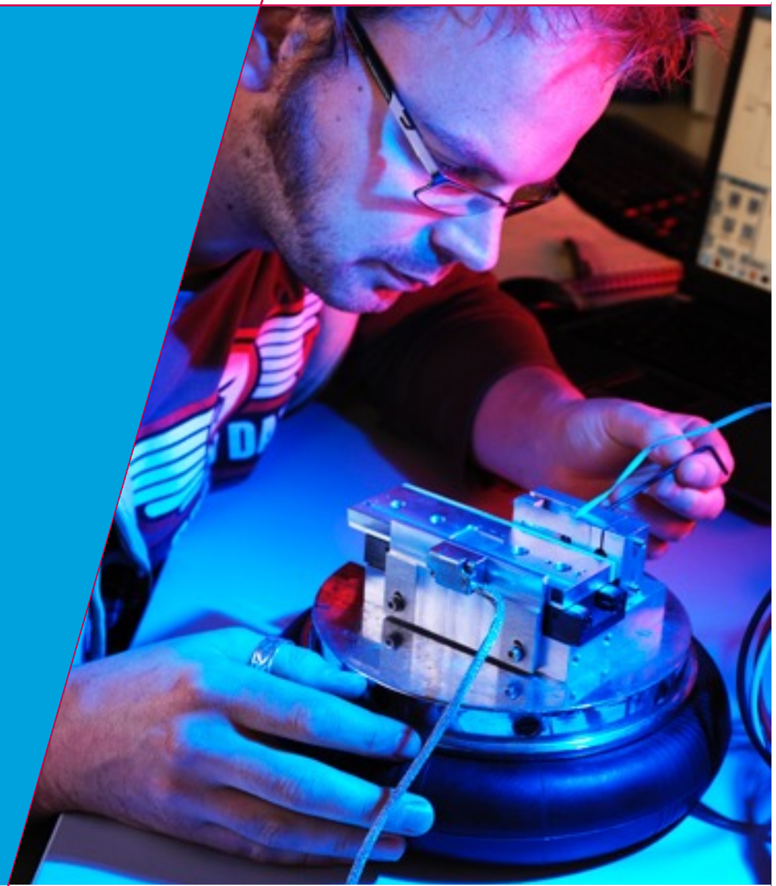
2IC30: Essential computer architecture. Number systems.



