Lecture 5 key concepts

- Decimal value of a number represented in 2’s complement
- Sign extension in 2’s complement representation
- 2’s complement addition and subtraction: concept and circuit implementation
- Arithmetic overflow: concept and detection

Which concepts are unclear to you after viewing L5?

A. 2’s complement
B. Sign extension
C. 2’s complement add/subtract
D. Arithmetic overflow
E. None

Give the decimal value of this signed number represented in 2’s complement: 0101010

A. +38

B. +40

C. +42

D. +44

Give the decimal value of this signed number represented in 2’s complement: 101010

A. -22

B. -24

C. -26

D. -28

Sign extension in 2's comp

When more bits are available than needed to represent a numerical value:

Unsigned representation	Fill msbs with 0
Sign-magnitude representaion	Only MSB is sign bit. The rest are plain magnitude.
2's complement representation	Must apply sign extension.

Examples

	4-bit	8-bit	Decimal
Unsigned	1101	0000 1101 (0D hex)	13
Sign-magnitude	0101 1101	0000 0101 (05 hex) 1000 0101 (85 hex)	+5 -5
2's complement	0101 1101	0000 0101 (05 hex) 1111 1101 (FD hex)	+5 -3

Sign extension (cont)

Show that these numbers in 2's complement representation have the same value.

(a) 1001 (b) 11001 (c) 111001

1001	11001	111001
1000 + 0001	10000 + 01001	100000 + 011001
-8 + 1 (dec)	-16 + 9 (dec)	-32 + 25 (dec)
-7 (dec)	-7 (dec)	-7 (dec)

2's complement representation and sign extension

1	0	0	0	0	0	0	0	0	0
1									
1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	0	0	1
1	1	1	1	1	1	1	0	1	0
1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	1	1
0									
0	1	1	1	1	1	1	1	1	1

2-bit

2’s complement representation and sign extension

1	0	0	0	0	0	0	0	0	0
1									
1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	0	0	1
1	1	1	1	1	1	1	0	1	0
1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	1	1
0									
0	1	1	1	1	1	1	1	1	1

3-bit

2’s complement representation and sign extension

1	0	0	0	0	0	0	0	0	0
1									
1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	0	0	1
1	1	1	1	1	1	1	0	1	0
1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	1	1
0									
0	1	1	1	1	1	1	1	1	1

4-bit

2’s complement representation and sign extension

1	0	0	0	0	0	0	0	0	0
1									
1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	0	0	1
1	1	1	1	1	1	1	0	1	0
1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	1	1
0									
0	1	1	1	1	1	1	1	1	1

5-bit

2’s complement representation and sign extension

1	0	0	0	0	0	0	0	0	0
1									
1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	0	0	1
1	1	1	1	1	1	1	0	1	0
1	1	1	1	1	1	1	0	1	1
			1	1	1	1	1	0	0
			1	1	1	1	1	0	1
			1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	1	1
0									
0	1	1	1	1	1	1	1	1	1

Sign extend

6-bit

F56-12

Which one of the following 8-bit 2’s complement arithmetic operations will result in overflow?

A. 19h + A0h
B. 90h – 8Ch
C. 39h – A0h

h denotes hexadecimal

2’s comp addition example

Perform this signed 8-bit addition in 2’s complement representation.

19(hex) + A0(hex) : no overflow

			unsigned
	0 0 0 1 1 0 0 1	+25 (dec)	25 (dec)
+	1 0 1 0 0 0 0 0	-96 (dec)	160 (dec)
0	1 0 1 1 1 0 0 1	-71 (dec)	185 (dec)

For machine

Human interpretation

2’s comp subtraction example

Perform this signed 8-bit subtraction in 2’s complement representation.

90(hex) – 8C(hex) : no overflow

			unsigned
	1 0 0 1 0 0 0 0	-112 (dec)	144 (dec)
	0 1 1 1 0 0 1 1	+115 (dec)	115 (dec)
+	1	+1 (dec)	1 (dec)
1	0 0 0 0 0 1 0 0	+4 (dec)	260 (dec)

For machine

Human interpretation

2’s comp overflow example

Perform this signed 8-bit subtraction in 2’s complement representation.

39(hex) – A0(hex)

			unsigned
	0 0 1 1 1 0 0 1	+57 (dec)	57 (dec)
	0 1 0 1 1 1 1 1	+95 (dec)	95 (dec)
+	1	+1 (dec)	1 (dec)
0	1 0 0 1 1 0 0 1	-103 (dec)	153 (dec)

Overflow has occurred because +153 (dec) cannot be represented in 8 bits 2’s comp

2’s comp Arithmetic Overflow

- Addition and subtraction are both performed as addition in the circuit
- Adding 2 values of opposite signs will NEVER lead to overflow
- Adding 2 values of same sign may lead to overflow
- Overflow is detected when the sign bit of the result is different from that of the values being added

End of L5 summary

Lecture 6 key concepts

- Combined circuit with addition and subtraction
- 4-bit parallel adder with registers
- Binary multiplication
- BCD addition and correction

Which concepts are unclear to you after viewing L6?

