Lecture 5
Number Representation and Arithmetic Circuits

吳文中

Positional Number System

- Positional Number System
  - E.g. 5185.68 = 5 x 10^3 + 1 x 10^2 + 8 x 10 + 5 x 1 + 6 x 10^{-1} +8 x 10^{-2}
  - Each digit position has an associated *weight*.
  - Each digit associated with a radix power
  - Allows negative power to be used after decimal point .
- General form: $d_{p-1}d_{p-2}\cdots d_1 d_0 . d_{-1}d_{-2}\cdots d_{-n}$
- Unsigned Integers: $d_{p-1}d_{p-2}\cdots d_1 d_0$
  - *n* digits before radix point, and *p* digits after radix point.

$$D = \sum_{i=-n}^{p-1} d_i \cdot r^i$$

  - where d_i denotes weight, and *r* denotes base radix
  - Leftmost digit: most significant digit, right most digit: least significant digit.
- Base radix = 10 is decimal system (decimal radix) general used decimal system.

Binary Number System

Base radix =2 is binary system (binary radix) and always used in digital system.

– $b_{p-1}b_{p-2}\cdots b_1b_0 .\ b_{-1}b_{-2}\cdots b_{-n}$

– n digits before binary point, and p digits after radix point.

$$B = \sum_{i=-n}^{p-1} b_i \cdot 2^i$$

– Digits in binary radix are binary digits or bits.

– Leftmost bit: most significant bit (MSB), rightmost bit: least significant bit (LSB).

Octal and Hexadecimal Numbers

- Octal number system: base radix 8
  - 1, 2, 3, 4, 5, 6, 7, 0
- Hexadecimal number system: base radix 16
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 0
- Octal and Hexadecimal number system are bridges between binary and decimal number system, digital world and human world.
  - $100\ 011\ 001\ 110_2 = 4316_8$
  - $1000\ 1100\ 1110_2 = 8CE_{16}$
  - $10.1011001011_2 = 010.101\ 100\ 101\ 100_2 = 2.5454_8$
    $=0010.1011\ 0010\ 1100_2 = 2.B2C_{16}$

Octal- or Hexadecimal Conversion

Binary	Decimal	Octal	3-bit	Hexadecimal	4-Bit String
0	0	0	000	0	0000
1	1	1	001	1	0001
10	2	2	010	2	0010
11	3	3	011	3	0011
100	4	4	100	4	0100
101	5	5	101	5	0101
110	6	6	110	6	0110
111	7	7	111	7	0111
1000	8	10	-	8	1000
1001	9	11	-	9	1001
1010	10	12	-	A	1010
1011	11	13	-	B	1011
1100	12	14	-	C	1100
1101	13	15	-	D	1101
1110	14	16	-	E	1110
1111	16	17	-	F	1111

General Positional-Number-System to Decimal Conversions

- Any base radix to decimal conversion

$$D = \sum_{i=-n}^{p-1} d_i \cdot r^i$$

- Nested expansion

$D=((\cdots((d_{p-1}) \cdot r + d_{p-2}) \cdot r + \cdot \cdot \cdot) \cdot r + d_1) \cdot r + d_0$

  - $F1AC_{16}$ $= 15 \times 16^3 + 1 \times 16^2 + 10 \times 16^1 + 12 \times 16^0$
    $= (((15) \cdot 16 + 1) \cdot 16 + 10) \cdot 16 + 12 = 61868_{10}$

Decimal to General Positional-Number-System Conversions

- Divide the D by r , the quotient Q will be d_0

  $d_0 = Q = (\cdots((d_{p-1}) \cdot r + d_{p-2}) \cdot r + \cdot \cdot \cdot) \cdot r + d_1$
- 179 / 2 = 89 remainder 1 (LSB)

  /2=44 remainder 1

  /2=22 remainder 0

  /2=11 remainder 0

  /2 = 5 remainder 1

  /2=2 remainder 1

  /2 = 1 remainder 0

  /2 = 0 remainder 1(MSB)
- $179_{10} = 10110011_2$

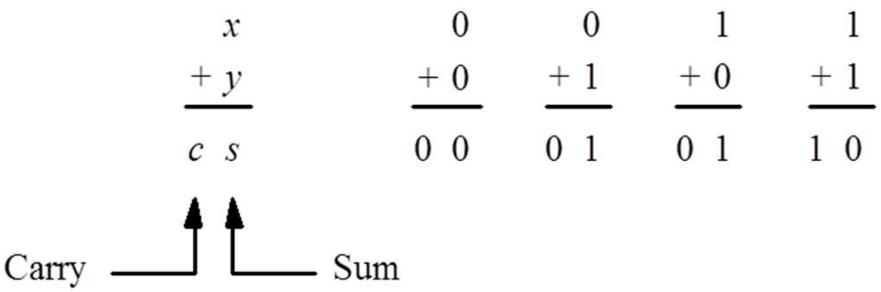
Conversion Methods for Common Radices

Conversion	Method	Example
Binary to		
Octal	Substitution	$10111011001_2 = 10\ 111\ 011\ 001_2 = 2731_8$
Hexadecimal	Substitution	$10111011001_2 = 101\ 1101\ 1001_2 = 5D9_{16}$
Decimal	Summation	$10111011001_2 = 1 \cdot 1024 + 0 \cdot 512 + 1 \cdot 256 + 1 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 1497_{10}$
Octal to		
Binary	Substitution	$1234_8 = 001\ 010\ 011\ 100_2$
Hexadecimal	Substitution	$1234_8 = 001\ 010\ 011\ 100_2 = 0010\ 1001\ 1100_2 = 29C_{16}$
Decimal	Summation	$1234_8 = 1 \cdot 512 + 2 \cdot 64 + 3 \cdot 8 + 4 \cdot 1 = 668_{10}$
Hexadecimal to		
Binary	Substitution	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2$
Octal	Substitution	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2 = 1\ 100\ 000\ 011\ 011\ 110_2 = 140336_8$
Decimal	Summation	$C0DE_{16} = 12 \cdot 4096 + 0 \cdot 256 + 13 \cdot 16 + 14 \cdot 1 = 49374_{10}$

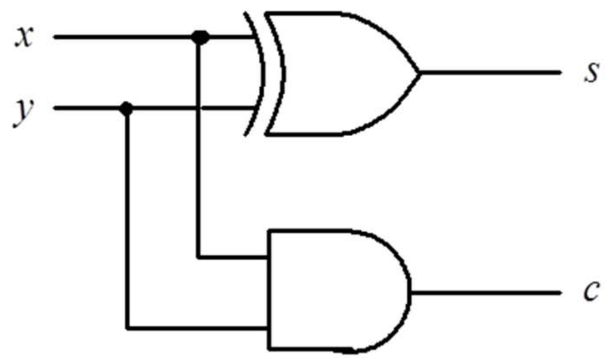
Conversion Methods for Common Radices (Con't)

Conversion	Method	Example
Decimal to Binary	Division	$108_{10} = 1101100_2$ $108_{10} \div 2 = 54$ remainder 0 (LSB) $\div 2 = 27$ remainder 0 $\div 2 = 13$ remainder 1 $\div 2 = 6$ remainder 1 $\div 2 = 3$ remainder 0 $\div 2 = 1$ remainder 1 $\div 2 = 0$ remainder 1 (MSB)
Octal	Division	$108_{10} = 154_8$ $108_{10} \div 8 = 13$ remainder 4 (least significant digit) $\div 8 = 1$ remainder 5 $\div 8 = 0$ remainder 1 (most significant digit)
Hexadecimal	Division	$108_{10} = 6C_{16}$ $108_{10} \div 16 = 6$ remainder 12 (least significant digit) $\div 16 = 0$ remainder 6 (most significant digit)

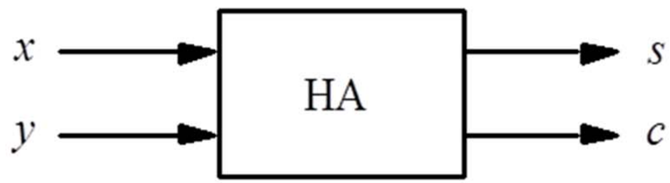
Addition of Unsigned Numbers



(a) The four possible cases



x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$s_i = \overline{x_i} y_i \overline{c_i} + x_i \overline{y_i}\,\overline{c_i} + \overline{x_i}\,\overline{y_i} c_i + x_i y_i c_i$$

An Addition Example

$X = x_4x_3x_2x_1x_0$	0 1 1 1 1	$(15)_{10}$	
$+\ Y = y_4y_3y_2y_1y_0$	0 1 0 1 0	$(10)_{10}$	
	1 1 1 0		Generated carries
$S = s_4s_3s_2s_1s_0$	1 1 0 0 1	$(25)_{10}$	

