

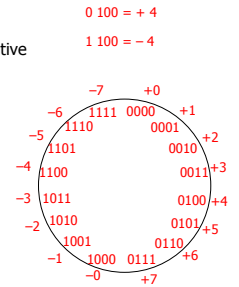
EECS150

Section 8 Arithmetic Circuits Fall 2001



Sign and Magnitude

- One bit dedicate to sign (positive or negative)
 - sign: 0 = positive (or zero), 1 = negative
- Rest represent the absolute value or magnitude
 - three low order bits: 0 (000) thru 7 (111)
- Range for n bits
 - +/- 2ⁿ⁻¹ - 1 (two representations for 0)
- Cumbersome addition/subtraction
 - must compare magnitudes to determine sign of result



Arithmetic Circuits

- Excellent Examples of Combinational Logic Design
- Time vs. Space Trade-offs
 - Doing things fast may require more logic and thus more space
 - Example: carry lookahead logic
- Arithmetic and Logic Units
 - General-purpose building blocks
 - Critical components of processor datapaths
 - Used within most computer instructions

1s Complement

- If N is a positive number, then the negative of N (its 1s complement or N') is N' = (2ⁿ - 1) - N
 - Example: 1s complement of 7

$$\begin{array}{r} 2^4 = 10000 \\ 1 = 00001 \\ \hline 2^4 - 1 = 1111 \\ 7 = 0111 \\ \hline 1000 = -7 \text{ in 1s complement form} \end{array}$$

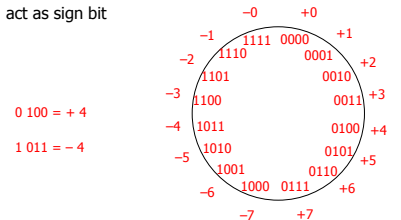
- Shortcut: simply compute bit-wise complement (0111 -> 1000)

Number Systems

- Representation of positive numbers is the same in most systems
- Major differences are in how negative numbers are represented
- Representation of negative numbers come in three major schemes
 - Sign and magnitude
 - 1s complement
 - 2s complement
- Assumptions
 - We'll assume a 4 bit machine word
 - 16 different values can be represented
 - Roughly half are positive, half are negative

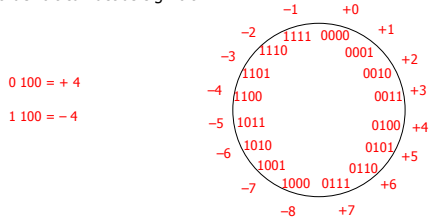
1s complement (cont'd)

- Subtraction implemented by 1s complement and then addition
- Two representations of 0
 - Causes some complexities in addition
- High-order bit can act as sign bit



2s Complement

- 1s complement with negative numbers shifted one position clockwise
 - Only one representation for 0
 - One more negative number than positive number
 - High-order bit can act as sign bit



0 100 = +4

1 100 = -4

Why Can the Carry-out be Ignored?

- Can't ignore it completely
 - Needed to check for overflow (see next two slides)
- When there is no overflow, carry-out may be true but can be ignored

- M + N when N > M:

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

ignoring carry-out is just like subtracting 2^n

- M + - N where N + M $\leq 2^{n-1}$

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

ignoring the carry, it is just the 2s complement representation for -(M + N)

2s complement (cont'd)

- If N is a positive number, then the negative of N (its 2s complement or N^*) is $N^* = 2^n - N$

- Example: 2s complement of 7

$$\begin{array}{r} 2^4 = 10000 \\ \text{subtract } 7 = \underline{0111} \\ \hline 1001 = \text{repr. of } -7 \end{array}$$

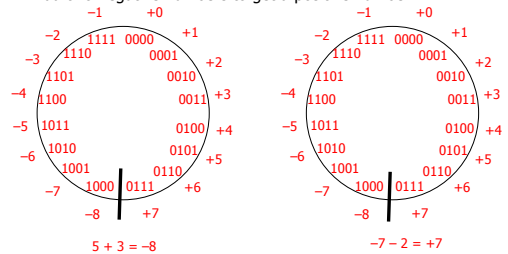
- Example: 2s complement of -7

$$\begin{array}{r} 2^4 = 10000 \\ \text{subtract } -7 = \underline{1001} \\ \hline 0111 = \text{repr. of } 7 \end{array}$$