A. Circuit for add/subtract

B. Parallel addition with registers

C. Multiplication

D. BCD addition

E. None

Subtraction performed as addition

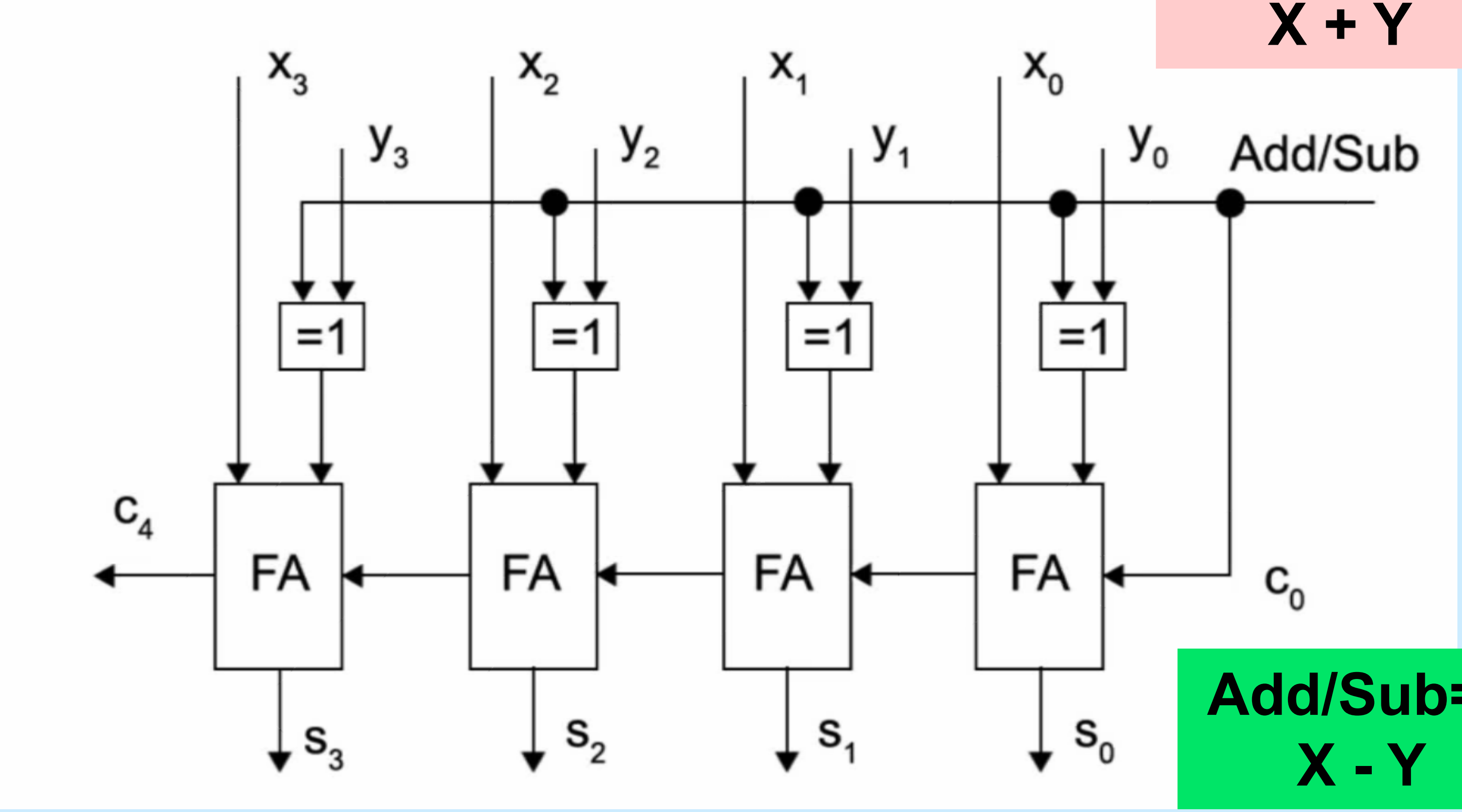
X - Y = X + (-Y)

- -Y is simply the 2’s complement of Y
- X, Y can be negative or positive
- 2’s complement of Y can be easily obtained by adding 1 to the 1’s complement of Y
- 1’s complement of Y is easily obtained by inverting each bit of Y
- Easy for digital circuits