Jan Friso Groote

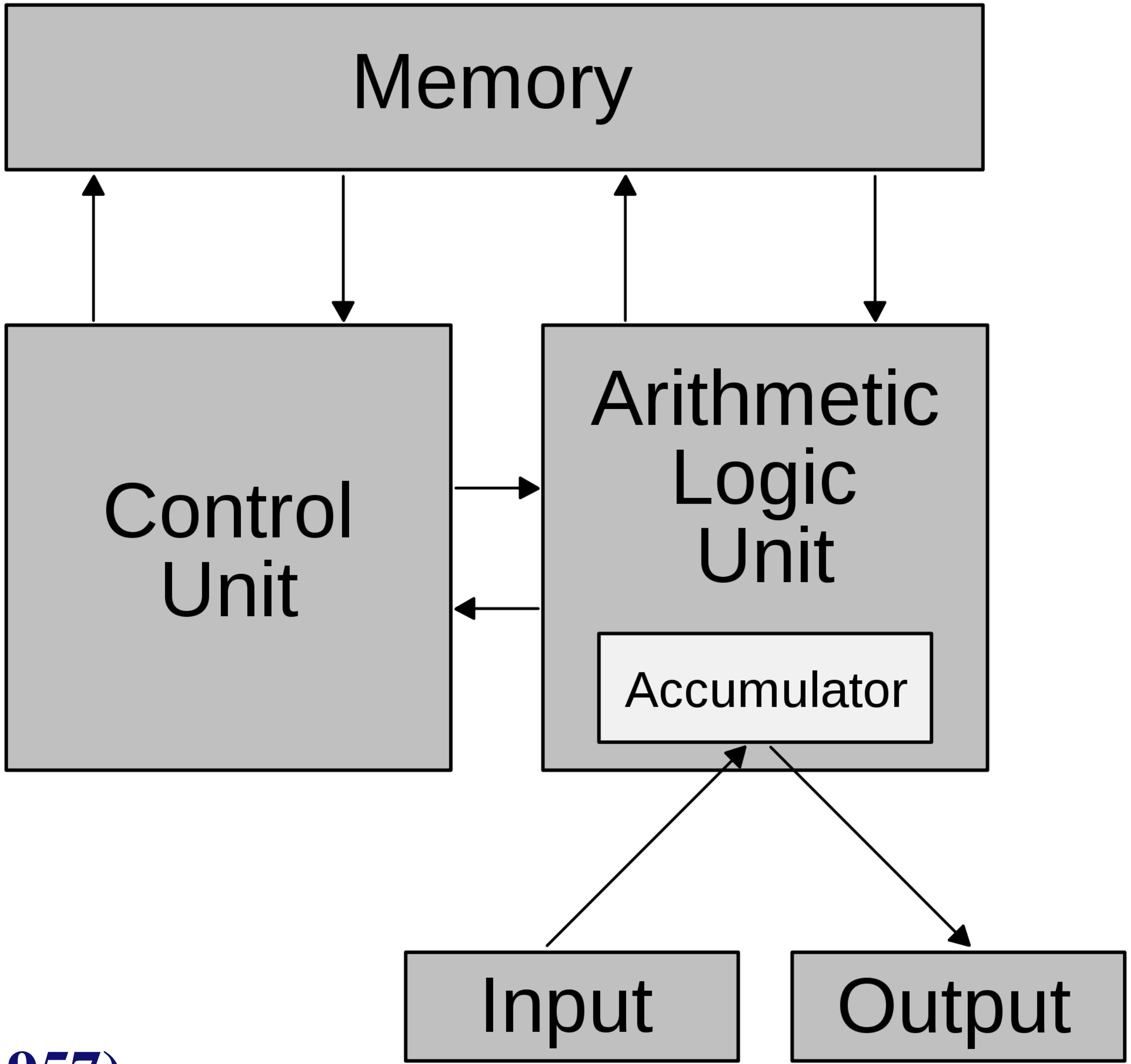
Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

How does a computer work?

Von Neumann architecture.

Program is stored in memory.



(1903-1957)

How does a computer work?

Components

Arithmetical logical unit (ALU).

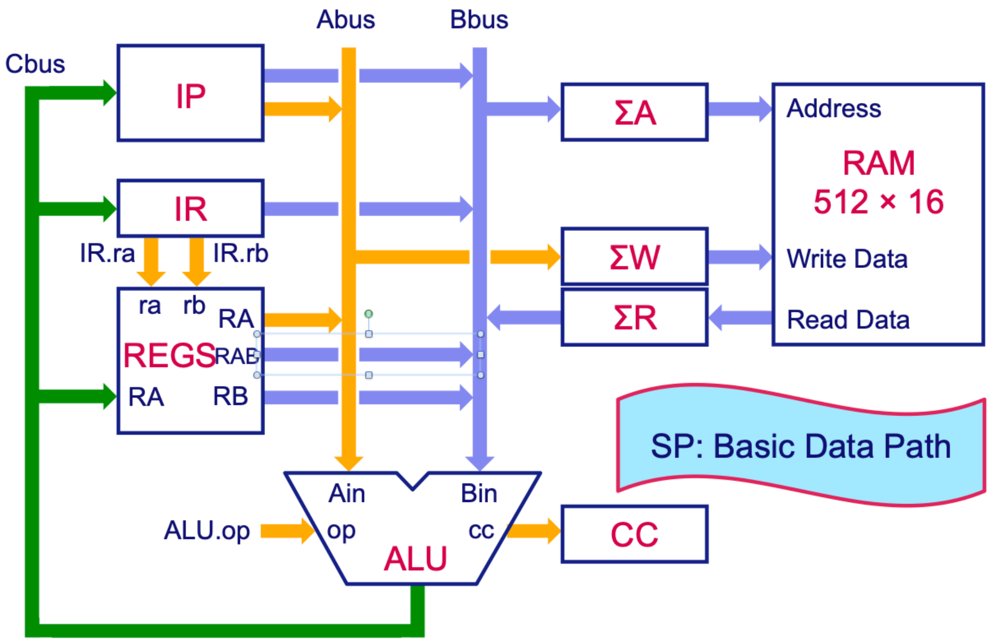
Registers (IP, REGS).