- Shortcut: 2s complement = bit-wise complement + 1
 - 0111 -> 1000 + 1 -> 1001 (representation of -7)
 - 1001 -> 0110 + 1 -> 0111 (representation of 7)

Overflow in 2s Complement

- Overflow conditions
 - Add two positive numbers to get a negative number
 - Add two negative numbers to get a positive number



5 + 3 = 8

-7 - 2 = +7

Addition / Subtraction

- Simple Addition and Subtraction
 - Simple scheme makes 2s complement the virtually unanimous choice for integer number systems in computers

$$\begin{array}{r} 4 \quad 0100 \\ +3 \quad 0011 \\ \hline 7 \quad 0111 \end{array} \quad \begin{array}{r} -4 \quad 1100 \\ +(-3) \quad 1101 \\ \hline -7 \quad 11001 \end{array}$$

$$\begin{array}{r} 4 \quad 0100 \\ -3 \quad 1101 \\ \hline 1 \quad 10001 \end{array} \quad \begin{array}{r} -4 \quad 1100 \\ +3 \quad 0011 \\ \hline -1 \quad 1111 \end{array}$$

Overflow Conditions

- Overflow when carry into sign bit position is not equal to carry-out

$$\begin{array}{r} 0111 \\ 5 \quad 0101 \\ \underline{-3} \quad 0011 \\ -8 \quad 1000 \\ \hline \text{overflow} \end{array} \quad \begin{array}{r} 1000 \\ -7 \quad 1001 \\ \underline{-2} \quad 1110 \\ 7 \quad 10111 \\ \hline \text{overflow} \end{array}$$

$$\begin{array}{r} 0000 \\ 5 \quad 0101 \\ \underline{-2} \quad 0010 \\ 7 \quad 0111 \\ \hline \text{no overflow} \end{array} \quad \begin{array}{r} 1111 \\ -3 \quad 1101 \\ \underline{-5} \quad 1011 \\ -8 \quad 11000 \\ \hline \text{no overflow} \end{array}$$

Circuits for Binary Addition

- Half adder (add 2 1-bit numbers)
 - Sum = $A_i' B_i + A_i B_i' = A_i \text{ xor } B_i$
 - Cout = $A_i B_i$
- Full adder (carry-in to cascade for multi-bit adders)
 - Sum = $C_i \text{ xor } A \text{ xor } B$
 - Cout = $B C_i + A C_i + A B = C_i (A + B) + A B$

A_i	B_i	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

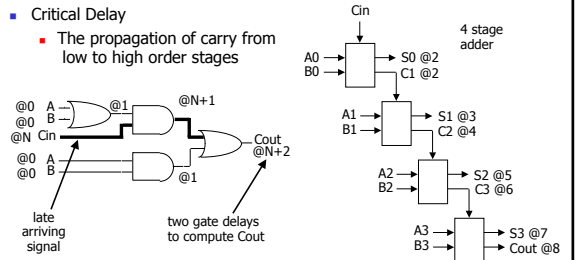
A_i	B_i	C_{in}	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

EECS150 - Fall 2001

1-13

Ripple-Carry Adders

- Critical Delay
 - The propagation of carry from low to high order stages

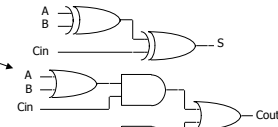


EECS150 - Fall 2001

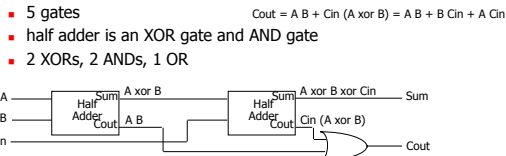
1-16

Full adder implementations

- Standard approach
 - 6 gates
 - 2 XORs, 2 ANDs, 2 ORs



- Alternative implementation

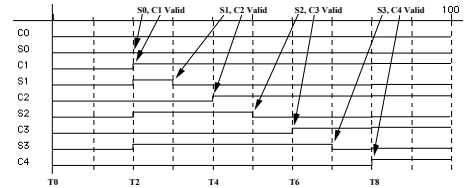


EECS150 - Fall 2001

1-14

Ripple-Carry Adders (cont'd)

- Critical delay
 - The propagation of carry from low to high order stages
 - 1111 + 0001 is the worst case addition
 - Carry must propagate through all bits