Addition/Subtraction

Combined add/subtract circuit

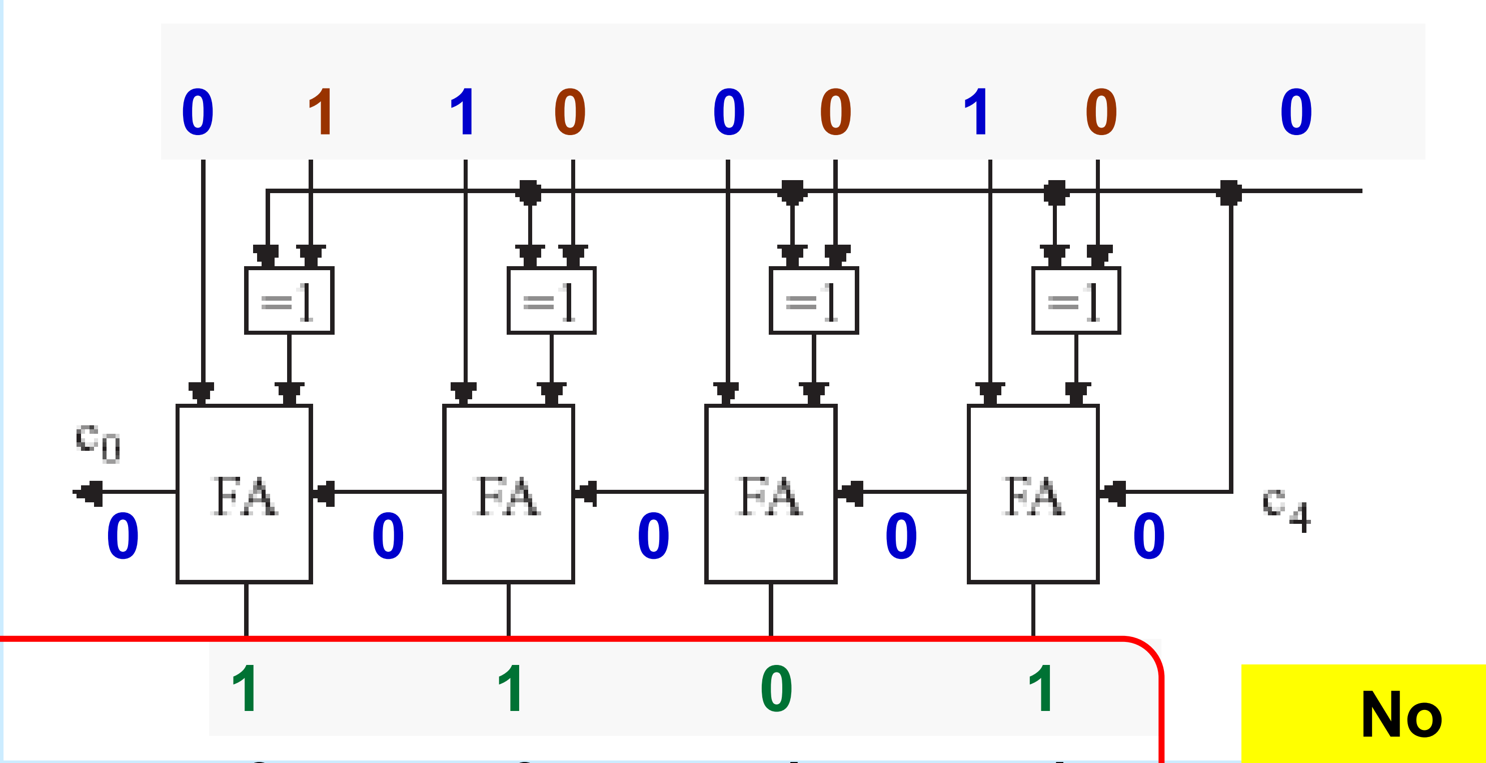
Add/Sub=0
X + Y



Add/Sub=1
X - Y

Addition example

Example 1: +5(dec) + -8(dec) = -3(dec)

