Arithmetic Circuits

(Part I)

Randy H. Katz
University of California, Berkeley

Spring 2004

Motivation

Arithmetic circuits are excellent examples of comb. logic design

- *Time vs. Space Trade-offs*
  - Doing things fast requires more logic and thus more space
  - Example: carry lookahead logic
- *Arithmetic Logic Units*
  - Critical component of processor datapath
  - Inner-most "loop" of most computer instructions

Overview

- *Binary Number Representation*
  - Sign & Magnitude, Ones Complement, Twos Complement
- *Binary Addition*
  - Full Adder Revisted
- *ALU Design*
- *BCD Circuits*
- *Combinational Multiplier Circuit*
- *Design Case Study: 8 Bit Multiplier*
- *Sequential Multiplier Circuit*

Number Systems
Representation of Negative Numbers

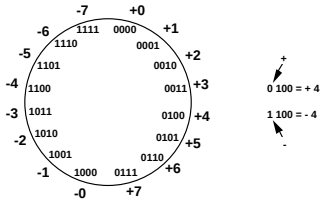
Representation of positive numbers same in most systems

Major differences are in how negative numbers are represented

Three major schemes:
- sign and magnitude
- ones complement
- twos complement

Assumptions:
- we'll assume a 4 bit machine word
- 16 different values can be represented
- roughly half are positive, half are negative

Number Systems
Sign and Magnitude Representation



High order bit is sign: 0 = positive (or zero), 1 = negative

Three low order bits is the magnitude: 0 (000) thru 7 (111)

Number range for n bits = $+/-2^{n-1} - 1$

Representations for 0

Number Systems
Sign and Magnitude

Cumbersome addition/subtraction

Must compare magnitudes to determine sign of result

Ones Complement

N is positive number, then $\overline{N}$ is its negative 1's complement

$\overline{N} = (2^n - 1) - N$

Example: 1's complement of 7

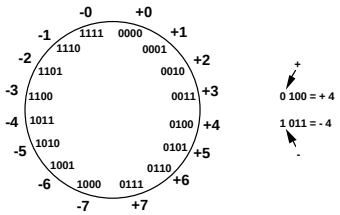
2^4 = 10000
-1 = 00001
1111
-7 = 0111
1000 = -7 in 1's comp.

Shortcut method:

simply compute bit wise complement

0111 -> 1000

Number Systems
Ones Complement



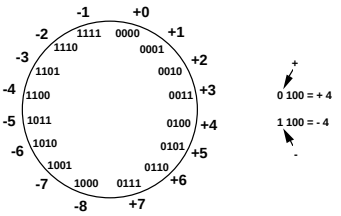
Subtraction implemented by addition & 1's complement

Still two representations of 0! This causes some problems

Some complexities in addition

Number Representations
Twos Complement

like 1's comp except shifted one position clockwise



Only one representation for 0

One more negative number than positive number

Number Systems
Twos Complement Numbers

$N^* = 2^n - N$

$2^4 = 10000$

Example: Twos complement of 7 — sub 7 = 0111

1001 = repr. of -7

Example: Twos complement of -7 — $2^4 = 10000$

sub -7 = 1001

0111 = repr. of 7

Shortcut method:

Twos complement = bitwise complement + 1

0111 -> 1000 + 1 -> 1001 (representation of -7)

1001 -> 0110 + 1 -> 0111 (representation of 7)

Number Representations
Addition and Subtraction of Numbers

Sign and Magnitude

result sign bit is the same as the operands' sign

4	0100	-4	1100	
+ 3	0011	+ (-3)	1011	
7	0111	-7	1111	

when signs differ, operation is subtract, sign of result depends on sign of number with the larger magnitude

4	0100	-4	1100
- 3	1011	+ 3	0011
1	0001	-1	1001

Number Systems
Addition and Subtraction of Numbers

Ones Complement Calculations

4	0100	-4	1011
+ 3	0011	+ (-3)	1100
7	0111	-7	10111
		End around carry	1
			1000

4	0100	-4	1011
- 3	1100	+ 3	0011
1	10000	-1	1110
End around carry	1		
	0001		

Number Systems
Addition and Subtraction of Binary Numbers

Ones Complement Calculations

Why does end-around carry work?