Multiplexers.

Memory address register (ΣA)

Memory Registers (ΣW, ΣR)

Instruction Register (IR)



ALU processes ‘numbers’.

Technische Universiteit Eindhoven University of Technology

How do we represent and manipulate numbers?

- Number systems:
  - Represention of natural numbers
  - Represention of negative numbers
  - Addition, subtraction, multiplication, division, …
  - Overflow conditions
- Implementation:
  - Arithmetic circuits
  - Arithmetic Logical Units

TU/e Technische Universiteit Eindhoven University of Technology

Positional number systems

1) base 10 (decimal; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

$$\langle 154\rangle_{10} = 1 \text{ x } 100 + 5 \text{ x } 10 + 4 \text{ x } 1$$
$$= 1 \text{ x } 10^2 + 5 \text{ x } 10^1 + 4 \text{ x } 10^0$$

2) base 2 (binary; 0, 1)

$$\langle 10011010\rangle_2 = 1 \text{ x } 2^7 + 0 \text{ x } 2^6 + 0 \text{ x } 2^5 + 1 \text{ x } 2^4 + 1 \text{ x } 2^3 + 0 \text{ x } 2^2 + 1 \text{ x } 2^1 + 0 \text{ x } 2^0$$
$$= 128 + 16 + 8 + 2 = 154$$

3) base 8 (octal; 0, 1, 2, 3, 4, 5, 6, 7)

$$\langle 232\rangle_8 = 2 \text{ x } 8^2 + 3 \text{ x } 8^1 + 2 \text{ x } 8^0$$
$$= 2 \text{ x } 64 + 3 \text{ x } 8 + 2 \text{ x } 1 = 128 + 24 + 2 = 154$$

4) base 16 (hexadecimal; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

$$\langle 9A\rangle_{16} = 9 \text{ x } 16^1 + 10 \text{ x } 16^0$$
$$= 144 + 10 = 154$$

Technische Universiteit Eindhoven University of Technology

General formula

The value of a number $a_{n\text{-}1}\ a_{n\text{-}2}\ \ldots\ a_0$ in base b is:

$$\langle a_{n\text{-}1}a_{n\text{-}2}\ldots a_0\rangle_b =$$
$$a_{n\text{-}1}b^{n\text{-}1} + a_{n\text{-}2}b^{n\text{-}2} + a_{n\text{-}3}b^{n\text{-}3} + a_{n\text{-}4}b^{n\text{-}4}$$
$$+ \ldots + a_3b^3 + a_2b^2 + a_1b^1 + a_0b^0$$

$$\langle a_{n\text{-}1}a_{n\text{-}2}\ldots a_0\rangle_b = \sum_{i=0}^{n\text{-}1} a_i b^i$$

Technische Universiteit
Eindhoven
University of Technology

Numbers in a table.

b_3	b_2	b_1	b_0	decimal	hexadecimal	octal
0	0	0	0	0	0	00
0	0	0	1	1	1	01
0	0	1	0	2	2	02
0	0	1	1	3	3	03
0	1	0	0	4	4	04
0	1	0	1	5	5	05
0	1	1	0	6	6	06
0	1	1	1	7	7	07
1	0	0	0	8	8	10
1	0	0	1	9	9	11
1	0	1	0	10	A	12
1	0	1	1	11	B	13
1	1	0	0	12	C	14
1	1	0	1	13	D	15
1	1	1	0	14	E	16
1	1	1	1	15	F	17

Octal machine.

Binary → octal → hexadecimal

recall: $\langle 154\rangle_{10} = \langle 10011010\rangle_2 = \langle 232\rangle_8 = \langle 9A\rangle_{16}$

grouping bits

$$\underbrace{10}_{2}\underbrace{011}_{3}\underbrace{010}_{2}\;{}_8 \qquad \underbrace{1001}_{9}\underbrace{1010}_{A}\;{}_{16}$$

why does this work?