Use of XOR Gates

- XOR function is defined as $x_1 \oplus x_2 = \overline{x_1} x_2 + x_1 \overline{x_2}$
- XNOR function is denoted as $\overline{x_1 \oplus x_2} = x_1 \odot x_2$
- $s_i = \overline{x_i} y_i \overline{c_i} + x_i \overline{y_i}\,\overline{c_i} + \overline{x_i}\,\overline{y_i} c_i + x_i y_i c_i$
  $= (\overline{x_i} y_i + x_i \overline{y_i}) \overline{c_i} + (\overline{x_i}\,\overline{y_i} + x_i y_i) c_i$
  $= (x_i \oplus y_i) \overline{c_i} + (\overline{x_i \oplus y_i}) c_i$
  $= x_i \oplus y_i \oplus c_i$
- XOR/ XNOR truth table

x_1	x_2	$\oplus$	$\odot$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder

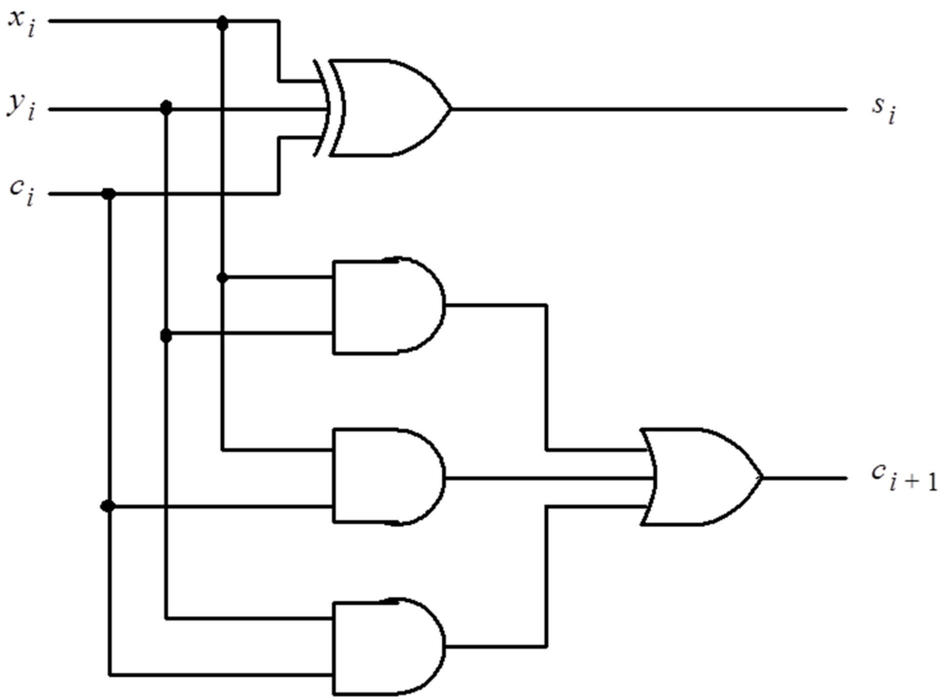
c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

c_i \ x_iy_i	00	01	11	10
0		1		1
1	1		1	

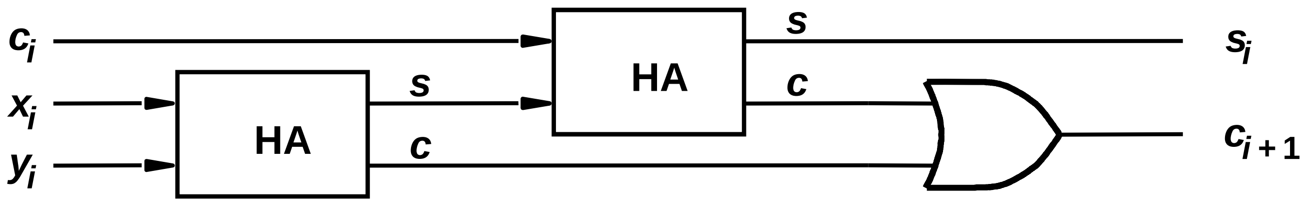
$$s_i = x_i \oplus y_i \oplus c_i$$

c_i \ x_iy_i	00	01	11	10
0			1	
1		1	1	1

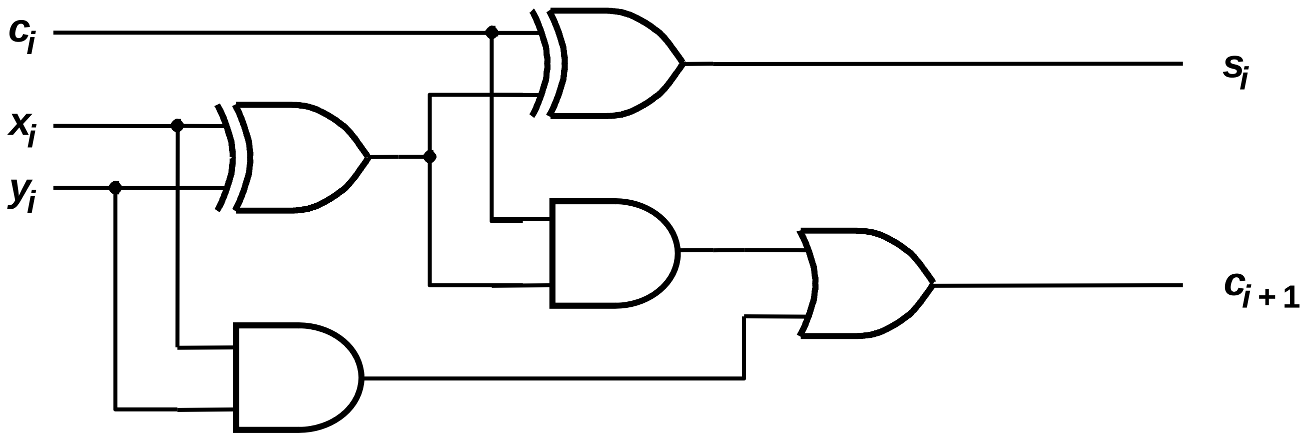
$$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$$



Decomposed Full-Adder



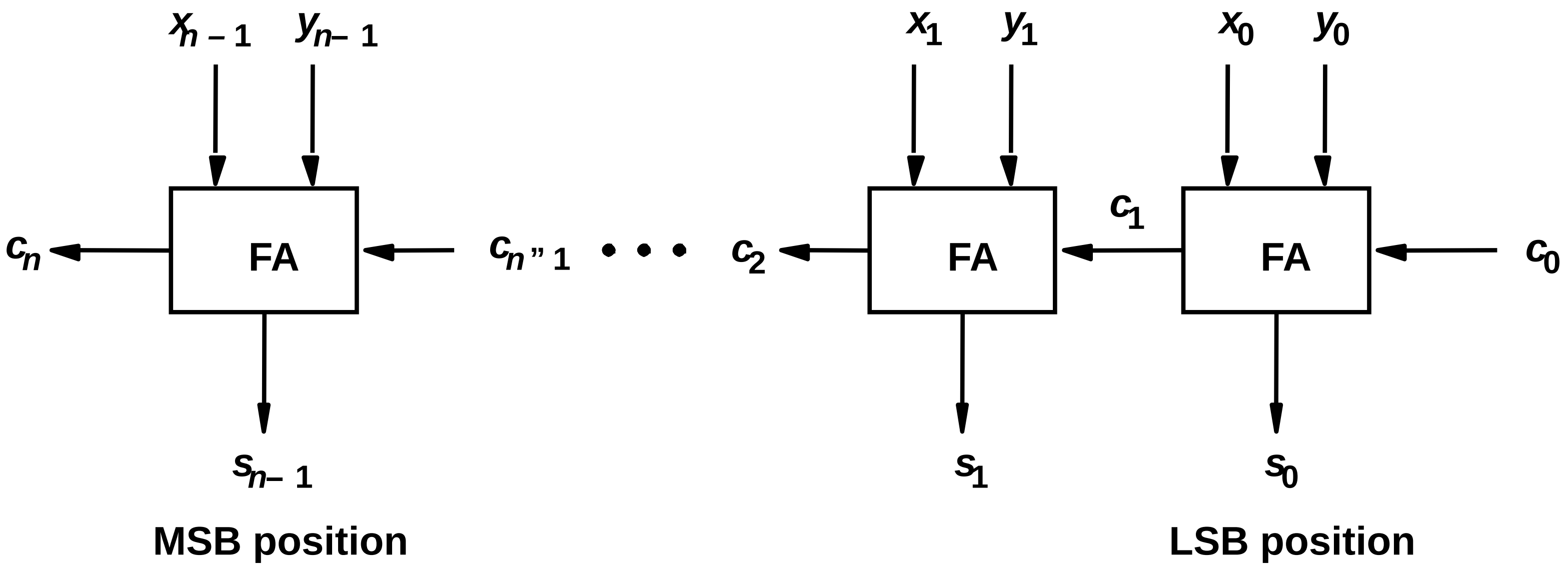
Block diagram



Detailed diagram

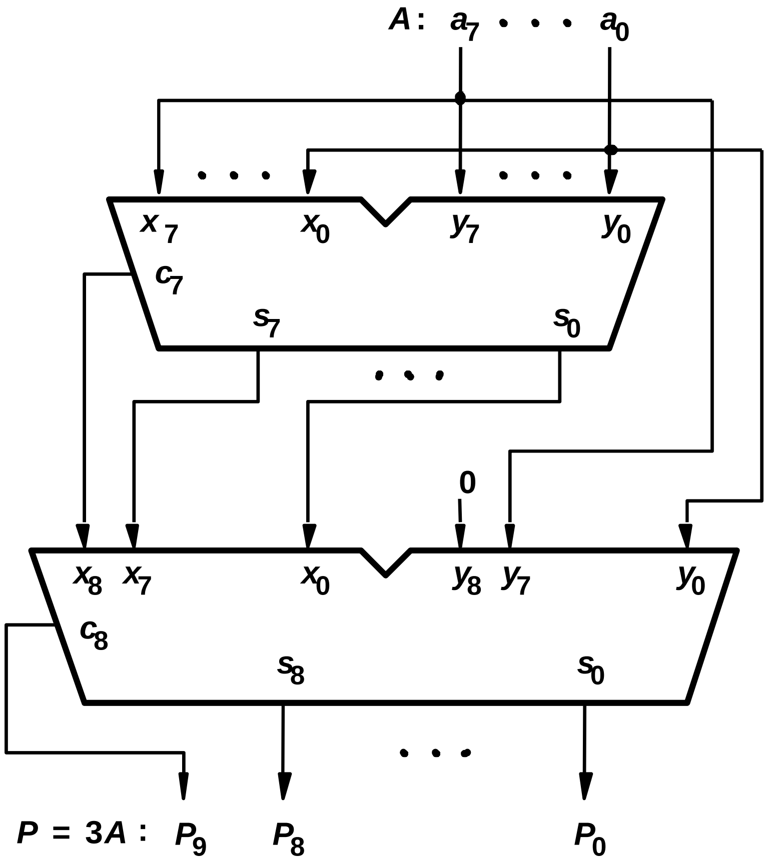
Ripple-Carry Adder

- Long delays!!

