Chapter # 5: Arithmetic Circuits

Contemporary Logic Design

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:

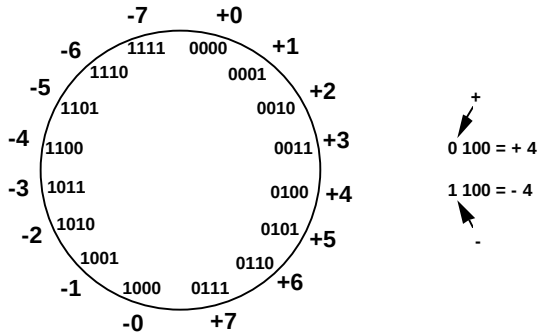
- **sign and magnitude**
- **ones complement**
- **twos complement**

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

5-3

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:

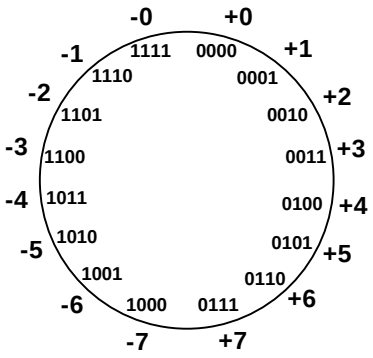
simply compute bit wise complement

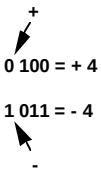
0111 -> 1000

5-4

Number Systems

Ones Complement





Subtraction implemented by addition & 1's complement

Still two representations of 0! This causes some problems

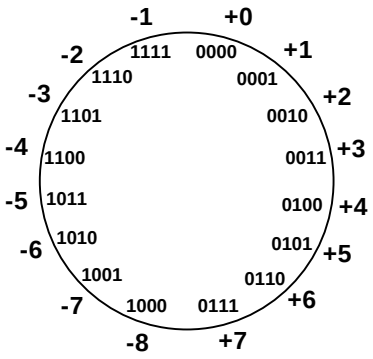
Some complexities in addition

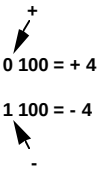
5-5

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

5-6

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

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

Overflow Conditions

	0 1 1 1			1 0 0 0	
5	0 1 0 1		-7	1 0 0 1	
3	0 0 1 1		-2	1 1 0 0	
-8	1 0 0 0		7	1 0 1 1 1	
Overflow			Overflow		

	0 0 0 0			1 1 1 1	
5	0 1 0 1		-3	1 1 0 1	
2	0 0 1 0		-5	1 0 1 1	
7	0 1 1 1		-8	1 1 0 0 0	
No overflow			No overflow		

Overflow when carry in to sign does not equal carry out

5-11

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

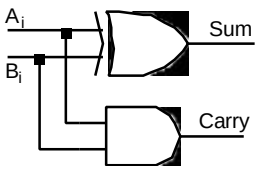
Bi \ Ai	0	1
0	0	1
1	1	0

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

$= Ai \oplus Bi$

Bi \ Ai	0	1
0	0	0
1	0	1

$Carry = Ai\,Bi$

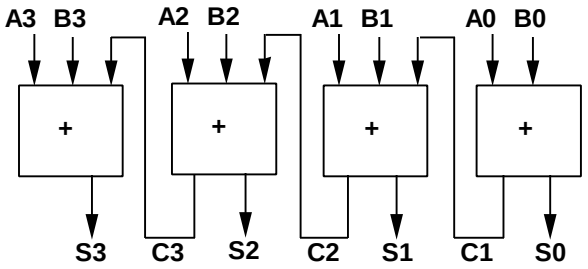


Half-adder Schematic

5-12

Full Adder

Cascaded Multi-bit Adder

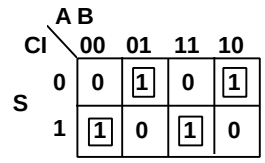


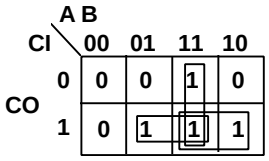
- usually interested in adding more than two bits
- this motivates the need for the full adder

5-13

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

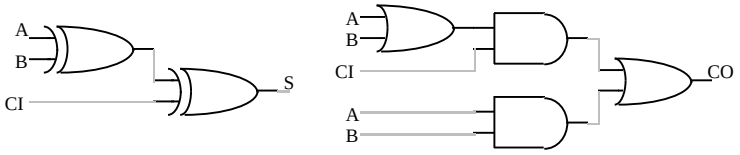
CO = B CI + A CI + A B = CI (A + B) + A B