Consider a n=$4k$ bits binary number → k digit hexadecimal number

$$a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + a_{n-3}2^{n-3} + a_{n-4}\boxed{2^{n-4}} = 2^{4(k-1)}$$

$$+ \ldots + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$$

$$= (a_{n-1}2^3 + a_{n-2}2^2 + a_{n-3}2^1 + a_{n-4}2^0)\boxed{2^{4(k-1)}}$$

$$+ \ldots + (a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0)\boxed{2^{4(0)}}$$

$$= b_{k-1}16^{k-1} + \ldots + b_0 16^0$$ with b_i as digits

also works the other way round!

TU/e Technische Universiteit Eindhoven University of Technology

Octal → hexadecimal

$$\langle 232 \rangle_8$$

$$\langle 10\ 011\ 010 \rangle_2$$

$$\langle 9\mathrm{A} \rangle_{16}$$

Base 10 → base X: division with remainder

$\langle 154 \rangle_{10}$

$154 / 3 =$	51	remains 1
$51 / 3 =$	17	remains 0
$17 / 3 =$	5	remains 2
$5 / 3 =$	1	remains 2
$1 / 3 =$	0	remains 1

$\langle 1\,2\,2\,0\,1 \rangle_3$

$= 1 + 18 + 54 + 81 = 154$

why does this work?

let $N = \langle a_{n-1}a_{n-2}\ldots a_1 a_0 \rangle_b$

$= a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \ldots a_1 b^1 + a_0 b^0$

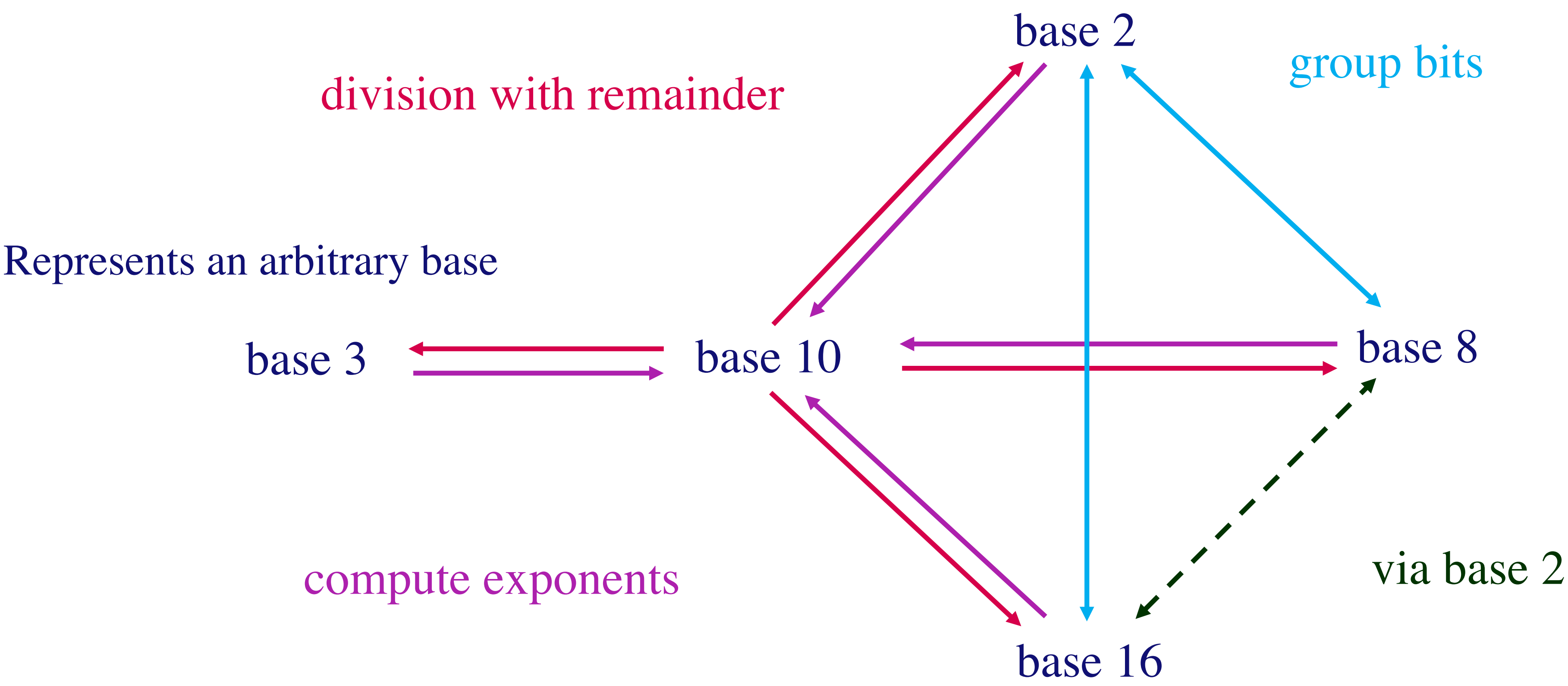
$Q_0 = N/b = \quad a_{n-1}b^{n-2} + a_{n-2}b^{n-3} + \ldots + a_1 b^0 \quad$ remains $\boxed{a_0}$