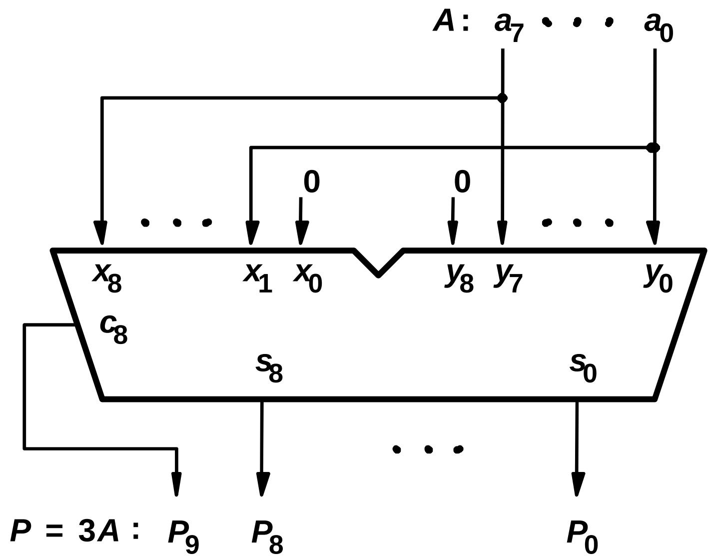


NTU ESOE

Design Example: P=3A

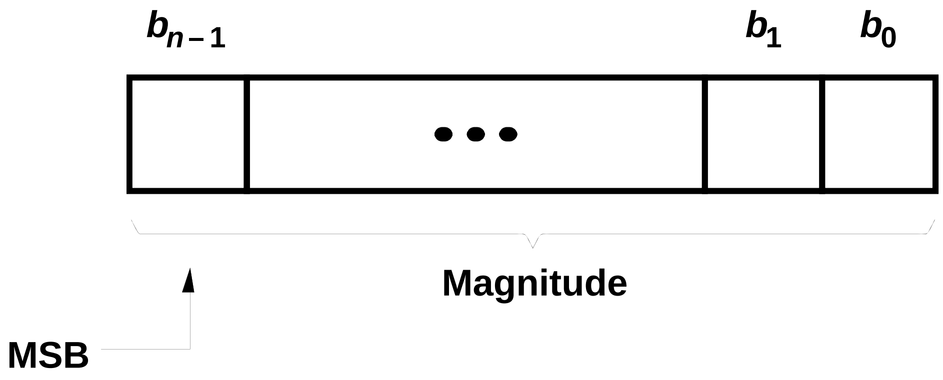


Naive approach

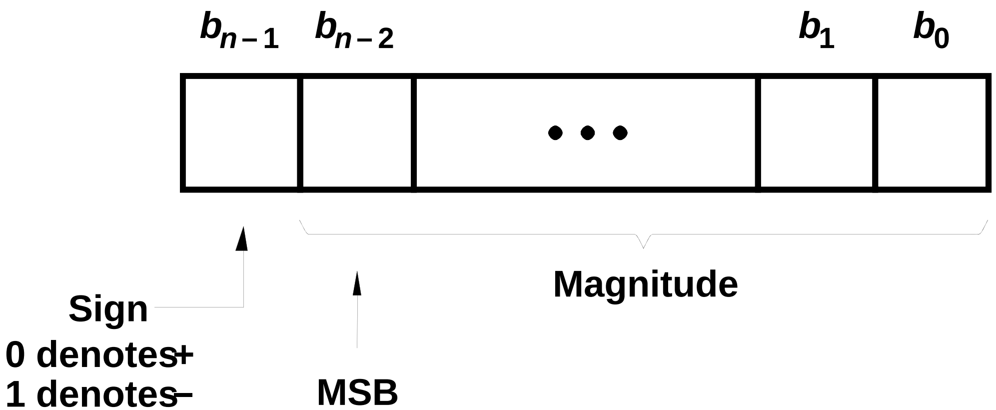


Efficient design

Signed Numbers



Unsigned number



Signed number

Representation of Negative Numbers

- Signed-Magnitude Representation
  - MSB is used as signed bit, and remaining bits as magnitude.
  - $01010101_2 = +85_{10}$, $11010101_2 = -85_{10}$
- 1's Complement Representation
  - Negates a number by taking its complement
  - $K=(2^n-1)-P$, e.g. n=4, converting +5 into -5; $K=(2^4-1)-P=1111-0101=1010$
  - Complemented twice leads to the original number
  - Can be obtained by simply by complementing each bit of the number.
- 2's Complement Representation
  - $K=2^n-P$,
  - e.g. n=4, converting +5 into -5; $K=2^4-P=10000-0101=1011$

Rule for Finding 2's Complements

- Given a signed number, $B = b_{n-1}\, b_{n-2} \ldots\, b_1\, b_0$, its 2's complement $K = k_{n-1}\, k_{n-2} \ldots\, k_1\, k_0$
- Examining the bits of *B* from right to left and taking the following action: Copy all bits of *B* that are 0 and the first bit that is 1, then simply complement the rest of the bits.
- e.g. If *B* = 0110, then we copy k_0=b_0=0 and k_1=b_1=1, and complement the rest that k_2=b_2'=0 and k_3=b_3'=1. Hence *K*=1010.
- If *B*=10110100 then *K*=01001100.

Interpretation of Four-bit Signed Integers

$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	−0	−7	−8
1001	−1	−6	−7
1010	−2	−5	−6
1011	−3	−4	−5
1100	−4	−3	−4
1101	−5	−2	−3
1110	−6	−1	−2
1111	−7	−0	−1

Addition and Subtraction

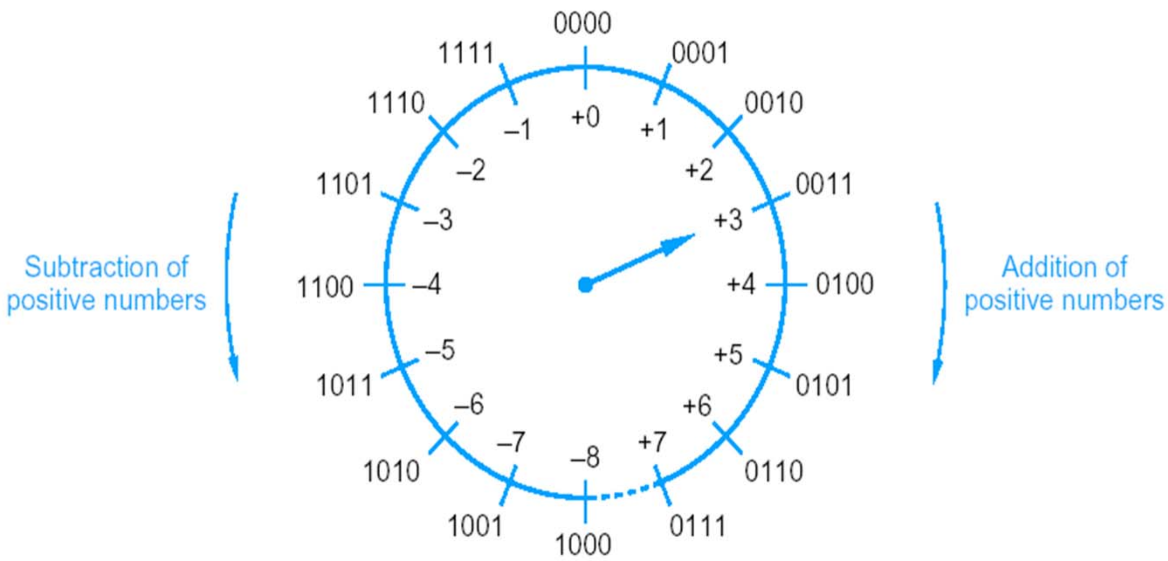
- Sign-and-magnitude addition
  - If both operands have the same sign, then magnitudes are added, and the resulting sum is given the sign of the operands.
  - If the operands have opposite sign, then it is necessary to subtract the smaller number from the larger one.
  - This means that logic circuits that compare and subtract numbers are also needed
  - For this reason, the sign-and-magnitude representation is not used in computers.

