Lec 4: Finite State Machines and Arithmetic

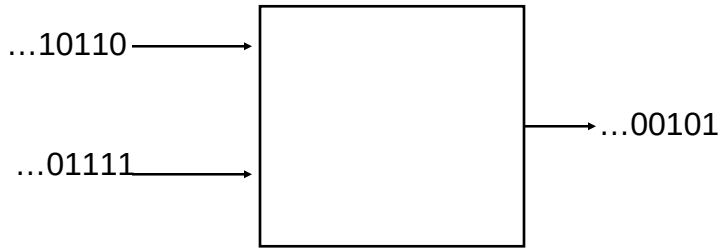
Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

References

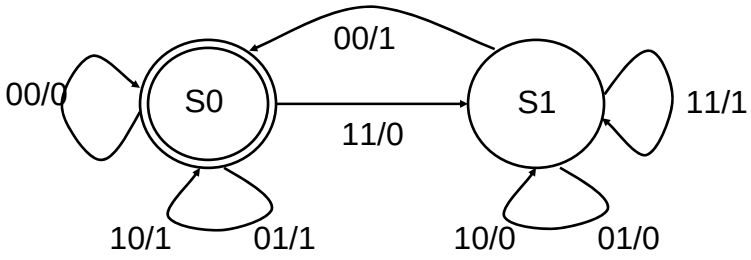
- Look at Appendix B in textbook
- Look in Chapter 3 in textbook for today

FSM: Serial Adder

- Add two input bit streams
  - streams are sent with least-significant-bit (lsb) first



FSM: State Diagram



- Two states: S0 (no carry), S1 (carry in)
- Inputs: a and b
- Output: z
  - Arcs labelled w/ input bits a and b, and output z

Serial Adder: State Table

a	b	state	z	next state
0	0	S0	0	S0
0	1	S0	1	S0
1	0	S0	1	S0
1	1	S0	0	S1
0	0	S1	1	S0
0	1	S1	0	S1
1	0	S1	0	S1
1	1	S1	1	S1

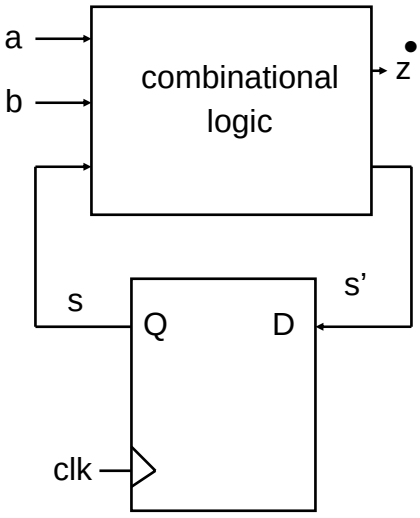
- Write down all input and state combinations

Serial Adder: State Assignment

a	b	s	z	s’
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

- Two states, so 1-bit is sufficient
  - A single flip-flop will encode the state

Serial Adder: Circuit



- Equations
  - $z = \bar{a}\bar{b}s + \bar{a}b\bar{s} + a\bar{b}\bar{s} + abs$
  - $s' = \bar{a}\bar{b}s + \bar{a}bs + ab\bar{s} + abs$

Recap

- We can build combinatorial circuits
  - Gates, minimization
- We can build stateful circuits
  - Record 1-bit values in latches and flip-flops
- Powerful combination
  - We can build real, useful devices
  - But we will often need to perform arithmetic

Binary Arithmetic

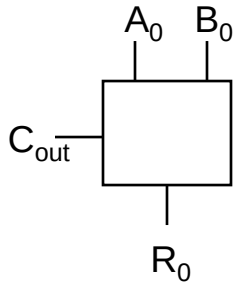
```
  12
+ 25
----
  37
```

- Arithmetic works the same way regardless of base
  - Add the digits in each position
  - Propagate the carry

```
  001100
+ 011010
--------
  100110
```

- Unsigned binary addition is pretty easy
  - Combine two bits at a time
  - Along with a carry