$Q_1 = Q_0/b = \quad a_{n-1}b^{n-3} + a_{n-2}b^{n-4} + \ldots + a_2 b^0 \quad$ remains $\boxed{a_1}$

$\vdots$

$Q_{n-1} = Q_{n-2}/b = \quad 0 \quad$ remains $\boxed{a_{n-1}}$

Conversions overview



Horner's rule to evaluate polynomials

$$a \times y^3 + b \times y^2 + c \times y^1 + d \times y^0$$

$=$ { isolate d and factor out y }

$$(a \times y^2 + b \times y^1 + c \times y^0) \times y + d$$

$=$ { isolate c and factor out y }

$$((a \times y^1 + b \times y^0) \times y + c) \times y + d$$

$=$ { isolate b and factor out y }

$$((a \times y + b) \times y + c) \times y + d$$

Hence: only 3 multiplications needed instead of 6.

Generally: evaluation of a polynomial with n+1 terms requires n multiplications, instead of ½n(n+1) if exponentiation is calculated by iterative multiplication.

Technische Universiteit Eindhoven University of Technology

Horner's rule: conversion of binary to decimal

$\langle 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \rangle_2$

$=$ { definition of binary value }

$1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$=$ { Horner's rule }

$(((((((1 \times 2 + 0) \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1) \times 2 + 0$

$=$ { arithmetic: inside out }

$((((((2 \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1) \times 2 + 0$

$=$ { arithmetic }

$(((((4 \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1) \times 2 + 0$

$=$ { and so on ... }

$77 \times 2 + 0$

$=$ { arithmetic }

154 .

TU/e Technische Universiteit Eindhoven University of Technology

Questions?

Technische Universiteit
Eindhoven
University of Technology

Addition and subtraction

addition base 3

$$
\begin{array}{rrrrrl}
 & 1 & 1 & & 1 & \text{carry (remember)} \\
1 & 2 & 2 & 0 & 1 & \\
 & & 2 & 1 & 2 & + \\
\hline
2 & 0 & 1 & 2 & 0 &
\end{array}
$$

$(154 + 23 = 177)$

$(= 6 + 9 + 162 = 177)$

subtraction base 3

$$
\begin{array}{rrrrrl}
 & & 3 & 3 & 3 & \text{borrow} \\
1 & 2 & 2 & 0 & 1 & \\
 & & -2 & -1 & -2 & \\
 & -1 & -1 & -1 & & + \\
\hline
1 & 1 & 2 & 1 & 2 &
\end{array}
$$

$(154 - 23 = 131)$

$(= 2 + 3 + 18 + 27 + 81 = 131)$

Technische Universiteit Eindhoven University of Technology

Arithmetic operations: Addition

Decimal:

$$\begin{array}{rccc} & 1 & 1 & \\ & & 9 & 7 \\ + & & 1 & 6 \\ \hline & 1 & 1 & 3 \end{array}$$

carries

Binary:

