COMP 3221

Microprocessors and Embedded Systems

Lecture 6: Number Systems - II

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

August, 2003

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

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

0000 ... 0000 0000 0000 0000_{two}	=	0_{ten}
0000 ... 0000 0000 0000 0001_{two}	=	1_{ten}
0000 ... 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 $-2{,}147{,}483{,}648_{ten}$

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}

  $= 1 \times -2^{31} + 1 \times 2^{30} + 1 \times 2^{29} + \ldots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

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

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

  $= -4_{ten}$

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 $111...111_{two}$
  - $111...111_{two} = -1_{ten}$
  - 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 a 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 17-20!)

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

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 v. 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

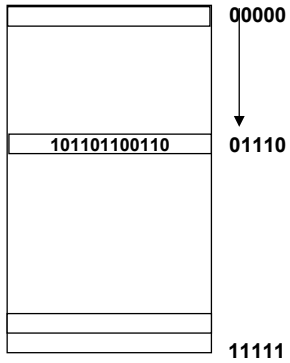
R= 1100 0100 0110 0101 0111 0101 1111 1100_{two} Result in 2's complement

1 Carry out

-ve sign indicates X is NOT greater than Y

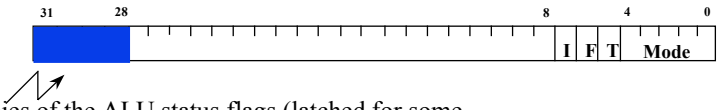
-ve sign and carry out indicates X is greater than Y if numbers were unsigned.

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

Status Flags in Program Status Register CPSR



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

* **Condition Code Flags**

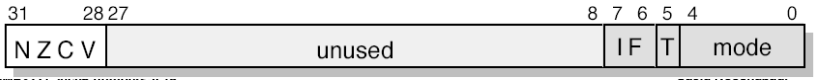
N = **N**egative result from ALU flag.
Z = **Z**ero result from ALU flag.
C = ALU operation **C**arried out
V = ALU operation o**V**erflowed (carry into the msb ≠ carry out of msb)

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



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.