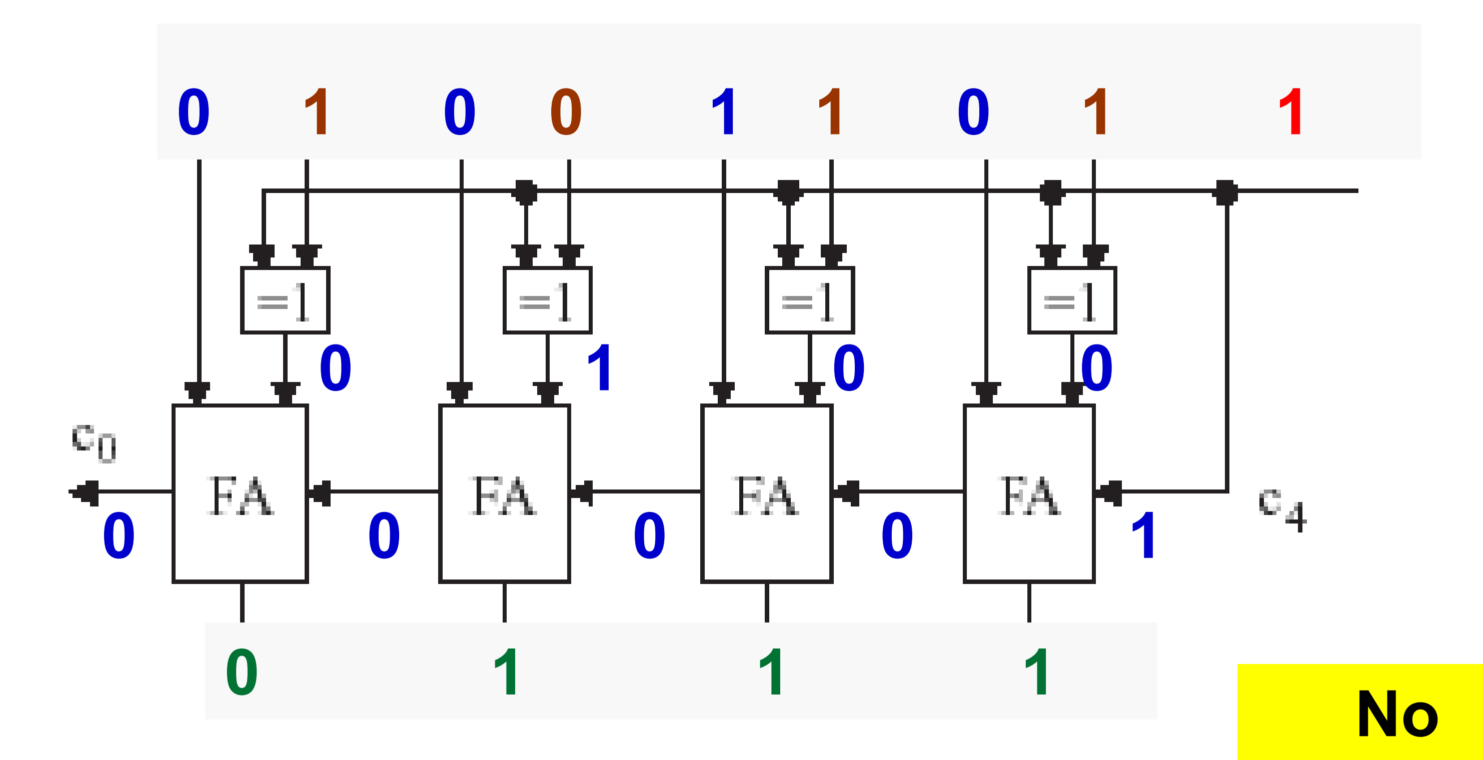


2's comp: 0 0 1 1

No overflow

Subtraction example

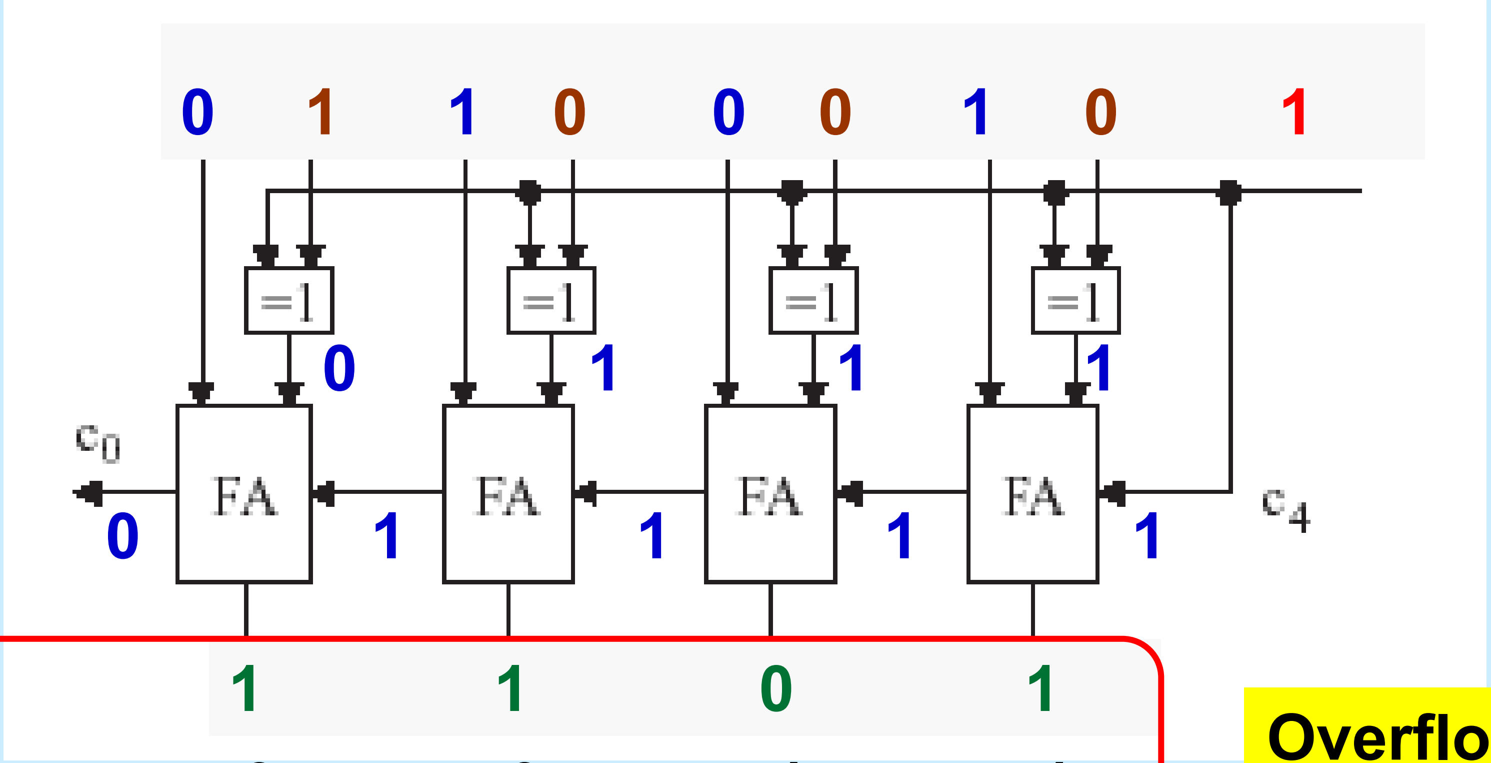
Example 2: 2(dec) - -5(dec) = +7(dec)