1's Complement Addition Example

4-bit example: +3	0011	+4	0100
+ +4	+ 0100	+ -7	+ 1000
+7	0111	-3	1100
-2	1101	+6	0110
+ -5	+ 1010	+ -3	+ 1100
-7	10111	+3	1 0010
+	1	+	1
	1000		0011

2's Complement Addition Rule

- Addition as unsigned binary addition except ignoring all carry beyond MSB.
- 4-bit example -2 1110
- + -6 + 1010
- -8 11000



Overflow of 2's Complement Addition

- If an addition operation produces a result that exceeds the range of the number system.
- 4-bit example:

-3	1101	+5	0101
+ -6	+ 1010	++6	+0110
-9	10111 = +7	+11	1011 = -5

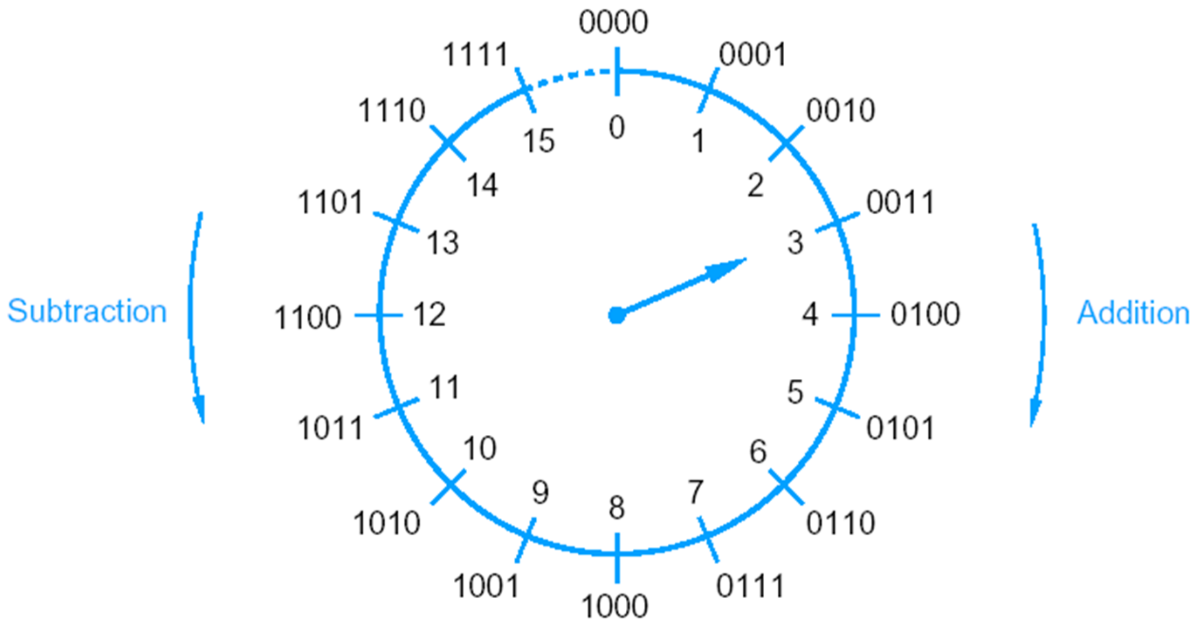
- Detection rule of overflow: signs of the addends are the same and the sign of the sum is different.
- Overflow = $c_3\overline{c_4} + \overline{c_3}c_4 = c_3 \oplus c_4$
- *n*-bit numbers : Overflow = $c_{n-1} \oplus c_n$

2's Complement Subtraction Rule

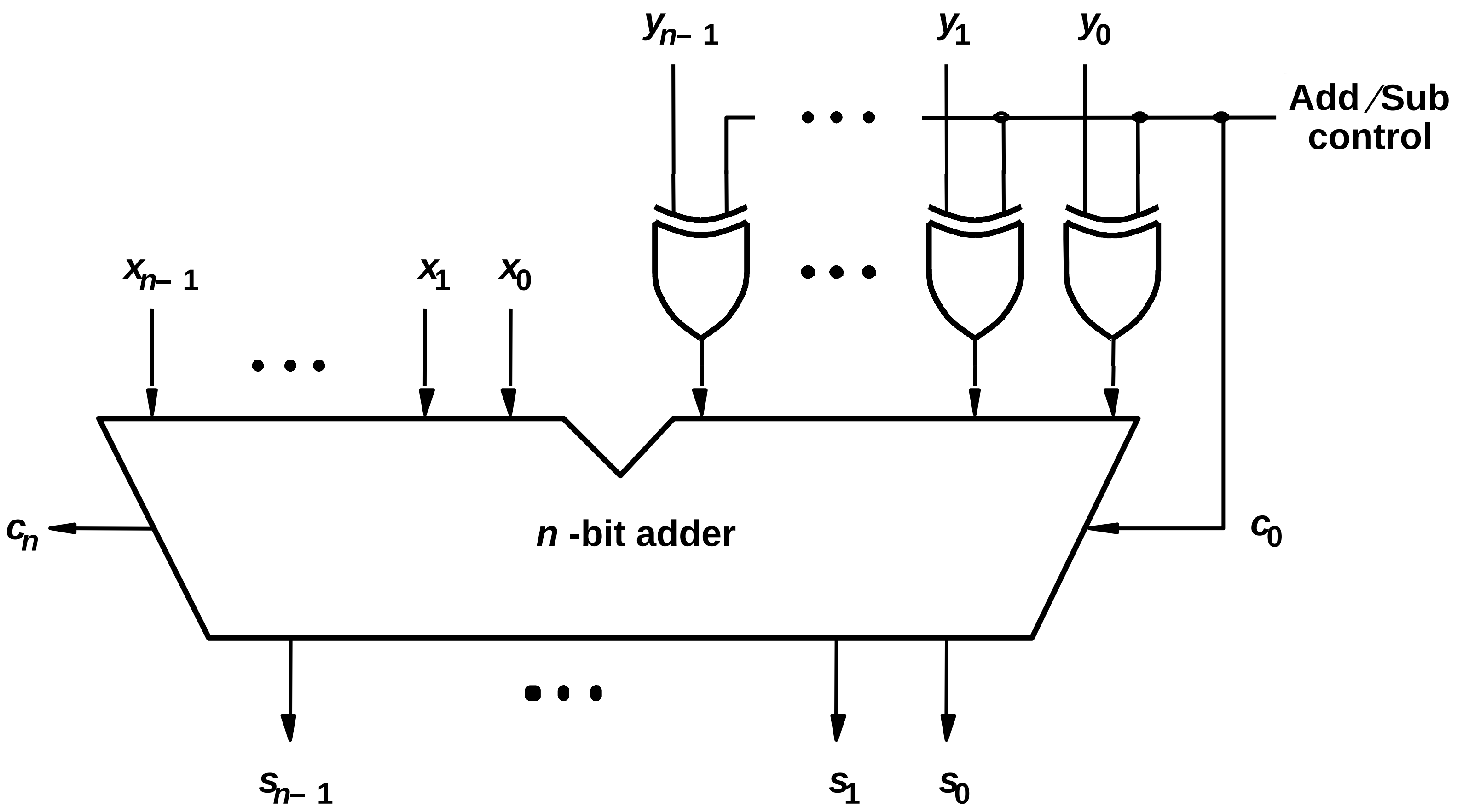
- 2's complement numbers may be subtracted as if they were ordinary unsigned binary numbers, and appropriate rules for detecting overflow.
- However, most subtraction circuits negate the subtrahend by taking it 2's complement and then add it to the minuend using normal rule of addition.
- 1 – c_{in}
- 4-bit example: +4 0100 0100
- \- +3 - 0011 + 1100
- +1 1 0001

2's Complement and Unsigned Binary System

- 2's complement and unsigned binary system can share the same adder circuits, however the results are interpreted differently.
- In unsigned binary system, carry on MSB indicates out-of range; while in signed, the overflow rule stated indicates out-of-range.



Adder/ Subtractor Unit



Summary of Addition and Subtraction

Number System	Addition Rules	Negation Rules	Subtraction Rules
Unsigned	Add the numbers. Result is out of range if a carry out of the MSB occurs.	Not applicable	Subtract the subtrahend from the minuend. Result is out of range if a borrow out of the MSB occurs.
Signed magnitude	(same sign) Add the magnitudes; overflow occurs if a carry out of MSB occurs; result has the same sign. (opposite sign) Subtract the smaller magnitude from the larger; overflow is impossible; result has the sign of the larger.	Change the sign bit.	Change the sign bit of the subtrahend and proceed as in addition.
Two's complement	Add, ignoring any carry out of the MSB. Overflow occurs if the carries into and out of MSB are different.	Complement all bits of the subtrahend; add 1 to the result.	Complement all bits of the subtrahend and add to the minuend with an initial carry of 1.
Ones' complement	Add; if there is a carry out of the MSB, add 1 to the result. Overflow if carries into and out of MSB are different.	Complement all bits of the subtrahend.	Complement all bits of the subtrahend and proceed as in addition.

Radix-Complement Scheme

- Complement of n digits D: $r^n - D$.
- Complement: complementing each digits and adding 1, because $r^n - D = ((r^n - 1) - D) + 1$ and $(r^n - 1) - D$ is obtained by complementing each digits.
- e.g. complement of 1849 is 8150 + 1 = 8151.
- The idea of performing a subtraction operation by addition of a complement of the subtrahend is not restricted to binary numbers.
- 74-36 = 74+100-100-36 = 74+(100-36)-100

Performance Issues

- A commonly used indicator of the value of a system is its *price/performance ratio.*
- The addition and subtraction of numbers are fundamental operations that are performed frequently, and the speed with which these operations are performed has a strong impact on the overall performance of a computer.
- The speed of any circuit is limited by the longest delay along the paths through the circuit.
- The longest delay on a ripple-carry adder is along the path from the y_i input, through the XOR and through they carry circuit of each adder stage, the *critical-path delay.*