5-14

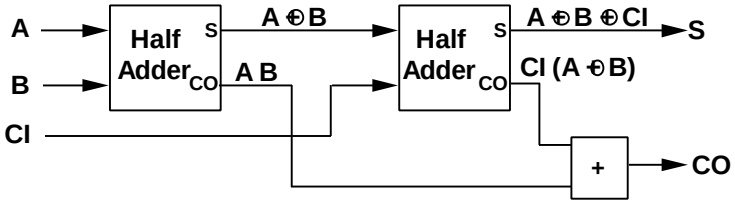
Networks for Binary Addition

Full Adder/Half Adder

Standard Approach: 6 Gates



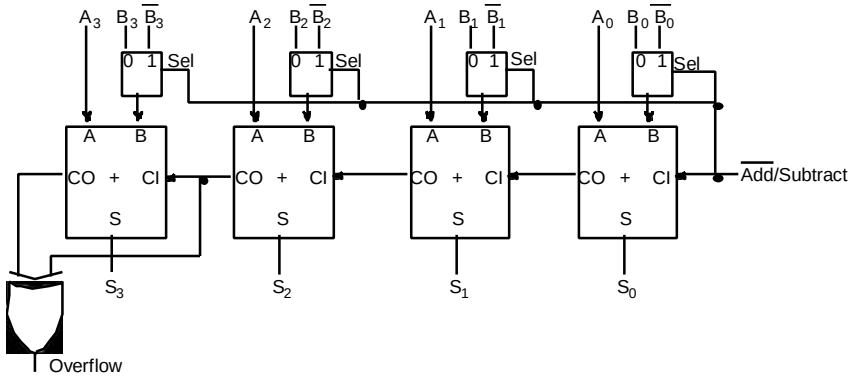
Alternative Implementation: 5 Gates



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

5-15

Adder/Subtractor



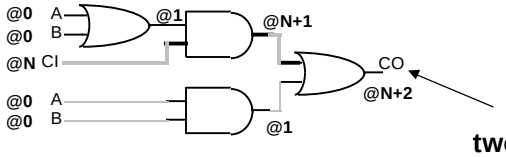
A - B = A + (-B) = A + $\overline{B}$ + 1

5-16

Carry Lookahead Circuits

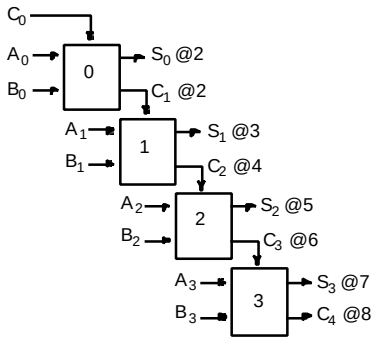
Critical delay: the propagation of carry from low to high order stages

late arriving signal



two gate delays to compute CO

4 stage adder



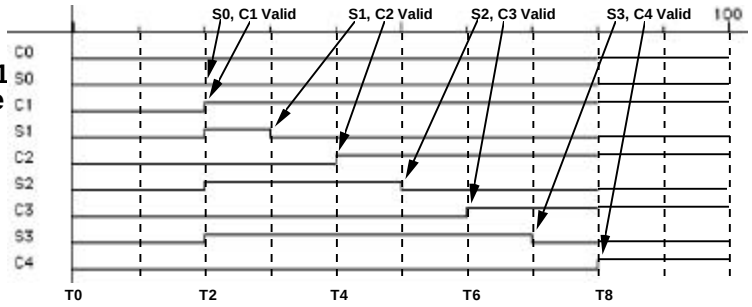
final sum and carry

5-17

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

5-18

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:

$$Si = Ai \text{ xor } Bi \text{ xor } Ci = Pi \text{ xor } Ci$$

$$Ci+1 = Ai\,Bi + Ai\,Ci + Bi\,Ci$$
$$= Ai\,Bi + Ci\,(Ai + Bi)$$
$$= Ai\,Bi + Ci\,(Ai \text{ xor } Bi)$$
$$= Gi + Ci\,Pi$$

5-19

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!

