 1111 = -1 in 8-bit representation

1111 1111 1111 1111 = -1 in 16-bit representation

2's comp. negative number has infinite 1s

0111 = 7 in 4-bit representation

0000 0111 = 7 in 8-bit representation

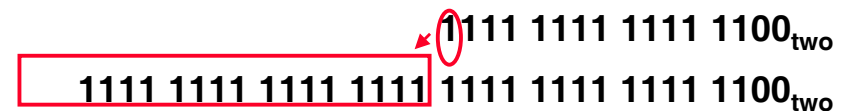
0000 0000 0000 0111 = 7 in 16-bit representation

• 2's comp. positive number has infinite 0s

Bit representation hides leading bits

Two's comp. shortcut: Sign extension

- Convert 2's complement number using n bits to more than n bits
- Simply replicate the most significant bit (sign bit) of smaller to fill new bits
 - 2's comp. positive number has infinite 0s
 - 2's comp. negative number has infinite 1s
 - Bit representation hides leading bits; sign extension restores some of them
 - 16-bit -4_{ten} to 32-bit:



Beyond Integers (Characters)

- 8-bit bytes represent characters, nearly every computer uses American Standard Code for Information Interchange (ASCII)

No. char	No. char	No. char	No. char	No. char	No. char
32	48 0	64 @	80 P	96 `	112 p
33	49 1	65 A	81 Q	97 a	113 q
34	50 2	66 B	82 R	98 b	114 r
35	51 3	67 C	83 S	99 c	115 s
...
47 /	63 ?	79 O	95 _	111 o	127 DEL

- Uppercase + 32 = Lowercase (e.g, B+32=b)
- tab=9, carriage return=13, backspace=8, Null=0

(Table in CD-ROM)

Strings

- Characters normally combined into strings, which have variable length
 - e.g., "Cal", "M.A.D", "ELEC2041"
- How represent a variable length string?
 - 1st position of string reserved for length of string (Pascal)
 - an accompanying variable has the length of string (as in a structure)
 - last position of string is indicated by a character used to mark end of string (C)
- C uses 0 (Null in ASCII) to mark end of string

Example String

- How many bytes to represent string "Popa"?
- What are values of the bytes for "Popa"?

No. char	No. char	No. char	No. char	No. char	No. char
32	48 0	64 @	80 P	96 `	112 p
33	49 1	65 A	81 Q	97 a	113 q
34	50 2	66 B	82 R	98 b	114 r
35	51 3	67 C	83 S	99 c	115 s
...
47 /	63 ?	79 O	95 _	111 o	127 DEL

- 80, 111, 112, 97, 0 DEC
- 50, 6F, 70, 61, 0 HEX

Strings in C: Example

- String simply an array of char


```
void strcpy (char x[], char y[]){
int i = 0; /* declare, initialize i*/

while ((x[i] = y[i]) != '\0') /* 0 */
    i = i + 1; /* copy and test byte */
}
```

What about non-Roman Alphabet?

◦ **Unicode**, universal encoding of the characters of most human languages

- Java uses Unicode
- needs 16 bits to represent a character
- 16-bits called **half word** in ARM

ASCII v. Binary

◦ Why not ASCII computers vs. binary computers?

- Harder to build hardware for add, subtract, multiply, divide
- Memory space to store numbers

◦ How many bytes to represent 1 billion?

◦ ASCII: "1000000000" => 11 bytes

◦ Binary: 0011 1011 1001 1010 1000 0000 0000 0000
=> 4 bytes

◦ up to 11/4 or almost 3X expansion of data size

What else is useful to represent?

◦ Numbers, Characters, logicals, ...

◦ Addresses

◦ Commands (operations)

• example:

- 0 => clap your hands
- 1 => snap your fingers
- 2 => slap your hands down

• execute: 1 0 2 0 1 0 2 0 1 0 2 0 1 0 2 0

• another example

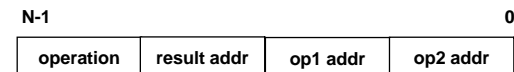
- 0 => add
- 1 => subtract
- 2 => compare
- 3 => multiply

How can we represent a machine instruction?

◦ Some bits for the operation

◦ Some bits for the address of each operand

◦ Some bits for the address of the result



$d = x + y$

add $d \ x \ y$

◦ Where could we put these things called instructions?