Carry-Lookahead Adder

- The carry-out function for stage i can be realized as

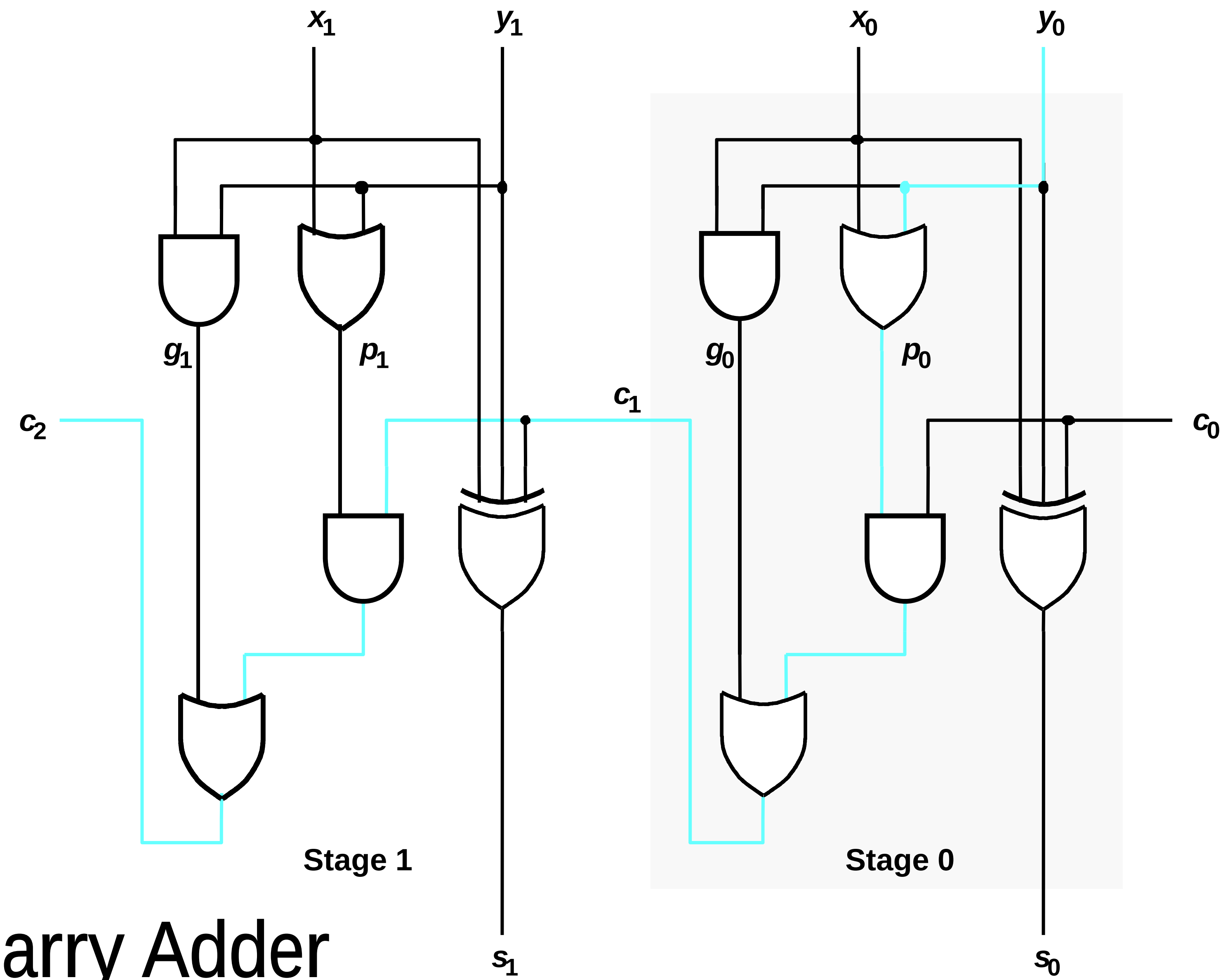
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

- Re-factored

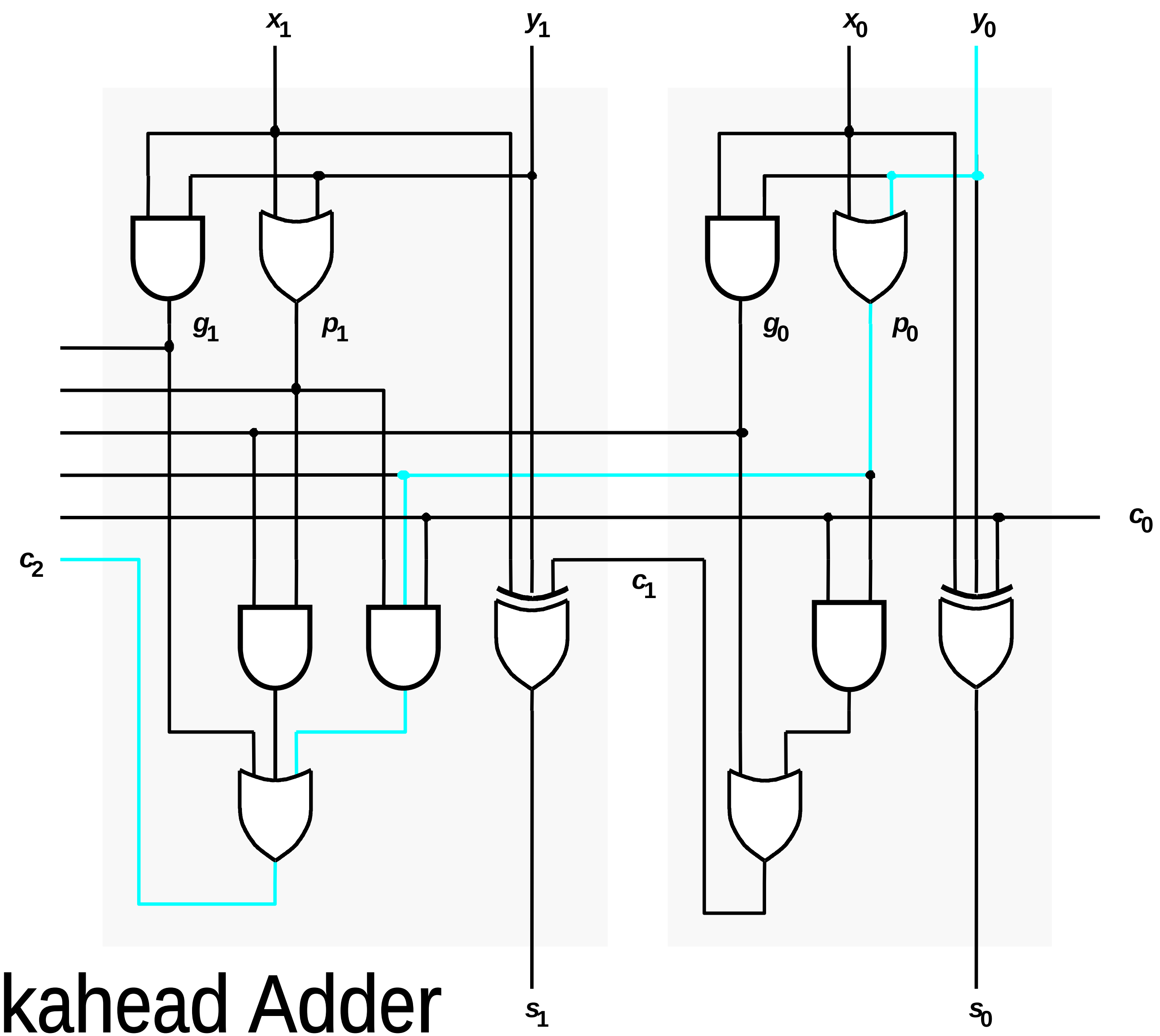
$$c_{i+1} = x_i y_i + (x_i + y_i) c_i$$
$$= g_i + p_i c_i$$

- $g_i = x_i y_i \qquad p_i = x_i + y_i$
- Expanding the carry adder of stage $i - 1$
- $c_{i+1} = g_i + p_i(g_{i-1} + P_{i-1} c_{i-1})$
- $= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots$