5-20

Carry Lookahead Logic

$G_i = A_i B_i$ Pi = Ai xor Bi

$C_1 = G_0 + P_0 C_0$
$= A_0B_0$

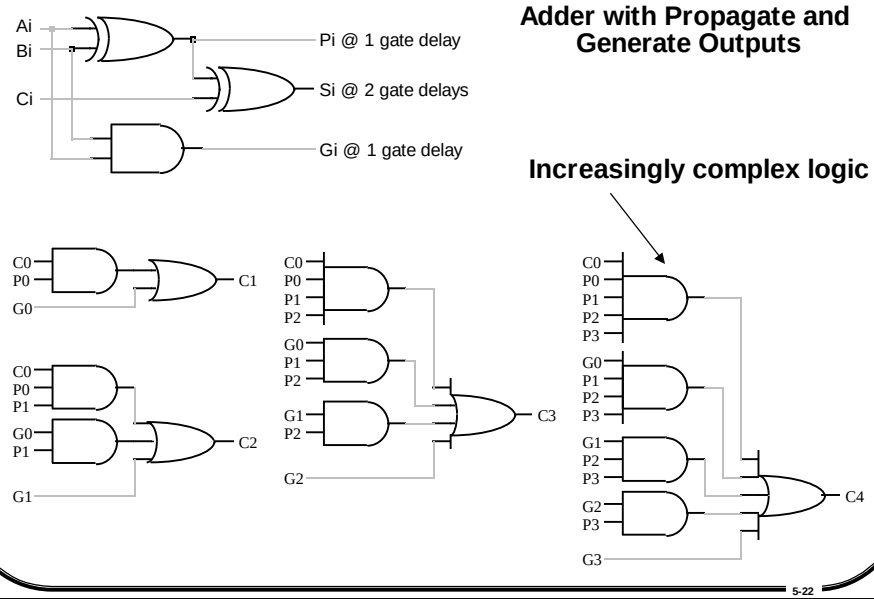
$C_2 = G_1 + P_1 C_1$
$= A_1B_1 + (A_1 \text{ xor } B_1) A_0B_0$

$C_3 = G_2 + P_2 C_2$
$= A_2B_2 + (A_2 \text{ xor } B_2)(A_1B_1 + (A_1 \text{ xor } B_1) A_0B_0)$

$C_4 = G_3 + P_3 C_3$
$= A_3B_3 + (A_3 \text{ xor } B_3)(A_2B_2 + (A_2 \text{ xor } B_2)(A_1B_1 + (A_1 \text{ xor } B_1)A_0B_0))$

5-21

Carry Lookahead Implementation

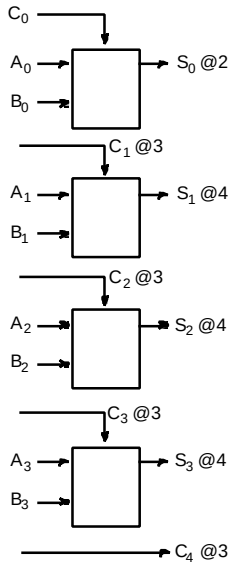


5-22

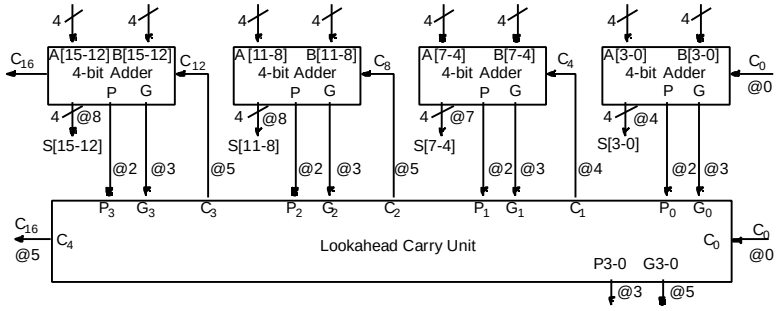
Cascaded Carry Lookahead

Carry lookahead logic generates individual carries

sums computed much faster



Cascaded Carry Lookahead

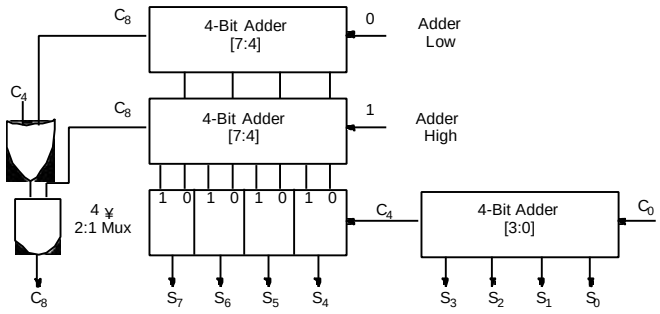


4 bit adders with internal carry lookahead

second level carry lookahead unit, extends lookahead to 16 bits

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

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

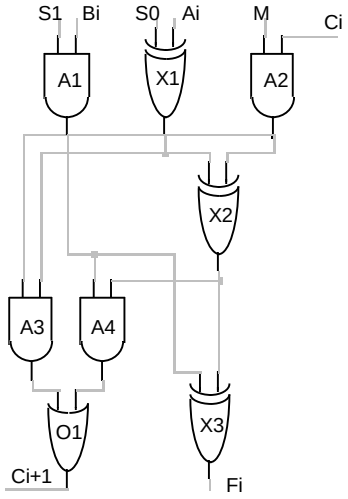
Logical and Arithmetic Operations

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

Arithmetic Logic Unit Design

Sample ALU

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:

