ELEC2041
Microprocessors and Interfacing

Lecture 7: Number Systems - II

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

March 2005

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:
  - $d_{31} \times -2^{31} + d_{30} \times 2^{30} + \ldots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$
- Example

  $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$

  $= 1x\text{-}2^{31} + 1x2^{30} + 1x2^{29} + \ldots + 1x2^2 + 0x2^1 + 0x2^0$

  $= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^2 + 0 + 0$

  $= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten}$

  $= -4_{ten}$

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 ($111...111_{two} = -1_{ten}$)
  - Let x′ mean the inverted representation of x
  - Then x + x′ = -1 => x + x′ + 1 = 0 => x′ + 1 = -x

```
 000011
+111100
--------
 111111
```

- Example: -4 to +4 to -4

x : [redacted]$_{two}$
x′: [redacted]$_{two}$
+1: [redacted]$_{two}$
()′: [redacted]$_{two}$
+1: [redacted]$_{two}$

Two's Complement's Arithmetic Example

- Example: 20 – 4 = 16
- 20 – 4 == 20 + (–4)

[redacted]0100$_{two}$ –
[redacted]0100$_{two}$

x: [redacted]$_{two}$ +
–Y: [redacted]$_{two}$
X + (–Y): [redacted]$_{two}$

[redacted]

Ignore Carry out

Two's Complement's Arithmetic Example

- Example: –2,147,483,648 –2 = –2,147,483,650 ?

x : [redacted]$_{two}$ –
Y: [redacted]$_{two}$

x: [redacted]$_{two}$ +
–Y: [redacted]$_{two}$
X + (–Y): [redacted]$_{two}$

[redacted] 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

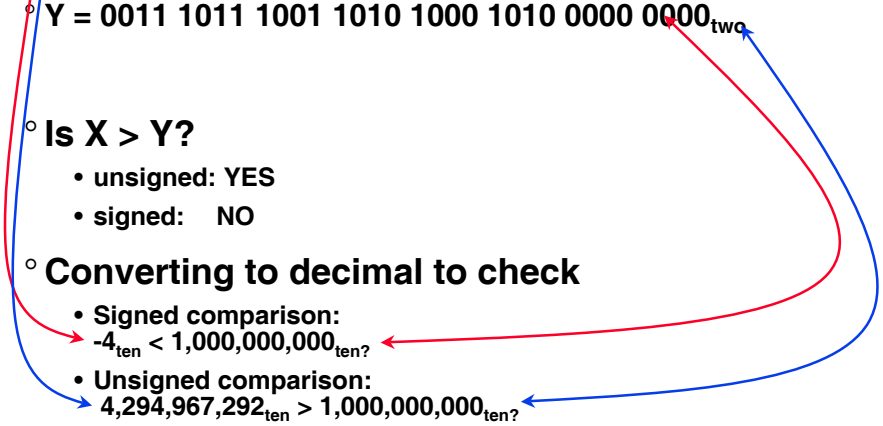
Overflow Conditions

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

$X = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$ –

$Y = 0011\ 1011\ 1001\ 1010\ 1000\ 1010\ 0000\ 0000_{two}$

$X = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$ +

– $Y = 1100\ 0100\ 0110\ 0101\ 0111\ 0110\ 0000\ 0000_{two}$ – Y in 2's complement

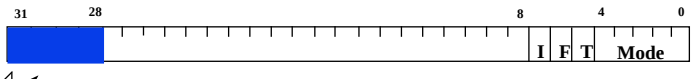
$R = 1100\ 0100\ 0110\ 0101\ 0111\ 0101\ 1111\ 1100_{two}$ Result in 2's complement

1 Carry out

-ve sign indicates X is NOT greater than Y

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**

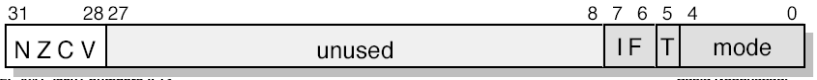
N = **N**egative result from ALU flag.
Z = **Z**ero result from ALU flag.
C = ALU operation **C**arried out
V = ALU operation o**V**erflowed (carry into the msb ≠ carry out of msb)