No overflow

Subtraction example (overflow)

Example 3: +5(dec) - -8(dec) = -3(dec)?



2's comp: 0 0 1 1

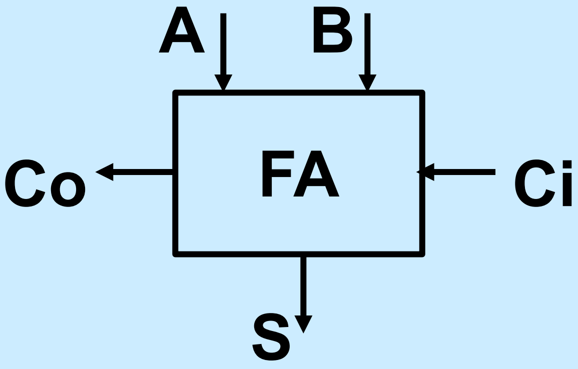
Overflow

Overflow detection

Same method regardless of addition or subtraction.

Check MSB Full Adder:

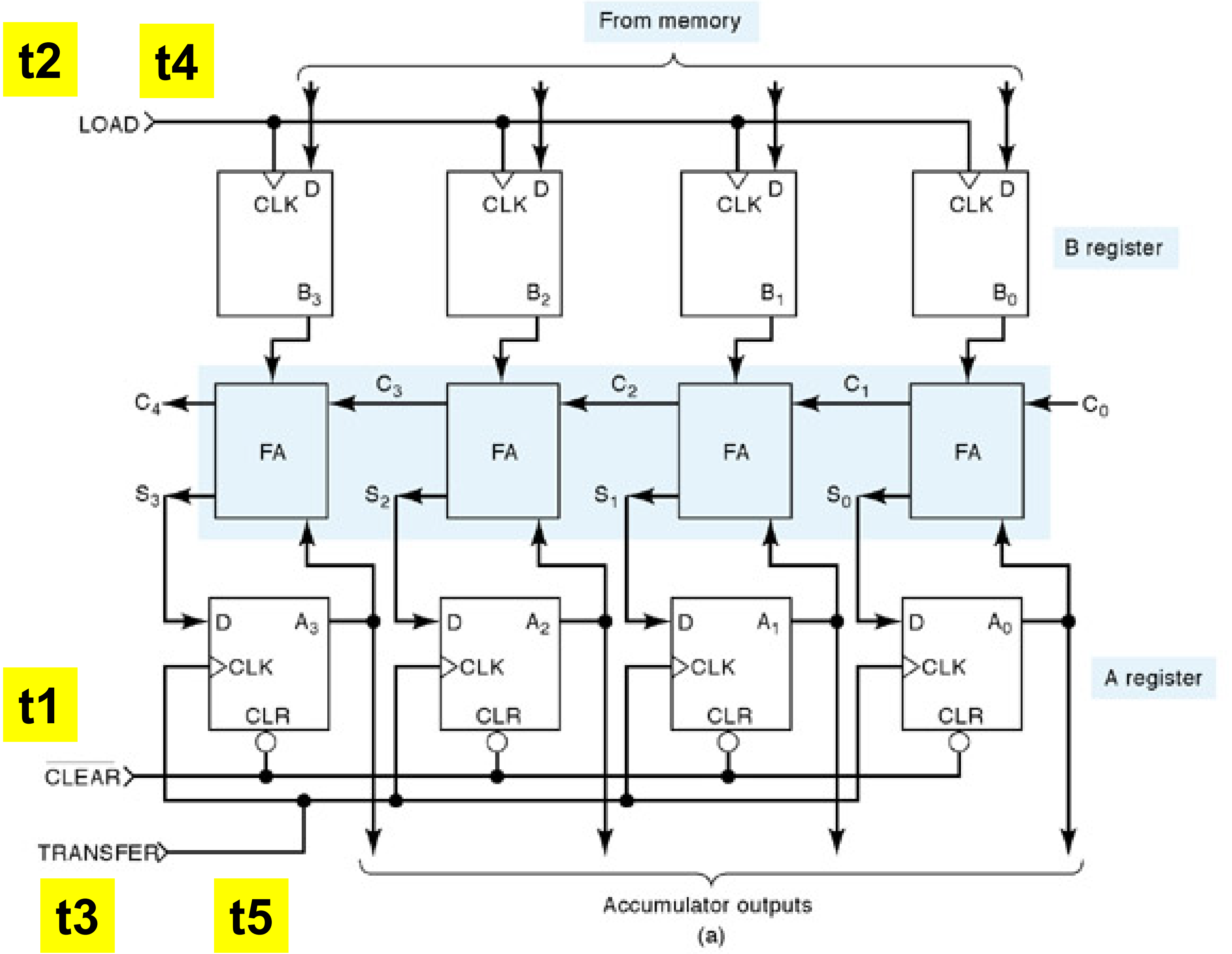
- No overflow if A ≠ B
- Overflow if A = B but ≠ S