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

Arithmetic operations: Subtraction

Decimal:

$$
\begin{array}{rccc}
 & -1 & -1 & \\
 & 1 & 1 & 3 \\
- & & 1 & 6 \\
\hline
 & 0 & 9 & 7
\end{array}
$$

borrows

Binary:

$$
\begin{array}{rcccc}
 & -1 & -1 & -1 & \\
 & 1 & 0 & 1 & 0 \\
- & & 0 & 1 & 1 \\
\hline
 & 0 & 1 & 1 & 1
\end{array}
$$

Binary addition of 2 bits: Half Adder

$(c, s) \;=\;$ "binary representation of $a + b$"

half adder

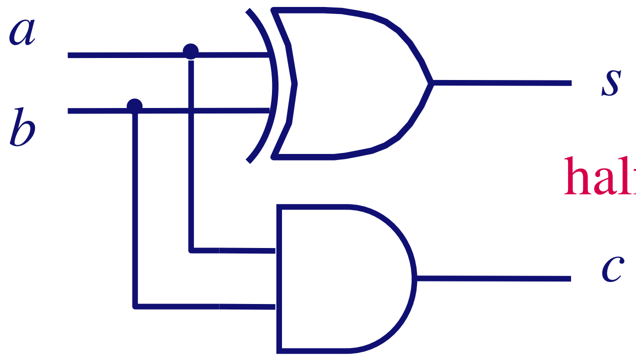
a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$b \backslash a$	0	1
0	0	$\boxed{1}$
1	$\boxed{1}$	0

$b \backslash a$	0	1
0	0	0
1	0	$\boxed{1}$

$$s \;=\; (\neg a \wedge b) \vee (a \wedge \neg b) \qquad\qquad c \;=\; a \wedge b$$

$$= \; a \oplus b \qquad \text{(exclusive OR)}$$



half adder

Binary addition of 3 bits: Full Adder

$$(cout, s) \quad = \quad \text{“binary representation of } a + b + cin \text{”}$$