Its equivalent to subtracting 2^n and adding 1

$M - N = M + \overline{N} = M + (2^n - 1 - N) = (M - N) + 2^n - 1 \quad (M > N)$

$-M + (-N) = \overline{M} + \overline{N} = (2^n - M - 1) + (2^n - N - 1) = 2^n + [2^n - 1 - (M + N)] - 1 \quad M + N < 2^{n-1}$

after end around carry:

$= 2^n - 1 - (M + N)$

this is the correct form for representing -(M + N) in 1's comp!

Number Systems
Addition and Subtraction of Binary Numbers

Twos Complement Calculations

	4	0100	-4	1100
	+ 3	0011	+ (-3)	1101
	7	0111	-7	11001

If carry-in to sign = carry-out then ignore carry

if carry-in differs from carry-out then overflow

	4	0100	-4	1100
	- 3	1101	+ 3	0011
	1	10001	-1	1111

Simpler addition scheme makes twos complement the most common choice for integer number systems within digital systems

Number Systems
Addition and Subtraction of Binary Numbers

Twos Complement Calculations

Why can the carry-out be ignored?

-M + N when N > M:

$M^* + N = (2^n - M) + N = 2^n + (N - M)$

Ignoring carry-out is just like subtracting 2^n

-M + -N where N + M < or = 2^{n-1}

$-M + (-N) = M^* + N^* = (2^n - M) + (2^n - N)$

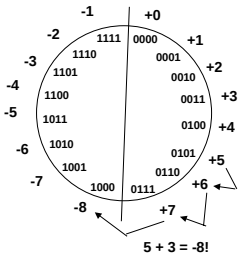
$= 2^n - (M + N) + 2^n$

After ignoring the carry, this is just the right twos compl. representation for -(M + N)!

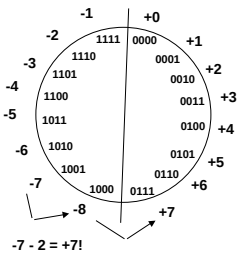
Number Systems
Overflow Conditions

Add two positive numbers to get a negative number

or two negative numbers to get a positive number



5 + 3 = -8!



-7 - 2 = +7!

Number Systems
Overflow Conditions

		0111			1000
5		0101	-7		1001
3		0011	-2		1100
-8		1000	7		10111
Overflow			Overflow		
		0000			1111
5		0101	-3		1101
2		0010	-5		1011
7		0111	-8		11000
No overflow			No overflow		

Overflow when carry in to sign does not equal carry out

Networks for Binary Addition
Half Adder

With twos complement numbers, addition is sufficient

Ai	Bi	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

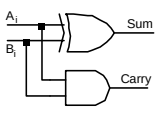
Bi \ Ai	0	1
0	0	1
1	1	0

$Sum = \overline{Ai}\,Bi + Ai\,\overline{Bi}$

$= Ai \oplus Bi$

Bi \ Ai	0	1
0	0	0
1	0	1

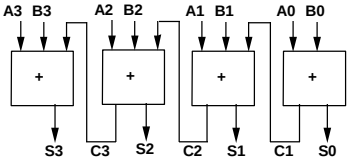
$Carry = Ai\,Bi$



Half-adder Schematic

Networks for Binary Addition
Full Adder

Cascaded Multi-bit Adder

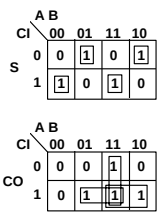


usually interested in adding more than two bits

this motivates the need for the full adder

Networks for Binary Addition
Full Adder

A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

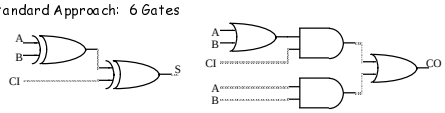


S = CI xor A xor B

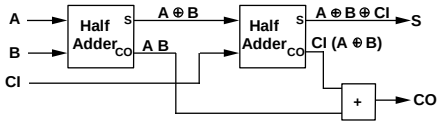
CO = B CI + A CI + A B = CI (A + B) + A B