V = (A XNOR B) AND (A XOR S)

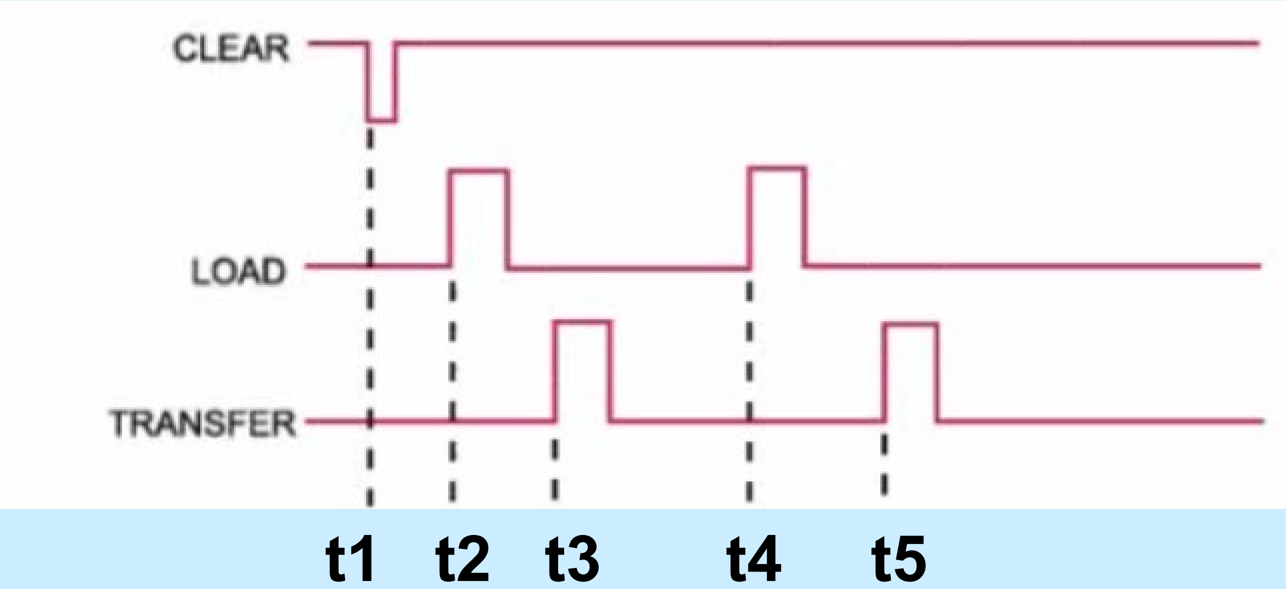
- Optional exercise: use a truth table to show that

V = Co XOR Ci


t2
t4
From memory
LOAD
CLK
D
B_3
B_2
B_1
B_0
B register
C_4
C_3
C_2
C_1
C_0
FA
S_3
S_2
S_1
S_0
D
A_3
A_2
A_1
A_0
CLK
CLR
A register
t1
CLEAR
TRANSFER
t3
t5
Accumulator outputs
(a)

Parallel adders with registers

Let x be the 1st value loaded from memory, y be the 2nd



What will be the FA’s output

- shortly before t4?
- shortly after t5?

