Computer Organization

Boolean Arithmetic

Boolean Arithmetic

- As seen in the diagram of the ALU, we need to create additional ways to perform operations on binary numbers
  - subtraction
  - multiplication
- Addition is the foundation for all arithmetic operations, so we can use it to create the others

Addition

$$\begin{array}{ccccc} 0 & 0 & 1 & 0 & \\ & 1 & 0 & 1 & 0 \\ + & & & 1 & 1 \\ \hline & 1 & 1 & 0 & 1 \end{array}$$

Binary addition

$$\begin{array}{ccccc} 0 & 1 & 1 & 0 & \\ & 7 & 8 & 7 & 5 \\ + & & 5 & 6 & 2 \\ \hline & 8 & 4 & 3 & 7 \end{array}$$

Decimal addition

Addition – cont.

- Computers represent integers using a fixed number of bits, sometimes called "word size"

	0	0	1	0	
		1	0	1	0
+		0	0	1	1
		1	1	0	1

Binary addition

	0	0	1	
	0	0	0	1
+	0	1	0	1
	0	1	1	0

Another example

	1	1	1	0	
		0	1	1	1
+		1	1	1	0
	1	0	1	0	1

Overflow

Addition – cont.

	0	⋯	0	0	0	0	0	1	1	0	1	1	1	0	0	0
	0	...	0	0	0	0	0	0	1	1	0	1	0	1	0	1
+	0	...	0	0	0	0	0	0	0	1	0	1	1	1	0	0
	0	...	0	0	0	0	0	1	0	0	1	1	0	0	0	1

Same addition algorithm for any n

Signed & Unsigned Numbers

- Many programming languages initiate numbers using specific data types
  - 16 bit - short
  - 32 bit - integer
  - 64 bit - long
- These data types can be set as signed or unsigned

Signed & Unsigned Numbers – cont.

- A data type set to a specific word number (how many bits it represents) has a specific range
- Signed data types have a range of $-2^{n-1}, \ldots, -1, 0, 1, \ldots, 2^{n-1} - 1$
  - Negative, 0, Positive
- Unsigned data types have a range of $0 \ldots 2^{n} - 1$
  - 0, Positive (strictly non-negative)

Unsigned Numbers

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	10
1011	11	11
1100	12	12
1101	13	13
1110	14	14
1111	15	15

- Ex. - word size: $n = 4$
- Represents the full capacity of bit combinations (0 ... 15)
- Strictly non-negative values

Signed Numbers – Incorrect

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	– 0
1001	9	– 1
1010	10	– 2
1011	11	– 3
1100	12	– 4
1101	13	– 5
1110	14	– 6
1111	15	– 7

- Ex. - word size: $n = 4$
- Represents half the full capacity of bit combinations
  - (-7, …, -1, -0, 0, 1, …, 7)
  - Most Significant Bit (left-most) dictates polarity (sign)
- Contains -0 (we would have to revise our hardware to handle this)
  - Should NOT be used

Signed Numbers – Correct

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	−8
1001	9	−7
1010	10	−6
1011	11	−5
1100	12	−4
1101	13	−3
1110	14	−2
1111	15	−1

- Ex. - word size: $n = 4$
- Represents half the full capacity of bit combinations
  - (-8, ..., -1, 0, 1, ..., 7)
- Perfectly compatible with existing hardware that assumes unsigned data
  - Does not contain a -0

Signed Numbers – Correct

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	– 8
1001	9	– 7
1010	10	– 6
1011	11	– 5
1100	12	– 4
1101	13	– 3
1110	14	– 2
1111	15	– 1

The representation

- Assumption: Word size = n bits
- The "two's complement" of x is defined to be $2^n - x$
- The negative of x is coded by the two's complement of x

From decimal to binary:

if $x \geq 0$ return *binary*(x)

else return *binary*($2^n - x$)

From binary to decimal:

if MSB = 0 return *decimal*(*bits*)

else return "–" and then (2^n – *decimal*(*bits*))

Two's Complement – Addition

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	–8
1001	9	–7
1010	10	–6
1011	11	–5
1100	12	–4
1101	13	–3
1110	14	–2
1111	15	–1

Algorithm: Regular addition, modulo 2^n

$$\begin{array}{r} 6 \\ + \\ -2 \end{array} = \begin{array}{r} 6 \\ + \\ 14 \\ \hline 20 \end{array} \% \ 16 = 4 \quad \text{codes} \ 4$$

$$\begin{array}{r} 3 \\ + \\ -5 \end{array} = \begin{array}{r} 3 \\ + \\ 11 \\ \hline 14 \end{array} \% \ 16 = 14 \quad \text{codes} \ -2$$

$$\begin{array}{r} -2 \\ + \\ -5 \end{array} = \begin{array}{r} 14 \\ + \\ 11 \\ \hline 25 \end{array} \% \ 16 = 9 \quad \text{codes} \ -7$$

Two's Complement – Addition

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	– 8
1001	9	– 7
1010	10	– 6
1011	11	– 5
1100	12	– 4
1101	13	– 3
1110	14	– 2
1111	15	– 1

$$\begin{array}{r} 4 \\ + \; -7 \end{array} = \; ?$$

$$\begin{array}{r} -2 \\ + \; -4 \end{array} = \; ?$$

Two's Complement – Addition

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	−8
1001	9	−7
1010	10	−6
1011	11	−5
1100	12	−4
1101	13	−3
1110	14	−2
1111	15	−1

$$\begin{array}{r} 4 \\ + \quad \\ -7 \end{array} \quad = \quad \begin{array}{r} 4 \\ + \quad \\ 9 \\ \hline \end{array}$$