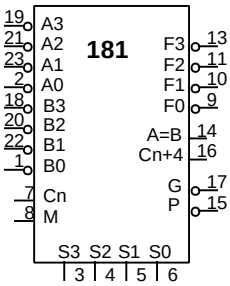
Cascaded XORs form output from Ai and Bi

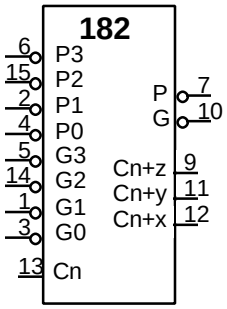
74181 TTL ALU

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

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

5-29

BCD Addition

5 = 0101 **5 = 0101**

3 = 0011 **8 = 1000**

1000 = 8 **1101 = 13!**

Problem when digit sum exceeds 9

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

5 = 0101 **9 = 1001**

8 = 1000 **7 = 0111**

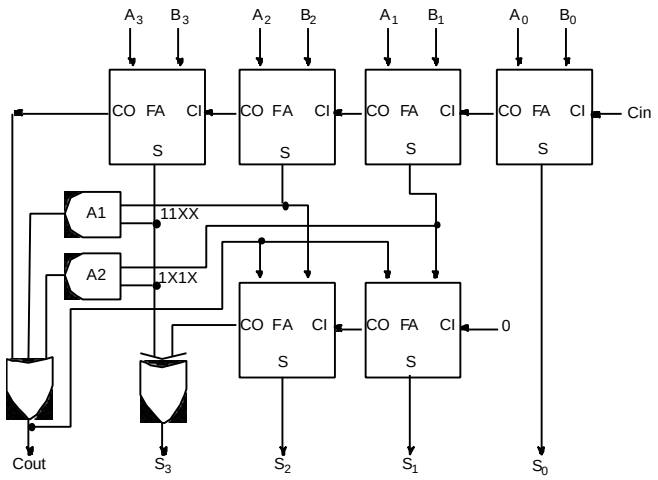
1101 **1 0000 = 16 in binary**

6 = 0110 **6 = 0110**

1 0011 = 1 3 in BCD **1 0110 = 1 6 in BCD**

5-30

BCD Addition



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

5-31

Combinational Multiplier

Basic Concept

multiplicand 1101 (13)

multiplier * 1011 (11)

Partial products

1101

1101

0000

1101

10001111 (143)

product of 2 4-bit numbers is an 8-bit number

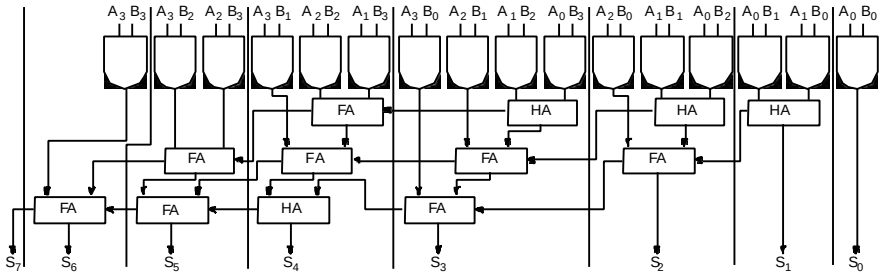
5-32

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

5-33

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

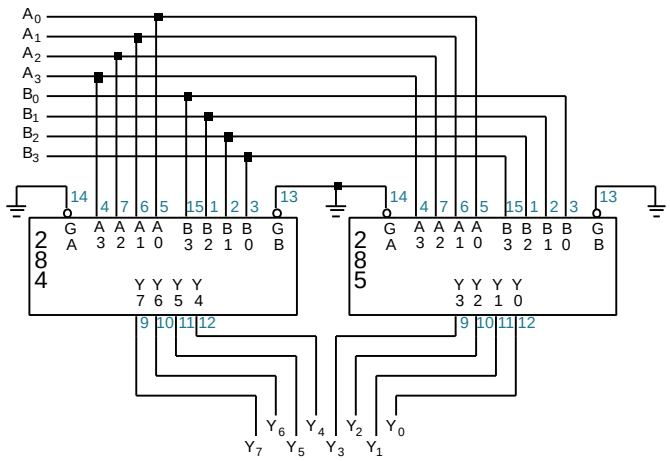
16 gates form the partial products

total = 88 gates!

5-34

Case Study: 8 x 8 Multiplier

TTL Multipliers



Two chip implementation of 4 x 4 multipler