Computer Systems Architecture

http://cs.nott.ac.uk/~txa/g51csa/

Thorsten Altenkirch and Liyang Hu

School of Computer Science
University of Nottingham

Lecture 06: Binary Addition and Signed Numbers

The University of Nottingham

Long Addition

Long Addition in Decimal

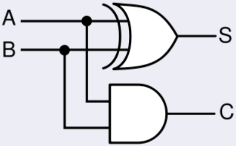
		6	2	9	5	1	4	1	3	
$+$		2	8	1	8	2	8	1	7	
	0	1	1	1	0	1	0	1		Carry
		9	1	1	3	4	2	3	0	

Long Addition in Binary

		0	1	1	1	0	1	1	0	$76_{16} = 118$
$+$		1	1	0	1	0	1	0	1	$D5_{16} = 213$
	1	1	1	1	0	1	0	0		Carry
	1	0	1	0	0	1	0	1	1	$14B_{16} = 331$

Single-Bit Adders

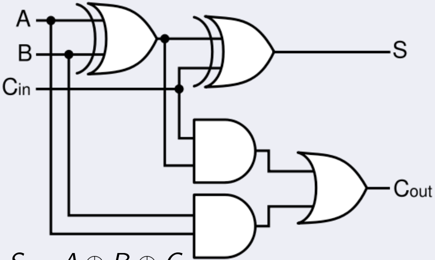
Half-Adder



$$S = A \oplus B$$
$$C = A \wedge B$$

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full-Adder



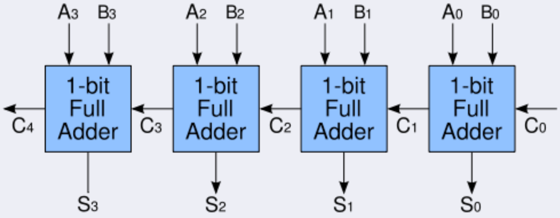
$$S = A \oplus B \oplus C_{\text{in}}$$
$$C_{\text{out}} = (A \wedge B) \vee (B \wedge C_{\text{in}}) \vee (A \wedge C_{\text{in}})$$

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Ripple Carry Adder

- A chain of full-adders can sum as many bits as needed
  - The C_{out} of each adder is wired to the C_{in} of the next
  - Present the n^{th} bit of inputs A and B to the n^{th} adder
  - Set C_0 to 0; carry out C_4 indicates overflow
  - Calculates $S = A + B$, where S_n is the n^{th} bit of S...
- The carry bits *ripples* across the full adders
- Simplest, but not the fastest (c.f. carry look-ahead)

4-Bit Ripple Carry Adder



Simulating addition in C

- Inputs x, y;

```
ci=0;
z=0;
for(i=0;i<sizeof(x)*8;i++) {
  xi=x&(1<<i);
  yi=y&(1<<i);
  zi=xi^yi^ci;
  csi=(xi&yi)|(yi&ci)|(xi&ci);
  z=z|zi;
  ci=csi;
}
```

- Output: z.
- For illustration only!
  Nobody would implement addition in software!

The University of Nottingham

Representing Negative Numbers

- How are negative numbers represented on a computer?
- Sign and Magnitude
  - What we use in decimal notation: $+/-$ and $0, 1, 2, \cdots$
- Ones' Complement
  - Represent $-x$ by its bitwise inverse (NOT)
- Excess-n
  - Zero represented by binary n: 0_2 is thus $-n$
- Two's Complement
  - For an n-bit number, represent $-x$ by $2^n - x$
- How do we add two integers in the above systems?

Example of 4-Bit Signed Encodings

Sign and Mag.

1111	−7
1110	−6
1101	−5
1100	−4
1011	−3
1010	−2
1001	−1
1000	−0
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7

Ones' Comp.

1000	−7
1001	−6
1010	−5
1011	−4
1100	−3
1101	−2
1110	−1
1111	−0
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7

Excess-3

0000	−3
0001	−2
0010	−1
0011	0
0100	+1
0101	+2
0110	+3
0111	+4
1000	+5
1001	+6
1010	+7
1011	+8
1100	+9
1101	+10
1110	+11
1111	+12

Two's Comp.