$$+ p_i p_{i-1} \ldots p_2 p_1 g_0 + p_i p_{i-1} \ldots p_1 p_0 c_0$$



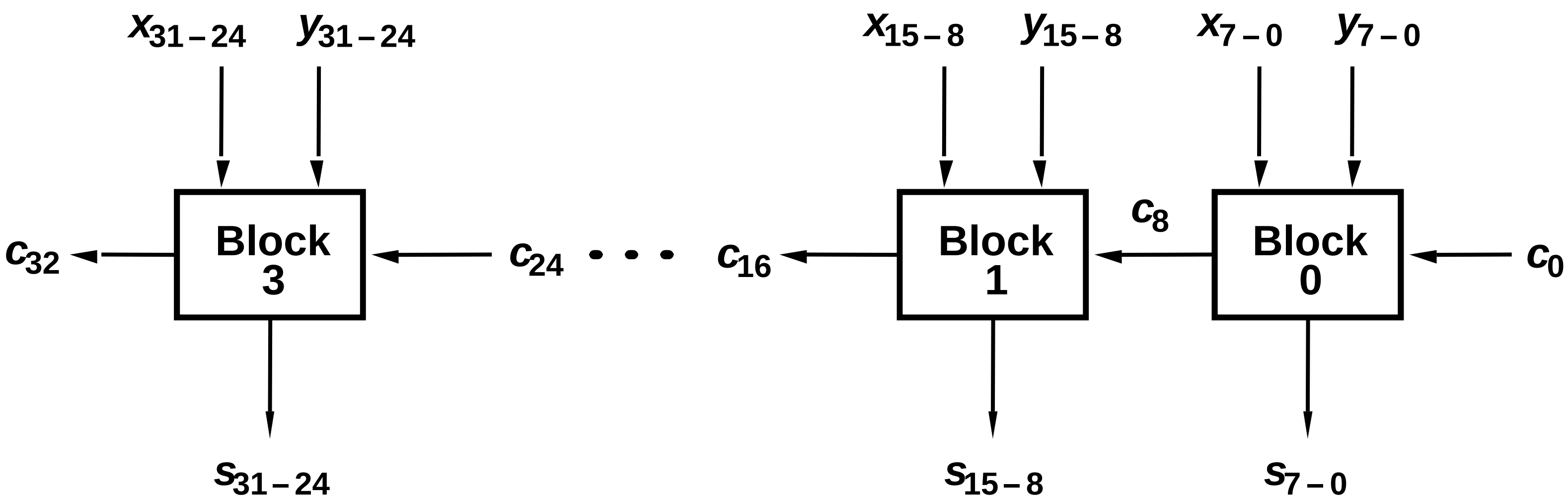
Ripple Carry Adder



Carry-Lookahead Adder

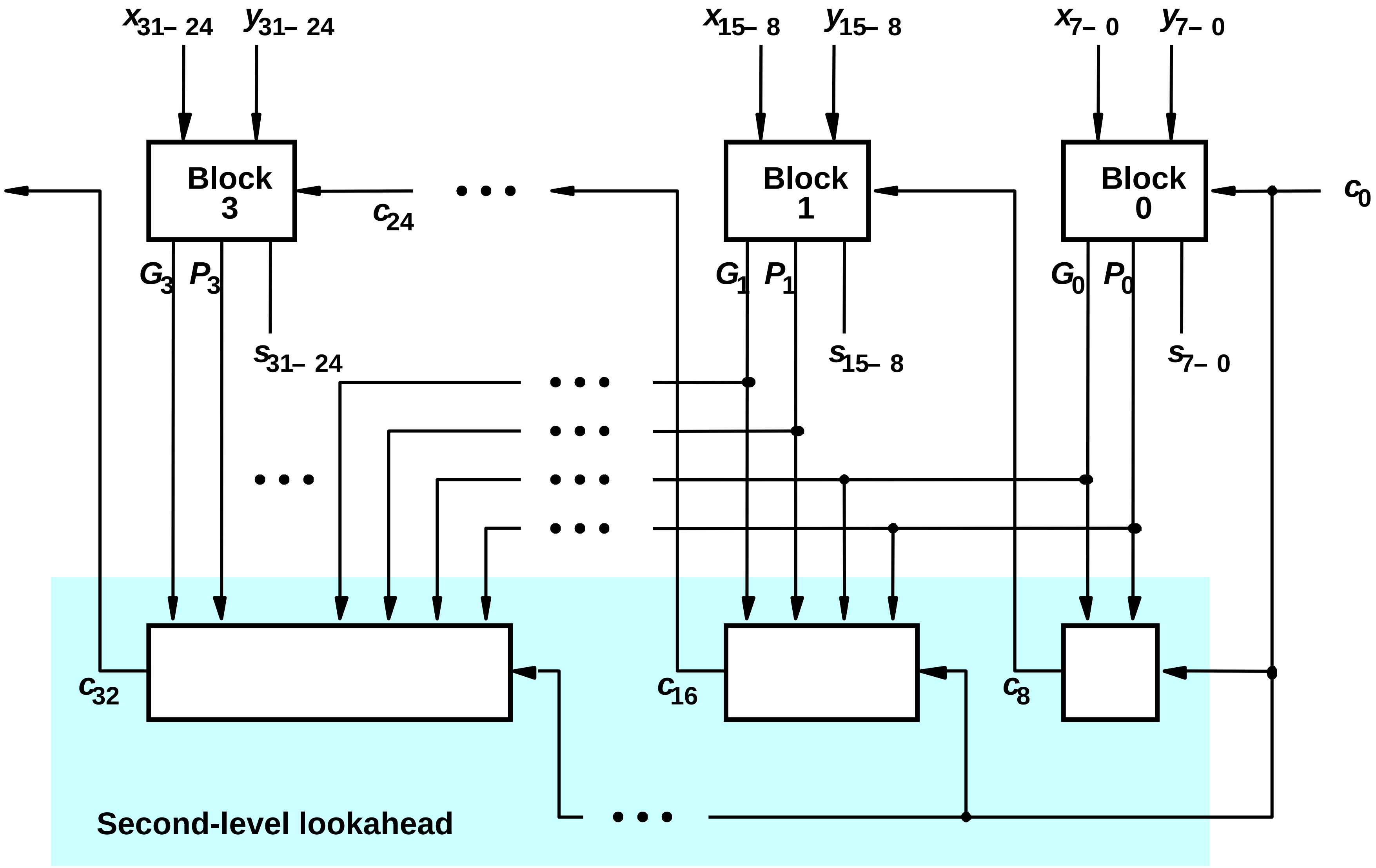
Hierarchical Carry-Lookahead adder with Ripple-carry between blocks

-

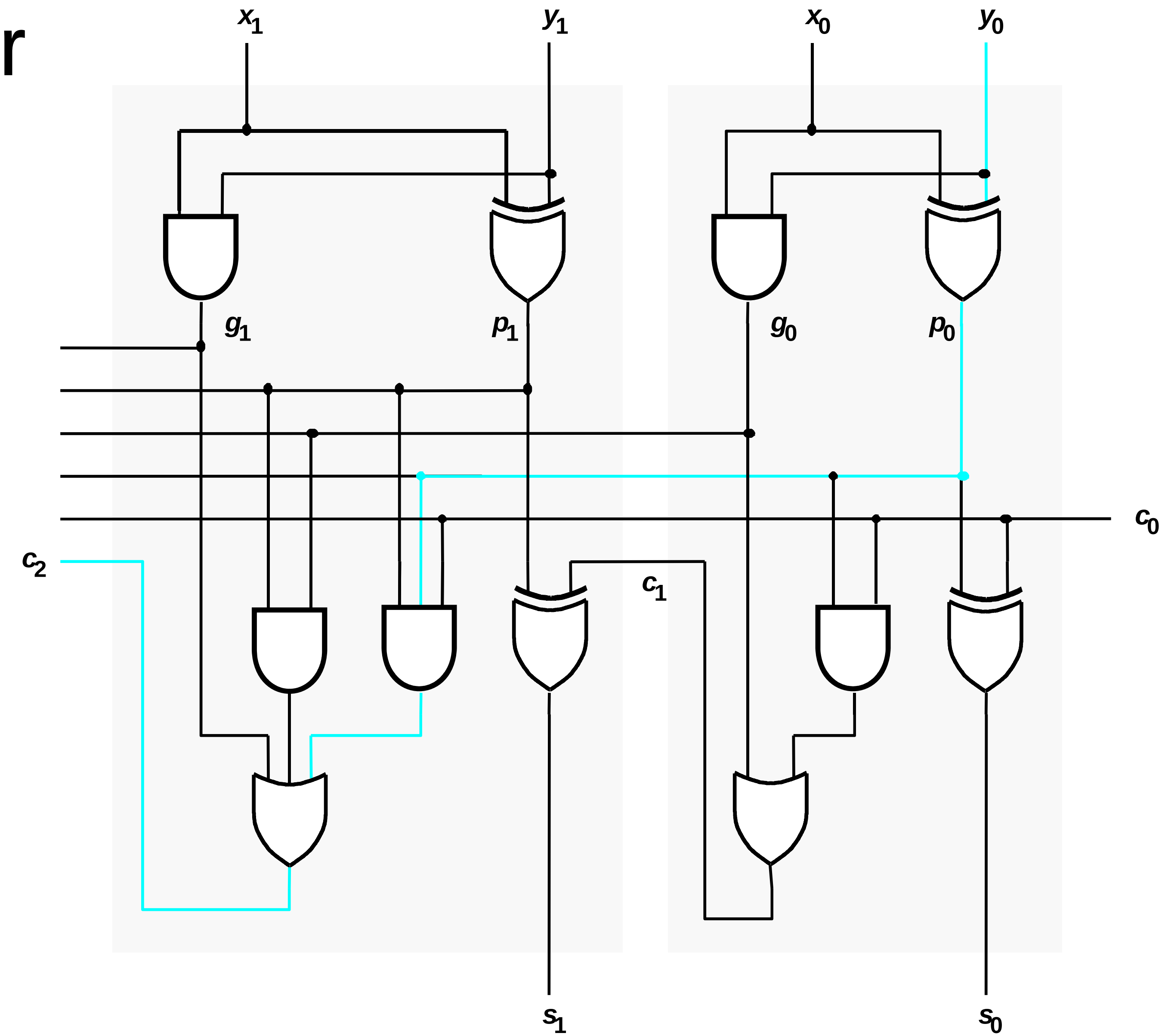


NTU ESOE

A Hierarchical Carry-Lookahead Adder



An Alternative Design for a Carry-Lookahead Adder



Multiplication by Hand

Multiplicand M	(14)	1 1 1 0
Multiplier Q	(11)	× 1 0 1 1
		1 1 1 0
		1 1 1 0
		0 0 0 0
		1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

Multiplicand M	(11)	1 1 1 0
Multiplier Q	(14)	× 1 0 1 1
Partial product 0		1 1 1 0
		+ 1 1 1 0
Partial product 1		1 0 1 0 1
		+ 0 0 0 0
Partial product 2		0 1 0 1 0
		+ 1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

Fast Multiplier

- 4x4 example, where the multiplicand and multiplier are $M=m_3m_2m_1m_0$, and Q= $q_3q_2q_1q_0$
- The partial product 0, $PP0 = pp0_3pp0_2pp0_1pp0_0$

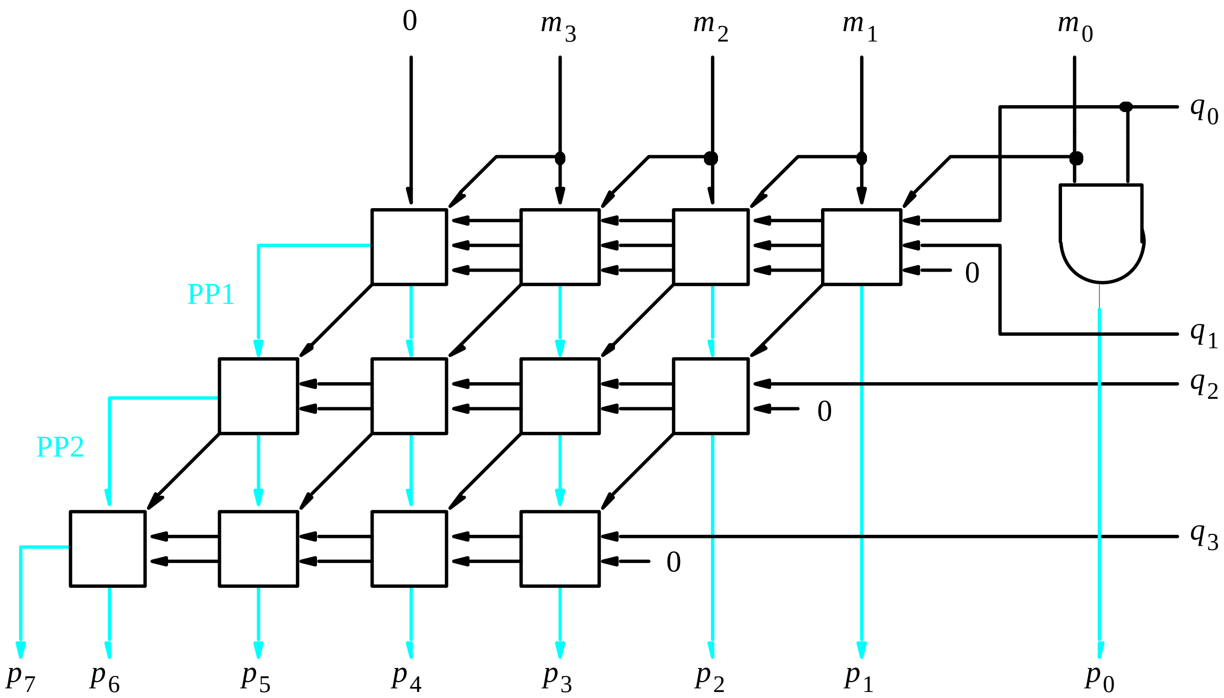
  $$PP0 = m_3q_0\, m_2q_0\, m_1q_0\, m_0q_0$$