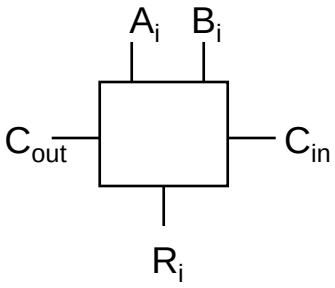
1-bit Adder



A_0	B_0	C_{out}	R_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Adds two 1-bit numbers, computes 1-bit result and carry out
- Useful for the rightmost binary digit, not much else

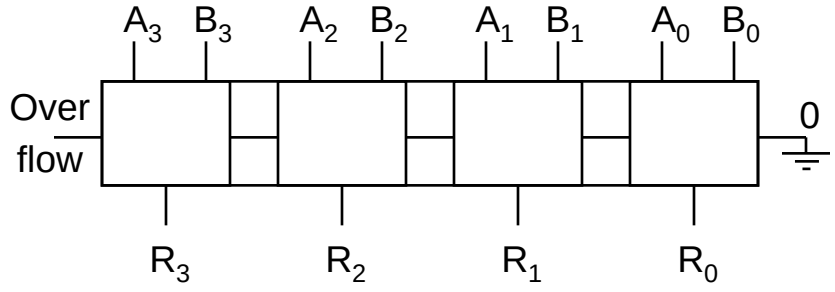
1-bit Adder with Carry



C_{in}	A_i	B_i	C_{out}	R_i
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

- Adds two 1-bit numbers, along with carry-in, computes 1-bit result and carry out
- Can be cascaded to add N-bit numbers

4-bit Adder



- Adds two 4-bit numbers, along with carry-in, computes 4-bit result and overflow
- Overflow indicates that the result does not fit in 4 bits

Two Questions

- Addition is very important
  - Is this design fast enough?
  - Also, is this design general enough?

Arithmetic with Negative Numbers

- Negative numbers complicate arithmetic
- Recall that for addition and subtraction, the rules are:
  - Both positive => add, result positive
  - One +, one - => subtract small number from larger one
  - Both negative => add, result negative

Arithmetic with Negative Numbers

- We could represent sign with an explicit bit
  - the “sign-magnitude form”
  - But arithmetic would be much easier to perform in hardware if we did not have to examine the operands’ signs
- Two’s complement representation enables arithmetic to be performed without examining the operands

Two’s Complement

- Nonnegative numbers are represented as usual
  - 0 = 0000
  - 1 = 0001
  - 3 = 0011
  - 7 = 0111
- To negate a number, flip all bits, add one
  - -1: 1 $\Rightarrow$ 0001 $\Rightarrow$ 1110 $\Rightarrow$ 1111
  - -3: 3 $\Rightarrow$ 0011 $\Rightarrow$ 1100 $\Rightarrow$ 1101
  - -7: 7 $\Rightarrow$ 0111 $\Rightarrow$ 1000 $\Rightarrow$ 1001
  - -8: 8 $\Rightarrow$ 1000 $\Rightarrow$ 0111 $\Rightarrow$ 1000
  - -0: 0 $\Rightarrow$ 0000 $\Rightarrow$ 1111 $\Rightarrow$ 0000 (this is good, -0 = +0)

Two's Complement Facts

- Negative numbers have a leading 1
  - Similar to signed magnitude form
  - Largest negative=1000…0
  - Largest positive=0111…1
  - Top most bit: sign bit
- N bits can be used to represent
  - unsigned: the range $0..2^N-1$
  - ex: 8 bits $\Rightarrow$ 0..255
  - two's complement: the range $-(2^{N-1})..(2^{N-1})-1$
  - ex: 8 bits $\Rightarrow$ (10000000)..(01111111)
  - $\Rightarrow$ -128..127

Want to change bit size of number?