Networks for Binary Addition
Full Adder/Half Adder

Standard Approach: 6 Gates

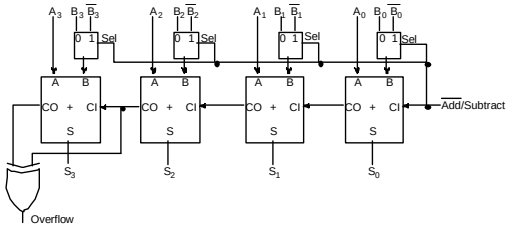


Alternative Implementation: 5 Gates



A B + CI (A xor B) = A B + B CI + A CI

Networks for Binary Addition
Adder/Subtractor

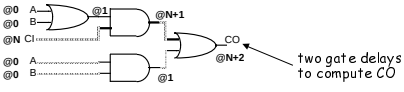


A - B = A + (-B) = A + $\overline{B}$ + 1

Networks for Binary Addition
Carry Lookahead Circuits

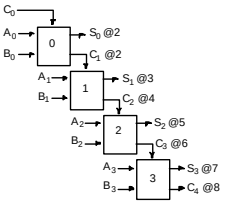
Critical delay: the propagation of carry from low to high order stages

late arriving signal



two gate delays to compute CO

4 stage adder

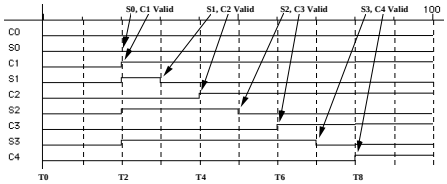


final sum and carry

Networks for Binary Addition
Carry Lookahead Circuits

Critical delay: the propagation of carry from low to high order stages

1111 + 0001 worst case addition



T0: Inputs to the adder are valid

T2: Stage 0 carry out (C1)

T4: Stage 1 carry out (C2)

T6: Stage 2 carry out (C3)

T8: Stage 3 carry out (C4)

2 delays to compute sum

but last carry not ready until 6 delays later

Networks for Binary Addition
Carry Lookahead Logic

Carry Generate Gi = Ai Bi *must generate carry when A = B = 1*

Carry Propagate Pi = Ai xor Bi *carry in will equal carry out here*

Sum and Carry can be reexpressed in terms of generate/propagate:

Si = Ai xor Bi xor Ci = Pi xor Ci

Ci+1 = Ai Bi + Ai Ci + Bi Ci

= Ai Bi + Ci (Ai + Bi)

= Ai Bi + Ci (Ai xor Bi)

= Gi + Ci Pi

Networks for Binary Addition
Carry Lookahead Logic

Reexpress the carry logic as follows:

$C1 = G0 + P0\ C0$

$C2 = G1 + P1\ C1 = G1 + P1\ G0 + P1\ P0\ C0$

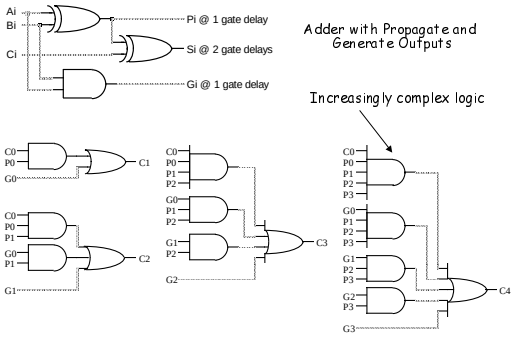
$C3 = G2 + P2\ C2 = G2 + P2\ G1 + P2\ P1\ G0 + P2\ P1\ P0\ C0$

$C4 = G3 + P3\ C3 = G3 + P3\ G2 + P3\ P2\ G1 + P3\ P2\ P1\ G0 + P3\ P2\ P1\ P0\ C0$

Each of the carry equations can be implemented in a two-level logic network

Variables are the adder inputs and carry in to stage 0!