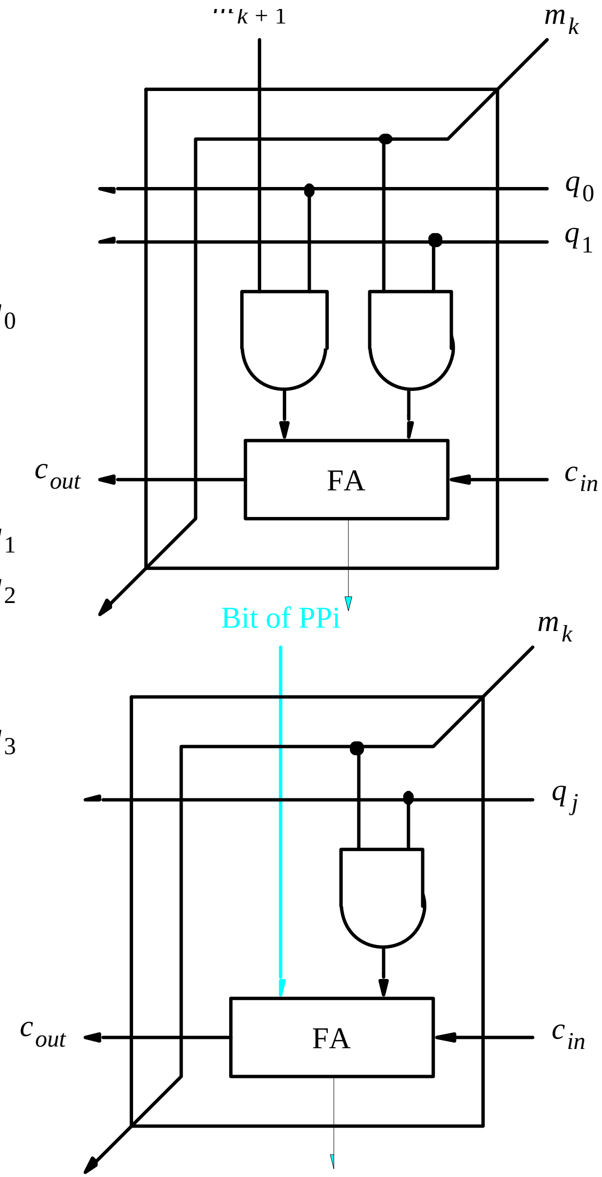
- The partial product 1, $PP1$

$$
\begin{array}{lccccc}
PP0: & 0 & pp0_3 & pp0_2 & pp0_1 & pp0_0 \\
 & +\, m_3q_1 & m_2q_1 & m_1q_1 & m_0q_1 & 0 \\
\hline
PP1: & pp1_4 & pp1_3 & pp1_2 & pp1_1 & pp1_0
\end{array}
$$

- The partial product 2, $PP2$ is generated using the AND of q_2 with M and adding to $PP1$, and so on.

Fast Multiplier Array Structure





NTU ESOE

Multiplication of Signed Numbers

- To avoid overflow, the new partial product must be larger by one extra bit

Multiplicand M	(+14)	0 1 1 1 0
Multiplier Q	(+11)	x 0 1 0 1 1
Partial product 0		0 0 0 1 1 1 0
		+ 0 0 1 1 1 0
Partial product 1		0 0 1 0 1 0 1
		+ 0 0 0 0 0 0
Partial product 2		0 0 0 1 0 1 0
		+ 0 0 1 1 1 0
Partial product 3		0 0 1 0 0 1 1
		+ 0 0 0 0 0 0
Product P	(+154)	0 0 1 0 0 1 1 0 1 0

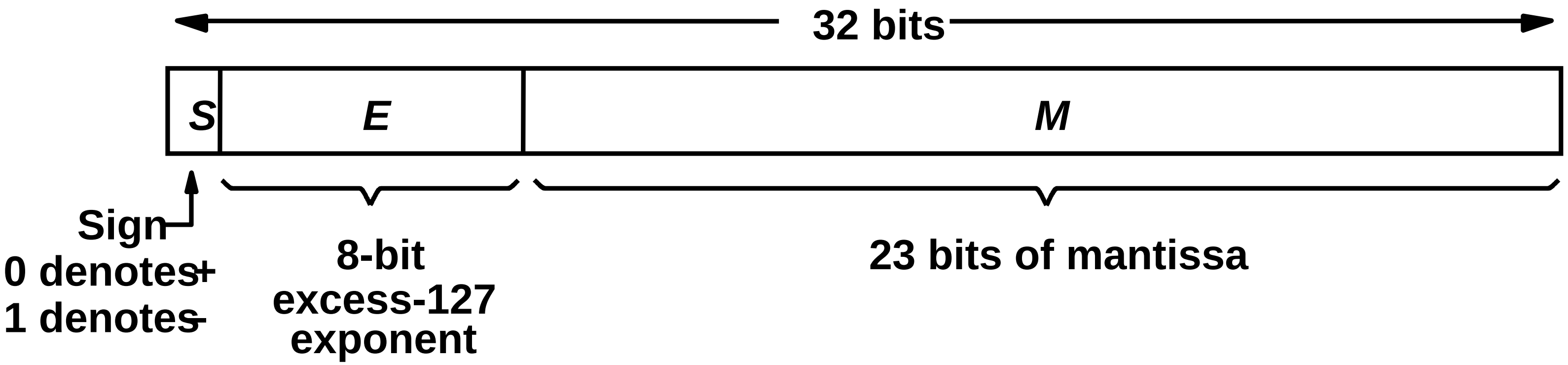
Multiplicand M	(–14)	1 0 0 1 0
Multiplier Q	(+11)	x 0 1 0 1 1
Partial product 0		1 1 1 0 0 1 0
		+ 1 1 0 0 1 0
Partial product 1		1 1 0 1 0 1 1
		+ 0 0 0 0 0 0
Partial product 2		1 1 1 0 1 0 1
		+ 1 1 0 0 1 0
Partial product 3		1 1 0 1 1 0 0
		+ 0 0 0 0 0 0
Product P	(–154)	1 1 0 1 1 0 0 1 1 0

Floating-Point Number

- Floating-point number = *Mantissa* x $R^{exponent}$
- Binary floating-point representation has been standardized by IEEE standard.

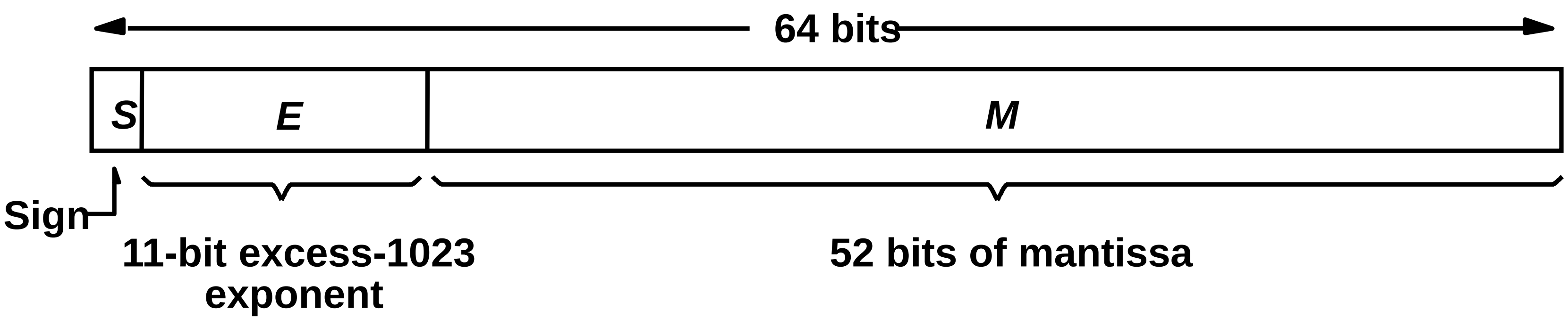
Single-Precision Floating-Point Format

- The IEEE standard specifies the exponent in the excess-127 format : *Exponent* = *E-127.* In this way, *E* becomes a positive integer
- Value = $\pm 1.M \times 2^{E-127}$



Double-Precision Floating-Point Format

- The IEEE standard specifies the exponent in the excess-1023 format : *Exponent = E-1023.* In this way, *E* becomes a positive integer
- Value = $\pm 1.M \times 2^{E\text{-}1023}$