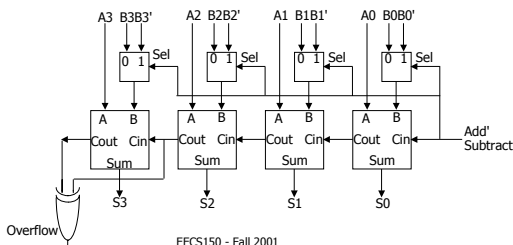


EECS150 - Fall 2001

1-17

Adder/Subtractor

- Use an adder to do subtraction thanks to 2s complement representation
 - $A - B = A + (-B) = A + B' + 1$
 - Control signal selects B or 2s complement of B



EECS150 - Fall 2001

1-15

Carry-Lookahead Logic

- Carry generate: $G_i = A_i B_i$
 - Must generate carry when $A = B = 1$
- Carry propagate: $P_i = A_i \text{ xor } B_i$
 - Carry-in will equal carry-out here
- Sum and Cout can be re-expressed in terms of generate/propagate:
 - $S_i = A_i \text{ xor } B_i \text{ xor } C_i = P_i \text{ xor } C_i$
 - $C_{i+1} = A_i B_i + A_i C_i + B_i C_i = A_i B_i + C_i (A_i + B_i) = A_i B_i + C_i (A_i \text{ xor } B_i) = G_i + C_i P_i$

EECS150 - Fall 2001

1-18

Carry-Lookahead Logic (cont'd)

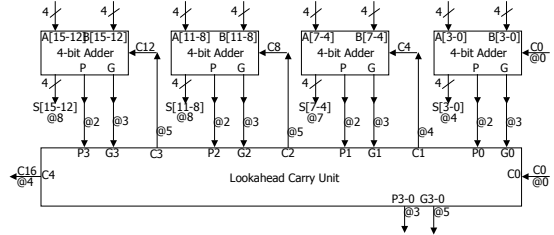
- Re-express the carry logic as follows:
 - $C1 = G0 + P0 C0$
 - $C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0$
 - $C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0$
 - $C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 C0$
- Each of the carry equations can be implemented with two-level logic
 - All inputs are now directly derived from data inputs and not from intermediate carries
 - this allows computation of all sum outputs to proceed in parallel

EECS150 - Fall 2001

1-19

Cascaded Carry-Lookahead Logic

- Carry-lookahead adder
 - 4 four-bit adders with internal carry lookahead
 - Second level carry lookahead unit extends lookahead to 16 bits

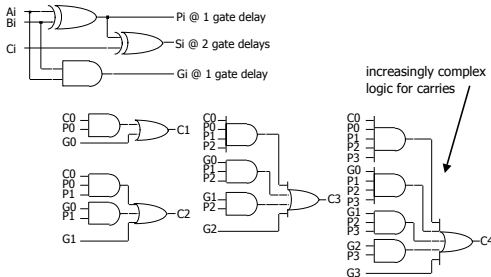


EECS150 - Fall 2001

1-22

Carry-Lookahead Implementation

- Adder with propagate and generate outputs

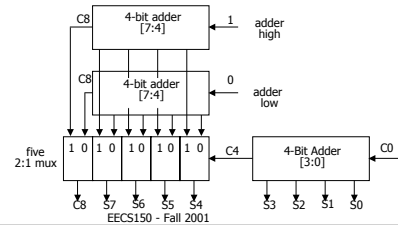


EECS150 - Fall 2001

1-20

Carry-Select Adder

- Redundant hardware to make carry calculation go faster
 - Compute two high-order sums in parallel while waiting for carry-in
 - One assuming carry-in is 0 and another assuming carry-in is 1
 - Select correct result once carry-in is finally computed

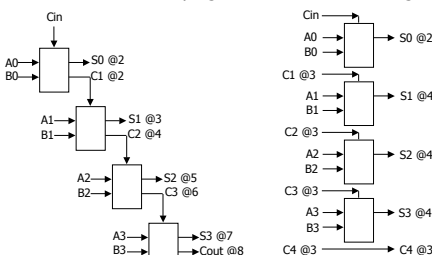


EECS150 - Fall 2001

1-23

Carry-Lookahead Implementation

- Carry-lookahead logic generates individual carries
 - Sums computed much more quickly in parallel
 - However, cost of carry logic increases with more stages