Networks for Binary Addition
Carry Lookahead Implementation

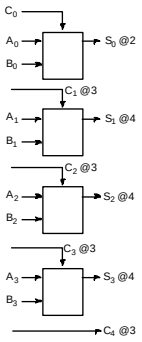


Networks for Binary Addition
Carry Lookahead Logic

Cascaded Carry Lookahead

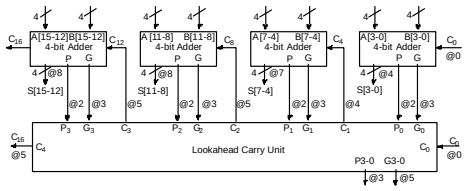
Carry lookahead logic generates individual carries

sums computed much faster



Networks for Binary Addition
Carry Lookahead Logic

Cascaded Carry Lookahead

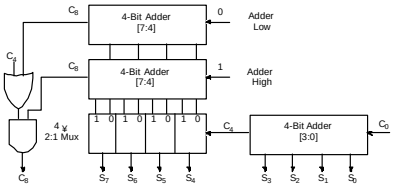


4 bit adders with internal carry lookahead

second level carry lookahead unit, extends lookahead to 16 bits

Networks for Binary Addition
Carry Select Adder

Redundant hardware to make carry calculation go faster



compute the high order sums in parallel

one addition assumes carry in = 0

the other assumes carry in = 1

Arithmetic Logic Unit Design
Sample ALU

M = 0, Logical Bitwise Operations

S1	S0	Function	Comment
0	0	Fi = Ai	Input Ai transferred to output
0	1	Fi = not Ai	Complement of Ai transferred to output
1	0	Fi = Ai xor Bi	Compute XOR of Ai, Bi
1	1	Fi = Ai xnor Bi	Compute XNOR of Ai, Bi

M = 1, C0 = 0, Arithmetic Operations

0	0	F = A	Input A passed to output
0	1	F = not A	Complement of A passed to output
1	0	F = A plus B	Sum of A and B
1	1	F = (not A) plus B	Sum of B and complement of A

M = 1, C0 = 1, Arithmetic Operations

0	0	F = A plus 1	Increment A
0	1	F = (not A) plus 1	Twos complement of A
1	0	F = A plus B plus 1	Increment sum of A and B
1	1	F = (not A) plus B plus 1	B minus A

Logical and Arithmetic Operations

Not all operations appear useful, but "fall out" of internal logic

Arithmetic Logic Unit Design

Sample ALU

Traditional Design Approach

Truth Table & Espresso

23 product terms!

Equivalent to 25 gates

```
.i 6
.o 2
.ilb m s1 s0 ci ai bi
.ob fi co
.p 23
111101 10
110111 10
1-0100 10
1-1110 10
10010- 10
10111- 10
-10001 10
010-01 10
-11011 10
011-11 10
--1000 10
0-1-00 10
--0010 10
0-0-10 10
-0100- 10
001-0- 10
-0001- 10
000-1- 10
-1-1-1 01
--1-01 01
--0-11 01
--110- 01
--011- 01
.e
```

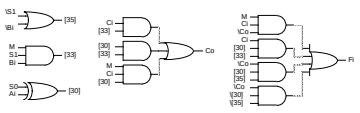
M	S1	S0	Ci	Ai	Bi	Fi	Ci+1
0	0	0	X	0	X	0	X
			X	1	X	1	X
	0	1	X	0	X	1	X
			X	1	X	0	X
	1	0	X	0	0	0	X
			X	0	1	1	X
	1	1	X	0	0	1	X
			X	0	1	0	X
			X	1	0	0	X
			X	1	1	1	X
1	0	0	0	0	X	0	X
			0	1	X	1	X
	0	1	0	0	X	1	X
			0	1	X	0	X
	1	0	0	0	0	0	0
			0	0	1	1	0
			0	1	0	1	0
			0	1	1	0	1
	1	1	0	0	0	1	0
			0	0	1	0	1
			0	1	0	0	0
			0	1	1	1	0
1	0	0	1	0	X	1	0
			1	1	X	0	1
	0	1	1	0	X	0	1
			1	1	X	1	0
	1	0	1	0	0	1	0
			1	0	1	0	1
			1	1	0	0	1
			1	1	1	1	1
	1	1	1	0	0	0	1
			1	0	1	1	1
			1	1	0	1	0
			1	1	1	0	1