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



"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/

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 2^{30} bytes, and a 1 Mbit/s connection transfers 10^{6} bps.**

kibi, mebi, gibi, tebi, pebi, exbi, zebi, 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.**

The way to remember #s

° How much is 2^{34} Bytes? Answer! 2^{XY} means… 2^{Y} X 2^{x}

X=0 ⇒ ---
X=1 ⇒ kibi = 2^{10} ~10^{3}
X=2 ⇒ mebi = 2^{20} ~10^{6}
X=3 ⇒ gibi = 2^{30} ~10^{9}
X=4 ⇒ tebi = 2^{40} ~10^{12}
X=5 ⇒ pebi = 2^{50} ~10^{15}
X=6 ⇒ exbi = 2^{60} ~10^{18}
X=7 ⇒ zebi = 2^{70} ~10^{21}
X=8 ⇒ yobi = 2^{80} ~10^{24}

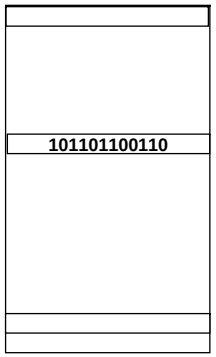
Y=0 ⇒ 1
Y=1 ⇒ 2
Y=2 ⇒ 4
Y=3 ⇒ 8
Y=4 ⇒ 16
Y=5 ⇒ 32
Y=6 ⇒ 64
Y=7 ⇒ 128
Y=8 ⇒ 256
Y=9 ⇒ 512

2^{34} Bytes = 2^{4} x gibi Bytes = 16GI Bytes

°How many bits do me need for 2.5 Ti Bytes? (i.e., what's `ceil` $\log_2$ = `lg` of 2.5 Ti Bytes)

2.5 Ti Bytes ➔ lg(2.5 x TI Bytes) = lg(2.5) + lg (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:

$1111\ 1111\ 1111\ 1100_{two}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$

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?
  1) 1st position of string reserved for length of string (Pascal)
  2) an accompanying variable has the length of string (as in a structure)
  3) 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

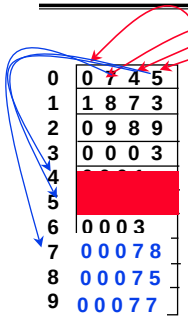
How can we represent a machine instruction?

- 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?

The Stored Program Computer



- Memory holds instructions and data as bits
- Instructions are fetched from memory and executed
  - operands fetched, manipulated, and stored
- Example 4-digit Instruction

operation	result addr	op1 addr	op2 addr

  - operation: 0 => add, 1 => sub
  - result address
  - op1 address
  - op2 address

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

- Example Data
  - 4 digit unsigned value
- What's in memory after executing 0,1,2?

So what's it all mean?

- We can write a program that will translate strings of 'characters' into 'computer instructions'
  - called a compiler or an assembler
- We can load these particular bits into the computer and execute them.
  - may manipulate numbers, characters, pixels... (application)
  - may translate strings to instructions (compiler)
  - may load and run more programs (operating system)

To remember

- We represent "things" in computers as particular bit patterns
  - numbers, characters, ... (data)
    - base, digits, positional notation
    - unsigned, 2s complement, 1s complement
  - addresses (where to find it)
  - instructions (what to do)
- Computer operations on the representation correspond to real operations on the real thing
  - representation of 2 plus representation of 3 = representation of 5
- two big ideas already!
  - Pliable Data: a program determines what it is
  - Stored program concept: instructions are just data

And in Conclusion...

- Computers do arithmetic in 2's complement
- Interpretation of numbers can be signed or unsigned
- The Arithmetic operation results should be interpreted depending the signed or unsigned interpretation of the operands.