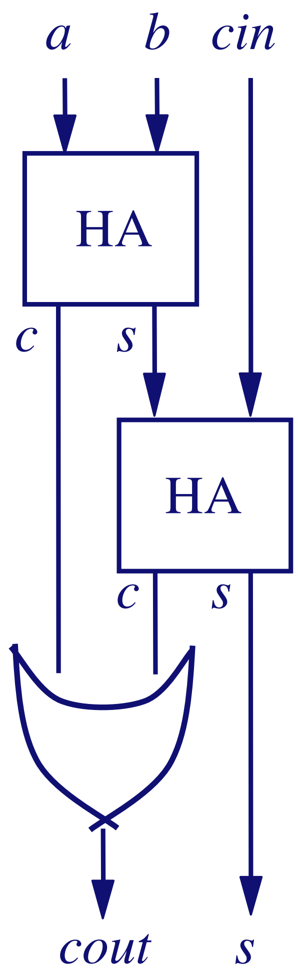
full adder

a	b	cin	$cout$	s
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

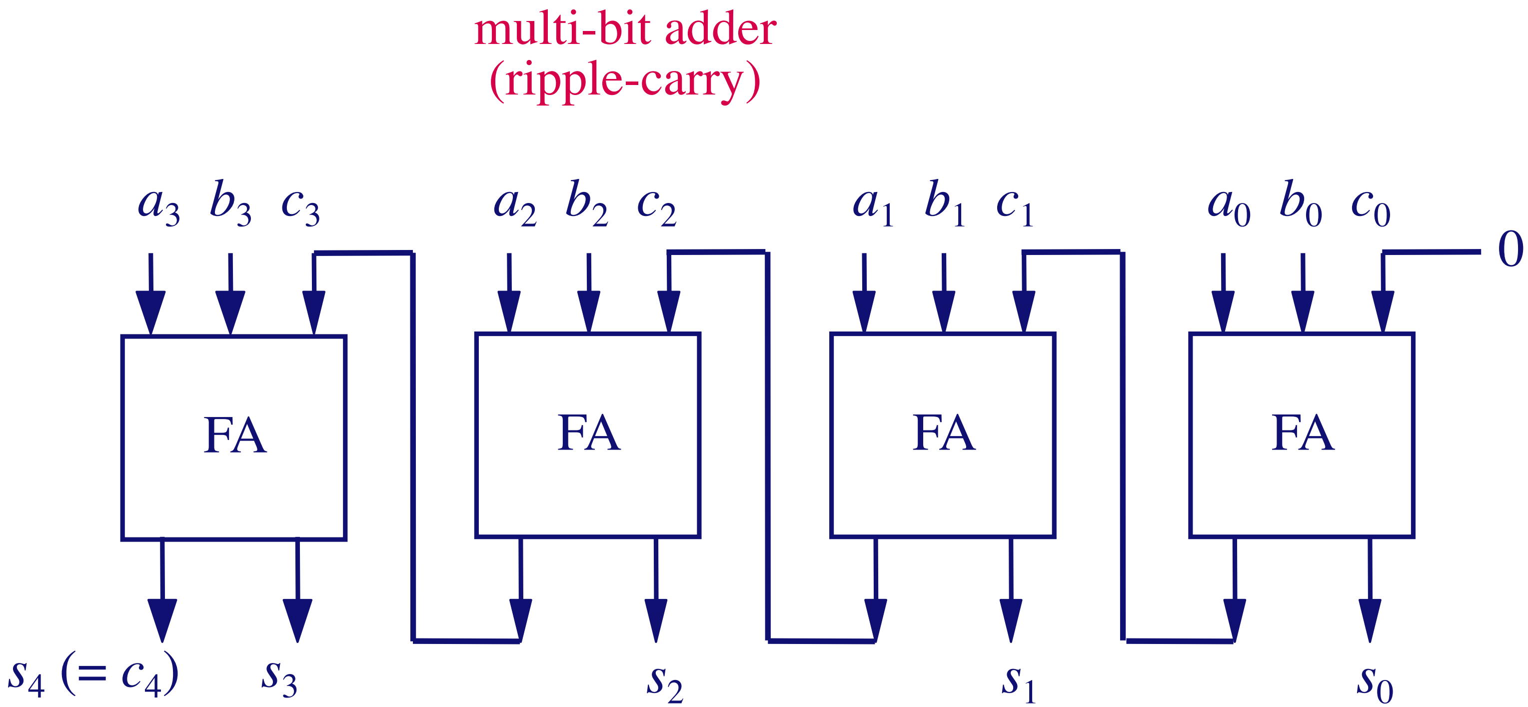
$$s \;=\; a \oplus b \oplus cin$$

$$\begin{aligned} cout &= (a \wedge cin) \vee (b \wedge cin) \vee (a \wedge b) \\ &= ((a \oplus b) \wedge cin) \vee (a \wedge b) \end{aligned}$$