1000	−8
1001	−7
1010	−6
1011	−5
1100	−4
1101	−3
1110	−2
1111	−1
0000	0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7

Method of Complements

- In base b with n digits, represent $-x$ using $b^n - x$
- But there's a quicker way to calculate $b^n - x$:
  - Subtract each digit of x from $b - 1$, then add 1 to result
- Addition as for unsigned numbers; just ignore carry digit

Calculate $8192 - 4235$ Using Ten's Complement

- Negate: $\begin{array}{rr} & 9999 \\ - & 4235 \\ \hline & 5764 \\ + & 1 \\ \hline & 5765 \end{array}$, then add: $\begin{array}{rr} & 8192 \\ + & 5765 \\ \hline & 13957 \end{array}$
- Drop the carry digit: thus $8192 - 4235 = 3957$

Two's Complements

- In base 2 with n bits, represent $-x$ using $2^n - x$
- But there's a quicker way to calculate $2^n - x$:
  - Take the bitwise inverse (NOT) of x, then add 1 to result
- Addition as for unsigned numbers; just ignore carry bit
- Conventionally we take the MSB to represent the sign
  - 0 for positive, 1 for negative
- Thus n bits can represent from -2^{n-1} up to $2^{n-1} - 1$
- All modern processors primarily use two's complement
  - Includes all the MIPS's integer arithmetic instructions
  - Different instruction variants for signed and unsigned
  - But floating point (future lecture) uses *sign and magnitude* as well as *excess-n*

Alternative View

8-Bit Two's Complement ($-128 \leq x < 127$)

	MSB							LSB
Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Weight	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Exercises

- Calculate $81 - 42$ in 8-bit two's complement
- We can negate `$s0` using

```
nor $s0, $zero, $s0
addi $s0, $s0, 1
```

  For what number will this not work?

Thursday quiz

Most of the following questions are multiple choice. There is at least one correct choice but there may be several. For each of the questions list all the roman numerals corresponding to correct answers but none of the incorrect ones.

Questions are marked as follows:

no errors	5 points
1 error	3 points
2 errors	1 point
$\geq$**3 errors**	0 points

Thursday quiz

1. Which of the following statements about MIPS machine code are correct?
   a. MIPS machine code consists of symbolic instructions?
   b. Each instruction is 32 bytes long.
   c. There are 3 basic formats: R,I,J.
   d. The opcode always occupies the leftmost 6 bits.
   e. The position of the opcode varies with the instruction.

Thursday quiz

2. What do we mean by *data transfer instructions*?
   a. Instructions to load memory data into registers.
   b. Instructions to store regsiters into memory.
   c. Instructions to read data from a keyboard.
   d. Instructions to write data on the console.
   e. Instructions to transfer data over the internet.

Thursday quiz

2. What do we mean by *data transfer instructions*?
   a. Instructions to load memory data into registers.
   b. Instructions to store registers into memory.
   c. Instructions to read data from a keyboard.
   d. Instructions to write data on the console.
   e. Instructions to transfer data over the internet.

Thursday quiz

3. What does the following program fragment do?

```
num:     .word 0
         ...
         la $s0,num
         li $t0,1
         sb $t0,1($s0)
         lw $t1,($s0)
```

- a) It stores 1 in register $t1.
- b) It stores 256 in register $t1.
- c) It modifies a byte in memory.
- d) It divides the content of $t0 by 256.
- e) It multiplies the content of $t0 by 256.

Thursday quiz

4. What can we say about `goto`?

a. It is the recommended way to implement loops in high level languages.
b. Dikstra wrote a letter titled *Goto Statment considered useful*.
c. In machine language goto statments (or jumps) are used to implement control flow.
d. Java has a `goto` statement.
e. C++ has a `goto` statement.

Thursday quiz

5. What is `bge` in MIPS assembler code?

   a. A machine instruction.
   b. A pseudoinstruction.
   c. A conditional jump.
   d. An unconditional jump.
   e. A data transfer instruction.