Unsigned binary multiplication

Multiplication – shift left (x2) and add

1x 101

2x 101

4x 101

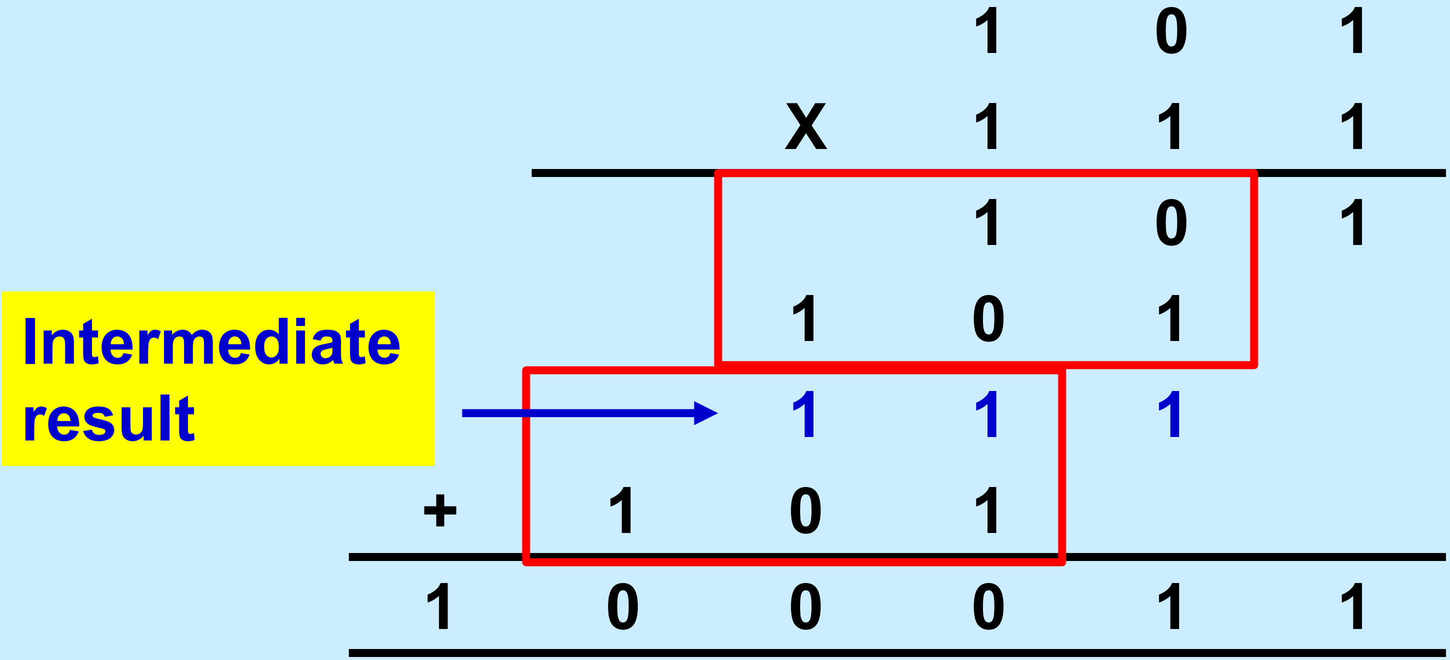
7x 101

$$
\begin{array}{cccccc}
 & & & 1 & 0 & 1 \\
 & & \text{X} & 1 & 1 & 1 \\
\hline
 & & & 1 & 0 & 1 \\
 & & 1 & 0 & 1 & \\
+ & 1 & 0 & 1 & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 \\
\hline
\end{array}
$$

Do we need a 5-bit adder?

Unsigned multiplication (cont)

Multiplication – shift left (x2) and add



```
            1   0   1
        X   1   1   1
    ---------------------
            1   0   1
        1   0   1
        1   1   1
+   1   0   1
-------------------------
1   0   0   0   1   1
-------------------------
```

Intermediate result

We need a 3-bit adder plus registers

2’s complement multiplication

Multiplication – shift left (x2) and add

					1	0	1
-1				X	1	1	1
1x 101		1	1	1	1	0	1
2x 101		1	1	1	0	1	
-4x 101	+	0	0	1	1		
-1x 101		0	0	0	0	1	1

-1 = -4 + 3

101x(-1) = 101x(-4+3) = 101x(-4) + 101x(+3)

Binary division

Basically the reverse of multiplication: by shift and subtract.