EECS150 - Fall 2001

1-21

Arithmetic Logic Unit Design

M = 0, logical bitwise operations

S1 S0	Function	Comment
0 0	F = Ai	input Ai transferred to output
0 1	F = not Ai	complement of Ai transferred to output
1 0	F = Ai xor Bi	compute XOR of Ai, Bi
1 1	F = 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

EECS150 - Fall 2001

1-24

Arithmetic Logic Unit Design

Sample ALU – truth table

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

BCD Addition

Decimal digits 0 thru 9 represented as 0000 thru 1001 in binary

Addition:

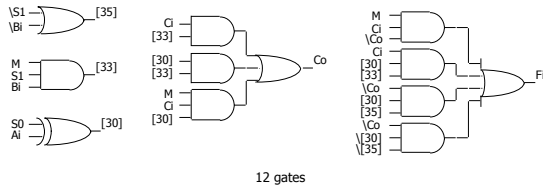
5 = 0101 5 = 0101
 3 = 0011 8 = 1000 Problem when digit sum exceeds 9
 1000 = 8 1101 = 13!

Solution: add 6 (0110) if sum exceeds 9!

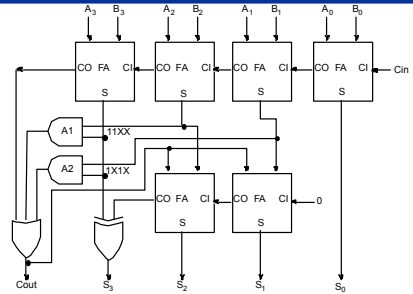
5 = 0101 9 = 1001
 8 = 1000 7 = 0111
 1101 1 0000 = 16 in binary
 6 = 0110 6 = 0110
 1 0011 = 13 in BCD 1 0110 = 16 in BCD

Arithmetic Logic Unit Design

Sample ALU – multi-level discrete gate logic implementation



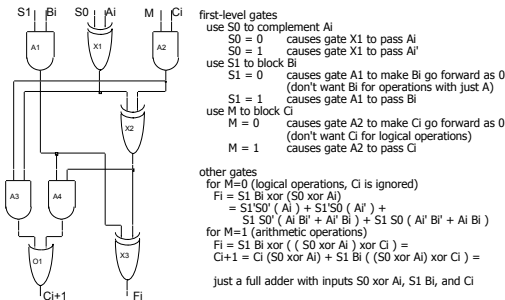
Adder Implementation



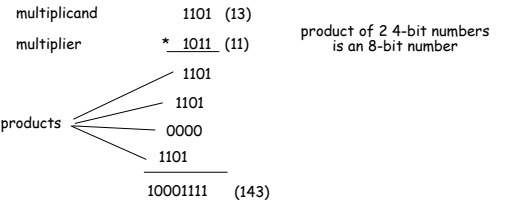
Add 0110 to sum whenever it exceeds 1001 (11XX or 1X1X)

Arithmetic Logic Unit Design

Sample ALU – clever multi-level implementation



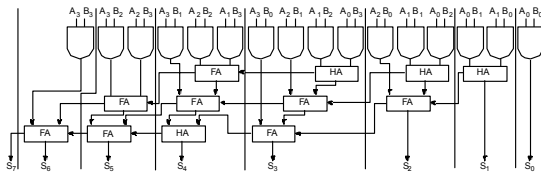
Combinational Multiplier



Partial Product Accumulation

				A3	A2	A1	A0		
				B3	B2	B1	B0		
				A2 B0	A2 B0	A1 B0	A0 B0		
			A3 B1	A2 B1	A1 B1	A0 B1			
		A3 B2	A2 B2	A1 B2	A0 B2				
	A3 B3	A2 B3	A1 B3	A0 B3					
S7	S6	S5	S4	S3	S2	S1	S0		

Partial Product Accumulation



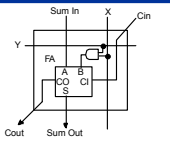
Note use of parallel carry-outs to form higher order sums

12 Adders, if full adders, this is 6 gates each = 72 gates

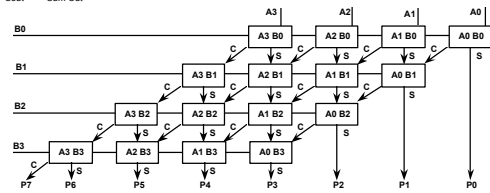
16 gates form the partial products

total = 88 gates!

Another Representation



Building block: full adder + and



4 x 4 array of building blocks