$13 \% 16 = 13$ codes -3

$$\begin{array}{r} -2 \\ + \quad \\ -4 \end{array} \quad = \quad \begin{array}{r} 14 \\ + \quad \\ 12 \\ \hline \end{array}$$

$26 \% 16 = 10$ codes -6

Two's Complement – Addition

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	– 8
1001	9	– 7
1010	10	– 6
1011	11	– 5
1100	12	– 4
1101	13	– 3
1110	14	– 2
1111	15	– 1

At the binary level (same algorithm):

$$\begin{array}{r} 6 \\ + \;-2 \end{array} \quad = \quad \begin{array}{r} 0110 \\ +\;1110 \\ \hline \cancel{1}0100 \end{array} \quad \text{codes } 4$$

Ignoring the overflow bit is the binary equivalent of modulo 2^n

$$\begin{array}{r} 3 \\ + \;-5 \end{array} \quad = \quad \begin{array}{r} 0011 \\ +\;1011 \\ \hline 1110 \end{array} \quad \text{codes } -2$$

$$\begin{array}{r} -2 \\ + \;-5 \end{array} \quad = \quad \begin{array}{r} 1110 \\ +\;1011 \\ \hline \cancel{1}1001 \end{array} \quad \text{codes } -7$$

Two's Complement – Addition

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	– 8
1001	9	– 7
1010	10	– 6
1011	11	– 5
1100	12	– 4
1101	13	– 3
1110	14	– 2
1111	15	– 1

<u>At the binary level</u> (same algorithm):

$$\begin{array}{r} 6 \\ + \;\; -2 \end{array} \;=\; \begin{array}{r} 0110 \\ +\;\; 1110 \\ \hline \cancel{1}0100 \end{array} \quad \text{codes } 4$$

More examples:

$$\begin{array}{r} 5 \\ + \;\; 7 \end{array} \;=\; \begin{array}{r} 0101 \\ +\;\; 0111 \\ \hline 1100 \end{array} \quad \text{codes } -4 \qquad 5 + 7 = -4 \qquad ???$$

$$\begin{array}{r} -7 \\ + \;\; -3 \end{array} \;=\; \begin{array}{r} 1001 \\ +\;\; 1101 \\ \hline \cancel{1}0110 \end{array} \quad \text{codes } 6 \qquad -7 + -3 = 6 \qquad ???$$

Two's Complement – Subtraction

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	– 8
1001	9	– 7
1010	10	– 6
1011	11	– 5
1100	12	– 4
1101	13	– 3
1110	14	– 2
1111	15	– 1

- Simply consider that $x - y \equiv x + (-y)$
- We just need to convert our second number and do simple addition
- But, how do we convert the polarity of a number?

Two's Complement – Conversion

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	– 8
1001	9	– 7
1010	10	– 6
1011	11	– 5
1100	12	– 4
1101	13	– 3
1110	14	– 2
1111	15	– 1

Insight: $$\begin{aligned} code(-x) = (2^n - x) &= 1 + (2^n - 1) - x \\ &= 1 + (\texttt{1111}) - x \\ &= 1 + flippedBits(x) \end{aligned}$$

<u>Algorithm:</u> To convert *bbb...b*:

Flip all the bits and add 1 to the result

Example: Convert 0010 (2)

$$\begin{array}{rl} & \texttt{1101} \text{ (flipped)} \\ + & \texttt{1} \\ \hline & \texttt{1110} \text{ } (-2) \end{array}$$

Two's Complement – Conversion

code(x)		x
`0000`	0	0
`0001`	1	1
`0010`	2	2
`0011`	3	3
`0100`	4	4
`0101`	5	5
`0110`	6	6
`0111`	7	7
`1000`	8	– 8
`1001`	9	– 7
`1010`	10	– 6
`1011`	11	– 5
`1100`	12	– 4
`1101`	13	– 3
`1110`	14	– 2
`1111`	15	– 1

Insight: $$code(-x) = (2^n - x) = 1 + (2^n - 1) - x$$
$$= 1 + (\texttt{1111}) - x$$
$$= 1 + \mathit{flippedBits}(x)$$

<u>Algorithm:</u> To convert *bbb...b*:

Flip all the bits and add 1 to the result

Practice: Convert `1010` (–6)

Two's Complement – Conversion

code(x)		x
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	–8
1001	9	–7
1010	10	–6
1011	11	–5
1100	12	–4
1101	13	–3
1110	14	–2
1111	15	–1

Insight: $$\begin{aligned} code(-x) = (2^n - x) &= 1 + (2^n - 1) - x \\ &= 1 + (1111) - x \\ &= 1 + flippedBits(x) \end{aligned}$$

Algorithm: To convert *bbb...b*:

Flip all the bits and add 1 to the result

Practice: Convert 1010 (–6)

$$\begin{array}{rl} & 0101 \text{ (flipped)} \\ + & 1 \\ \hline & 0110 \text{ (6)} \end{array}$$