Binary division will not be tested

http://courses.cs.vt.edu/~cs1104/Division/ShiftSubtract/Shift.Subtract.html

https://www.calculatorsoup.com/calculators/math/longdivision.php

BCD addition example

BCD addition/correction (479 + 461 = 940)

	0 1 0 0	0 1 1 1	1 0 0 1
+	0 1 0 0	0 1 1 0	0 0 0 1
	1 0 0 0	1 1 0 1	1 0 1 0

Without correction

BCD addition example

BCD addition/correction (479 + 461 = 940)

$$
\begin{array}{lccc}
 & 1 & 1 & \\
 & 0\,1\,0\,0 & 0\,1\,1\,1 & 1\,0\,0\,1 \\
+ & 0\,1\,0\,0 & 0\,1\,1\,0 & 0\,0\,0\,1 \\
\hline
 & 1\,0\,0\,1 & 1\,1\,1\,0 & 1\,0\,1\,0 \\
+ & 0\,0\,0\,0 & 0\,1\,1\,0 & 0\,1\,1\,0 \\
\hline
 & 1\,0\,0\,1 & 0\,1\,0\,0 & 0\,0\,0\,0
\end{array}
$$

With correction

What is the total number of FAs needed for 3-digit BCD addition (including possible correction)?

A. 24
B. 21
C. 18
D. 15

Results that need correction

Decimal	Cout	S3	S2	S1	S0
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0

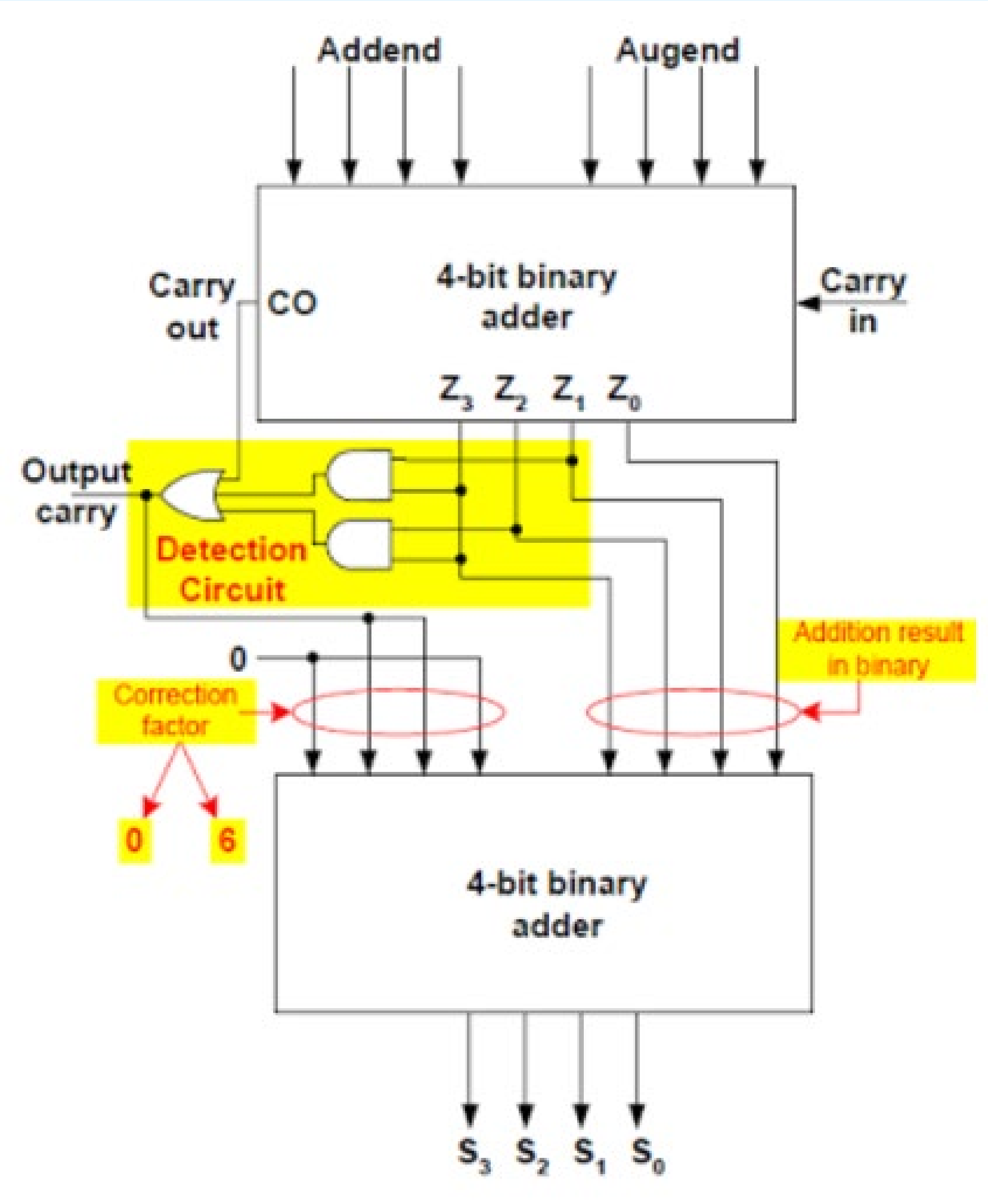


Image from http://implement-logic.blogspot.sg/2011/11/bcd-adder.html

End of L6 summary