Arithmetic Logic Unit Design

Sample ALU

Multilevel Implementation

```
.model alu.espresso
.inputs m s1 s0 ci ai bi
.outputs fi co
.names m ci co [30] [33] [35] fi
110--- 1
-1-11- 1
--01-1 1
--00-0 1
.names m ci [30] [33] co
-1-1 1
--11 1
111- 1
.names s0 ai [30]
01 1
10 1
.names m s1 bi [33]
111 1
.names s1 bi [35]
0- 1
-0 1
.end
```

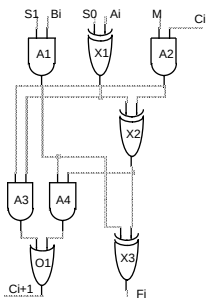


12 Gates

Arithmetic Logic Unit Design

Sample ALU

Clever Multi-level Logic Implementation



8 Gates (but 3 are XOR)

S1 = 0 blocks Bi
Happens when operations involve Ai only

Same is true for Ci when M = 0

Addition happens when M = 1

Bi, Ci to Xor gates X2, X3

S0 = 0, X1 passes A

S0 = 1, X1 passes $\overline{A}$

Arithmetic Mode:

Or gate inputs are Ai Ci and Bi (Ai xor Ci)

Logic Mode:

Cascaded XORs form output from Ai and Bi

Arithmetic Logic Unit Design

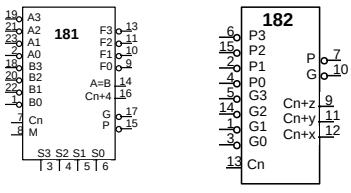
74181 TTL ALU

Selection				M = 1	M = 0, Arithmetic Functions	
S3	S2	S1	S0	Logic Function	Cn = 0	Cn = 1
0	0	0	0	F = not A	F = A minus 1	F = A
0	0	0	1	F = A nand B	F = A B minus 1	F = A B
0	0	1	0	F = (not A) + B	F = A (not B) minus 1	F = A (not B)
0	0	1	1	F = 1	F = minus 1	F = zero
0	1	0	0	F = A nor B	F = A plus (A + not B)	F = A plus (A + not B) plus 1
0	1	0	1	F = not B	F = A B plus (A + not B)	F = A B plus (A + not B) plus 1
0	1	1	0	F = A xnor B	F = A minus B minus 1	F = (A + not B) plus 1
0	1	1	1	F = A + not B	F = A + not B	F = A minus B
1	0	0	0	F = (not A) B	F = A plus (A + B)	F = (A + not B) plus 1
1	0	0	1	F = A xor B	F = A plus B	F = A plus (A + B) plus 1
1	0	1	0	F = B	F = A (not B) plus (A + B)	F = A (not B) plus (A + B) plus 1
1	0	1	1	F = A + B	F = (A + B)	F = (A + B) plus 1
1	1	0	0	F = 0	F = A	F = A plus A plus 1
1	1	0	1	F = A (not B)	F = A B plus A	F = AB plus A plus 1
1	1	1	0	F = A B	F= A (not B) plus A	F = A (not B) plus A plus 1
1	1	1	1	F = A	F = A	F = A plus 1

Arithmetic Logic Unit Design

74181 TTL ALU

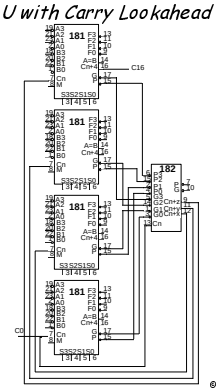
Note that the sense of the carry in and out are OPPOSITE from the input bits



Fortunately, carry lookahead generator maintains the correct sense of the signals

Arithmetic Logic Unit Design

16-bit ALU with Carry Lookahead



Lecture Review

We have covered:

- *Binary Number Representation*
  positive numbers the same
  difference is in how negative numbers are represented
  twos complement easiest to handle:
    one representation for zero, slightly
    complicated complementation, simple addition
- *Binary Networks for Additions*
  basic HA, FA
  carry lookahead logic
- *ALU Design*
  specification and implementation