Binary Coded Decimal Representation

- A decimal digit is coded as a 4-bit binary string

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

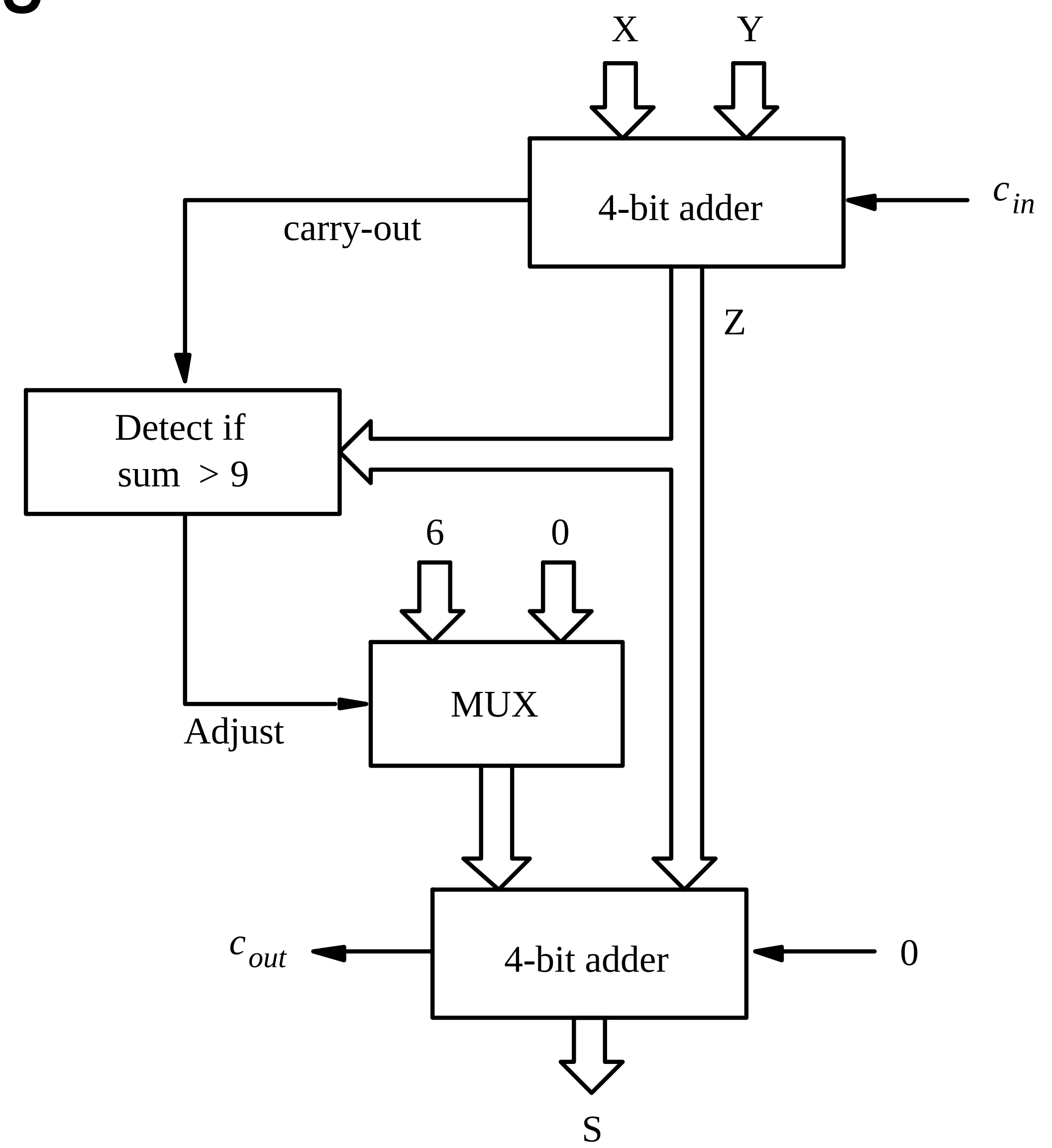
Addition of BCD digits

- Need a block that detects weather $Z > 9$, Adjust, which controls the multiplexer that provides the correction (+ 6) when needed .

$$
\begin{array}{r r r}
X & 0\,1\,1\,1 & 7 \\
+\ Y & +\ 0\,1\,0\,1 & +\ 5 \\
\hline
Z & 1\,1\,0\,0 & 12 \\
 & +\ 0\,1\,1\,0 & \\
\cline{2-2}
\text{carry} \rightarrow & 1\,\underbrace{0\,0\,1\,0}_{S\,=\,2} &
\end{array}
$$

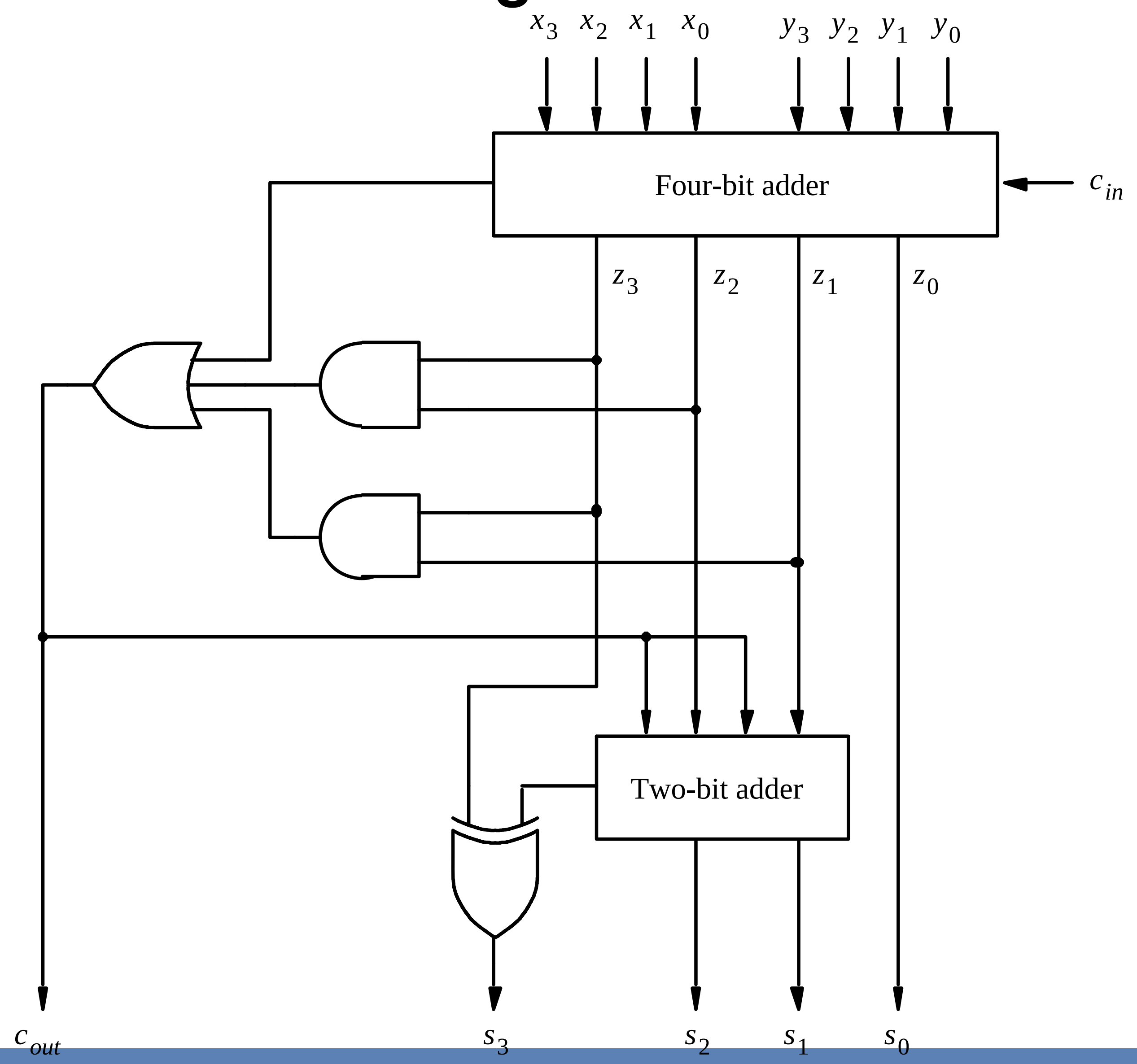
$$
\begin{array}{r r r}
X & 1\,0\,0\,0 & 8 \\
+\ Y & +\ 1\,0\,0\,1 & +\ 9 \\
\hline
Z & 1\,0\,0\,0\,1 & 17 \\
 & +\ 0\,1\,1\,0 & \\
\cline{2-2}
\text{carry} \rightarrow & 1\,\underbrace{0\,1\,1\,1}_{S\,=\,7} &
\end{array}
$$

One-digit BCD Adder



NTU ESOE

Circuit for a one-digit BCD Adder



ASCII Character Code

Bit positions 3210	Bit positions 654: 000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	1	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL

NUL	Null/Idle	SI	Shift in
SOH	Start of header	DLE	Data link escape
STX	Start of text	DC1-DC4	Device control
ETX	End of text	NAK	Negative acknowledgement
EOT	End of transmission	SYN	Synchronous idle
ENQ	Enquiry	ETB	End of transmitted block
ACQ	Acknowledgement	CAN	Cancel (error in data)
BEL	Audible signal	EM	End of medium
BS	Back space	SUB	Special sequence
HT	Horizontal tab	ESC	Escape
LF	Line feed	FS	File separator
VT	Vertical tab	GS	Group separator
FF	Form feed	RS	Record separator
CR	Carriage return	US	Unit separator
SO	Shift out	DEL	Delete/Idle

Bit positions of code format = $\boxed{6}\boxed{5}\boxed{4}\boxed{3}\boxed{2}\boxed{1}\boxed{0}$