- 4 bit to 8 bit
  - For positive number, just 0's in new 4 bits
  - For negative number, 1's in new 4 bits
- What about shifting
  - sll (shift left logical)
  - sra (shift right arithmetic)
  - srl (shift right logical)

Two's Complement Addition

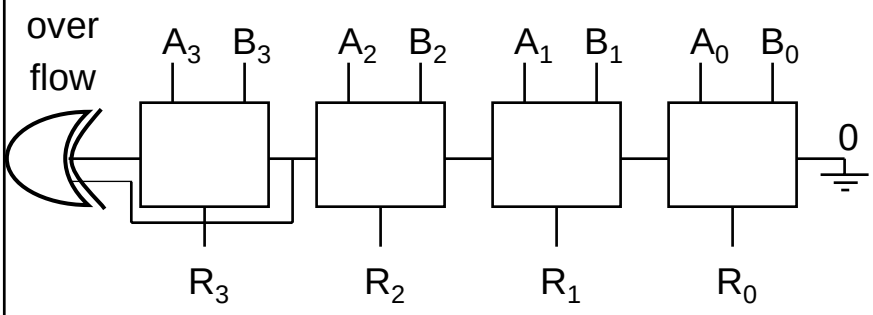
- Perform addition as usual, regardless of sign
  - 1 = 0001, 3 = 0011, 7 = 0111, 0 = 0000
  - -1 = 1111, -3 = 1101, -7 = 1001
- Examples
  - 1 + -1 = 1111 + 0001 = 0000 (0)
  - -3 + -1 = 1111 + 1101 = 1100 (-4)
  - -7 + 3 = 1001 + 0011 = 1100 (-4)
  - 7 + (-3) = 0111 + 1101 = 0100 (4)
  - 7 + 1, -7 + -3, -7 + -1

Overflow

- When can it occur?
  - If you add a negative and positive number
    - Cannot occur (Why?)
  - If you add two negatives or two positives
    - Can occur (Why?)
  - Add two positives, and get a negative number
  - Or, add two negatives, get a positive number
    - Overflow!
  - Overflow when
    - Carry into most significant bit (msb) != carry out of msb

Two's Complement Adder

- Let's build a two's complement adder



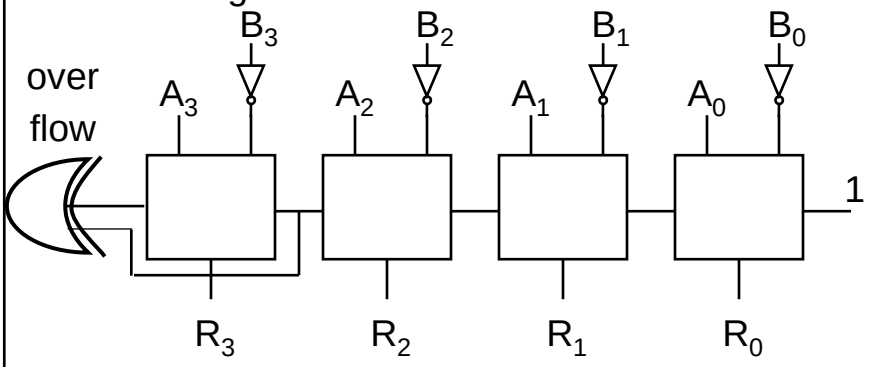
- Already built, just needed to modify overflow checking

Subtraction

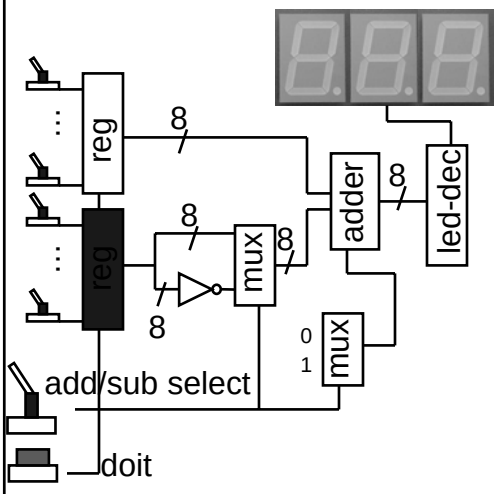
- Why create a new circuit?
- Just use addition
  - How?

Two's Complement Subtraction

- Subtraction is simply addition, where one of the operands has been negated
  - Negation is done by inverting all bits and adding one



A Calculator



- Enter numbers to be added or subtracted using toggle switches
- Select: ADD or SUBTRACT
- Muxes feed A and B,or A and –B, to the 8-bit adder
- The 8-bit decoder for the hex display is straightforward