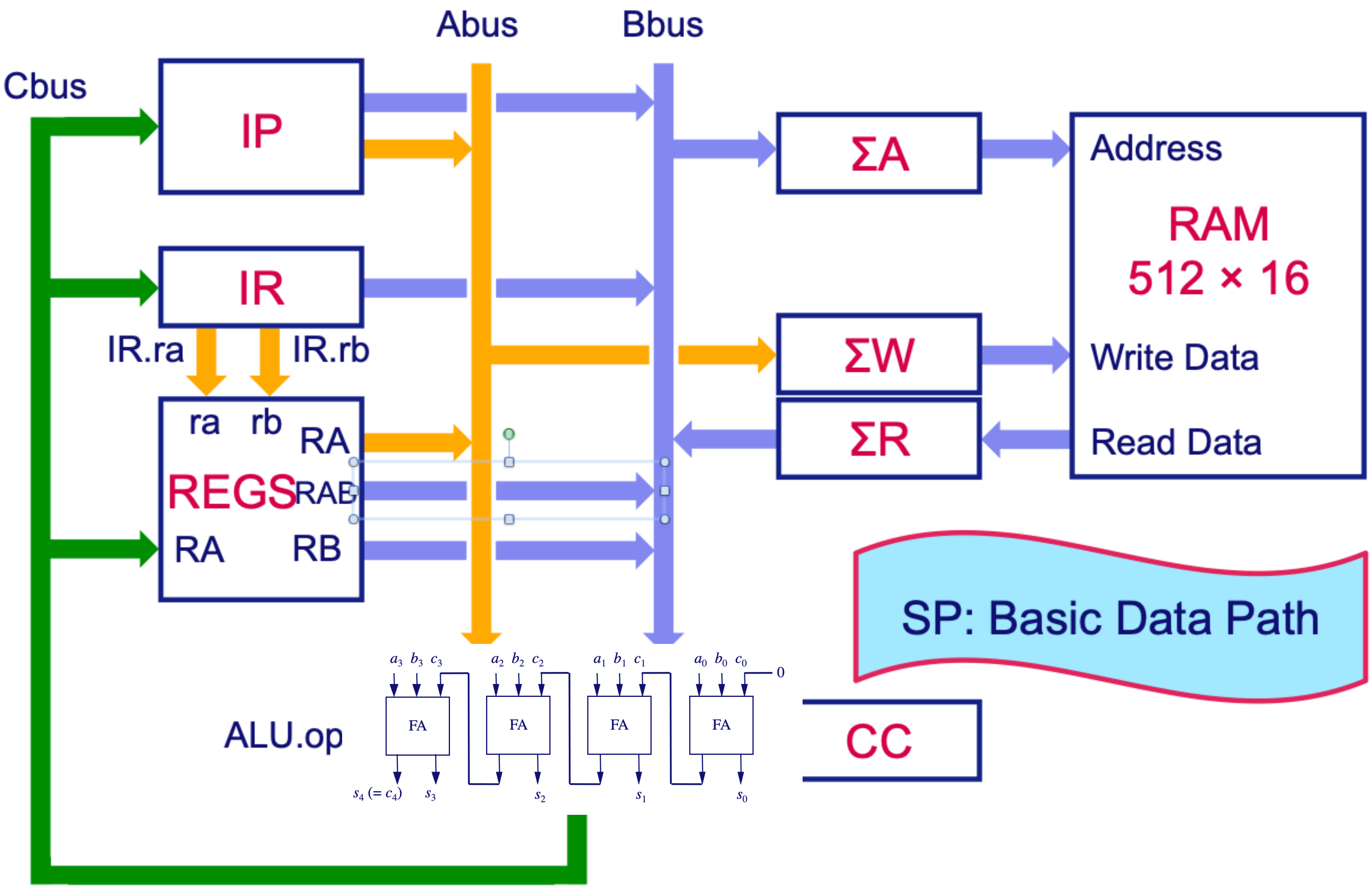
full adder



Binary addition of n-bit numbers



The adder is part of the ALU

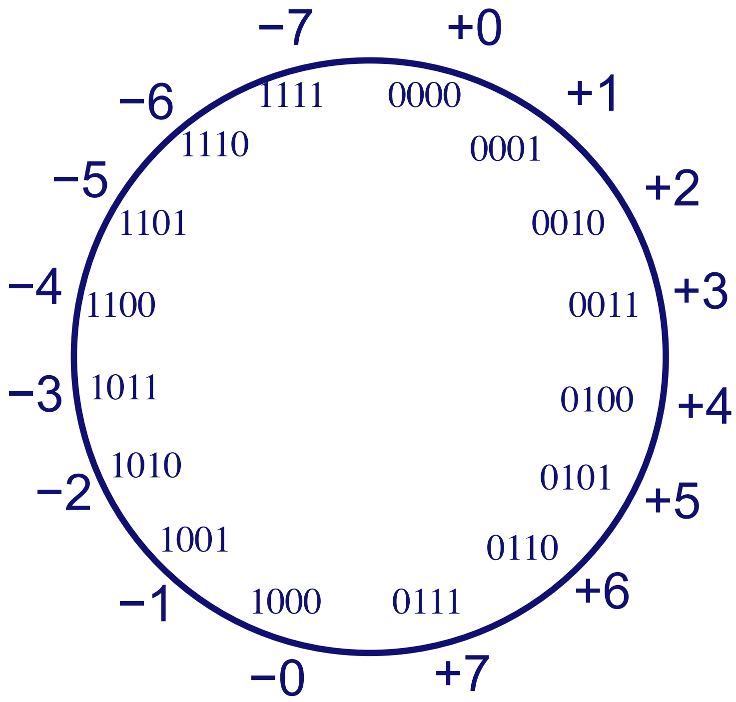


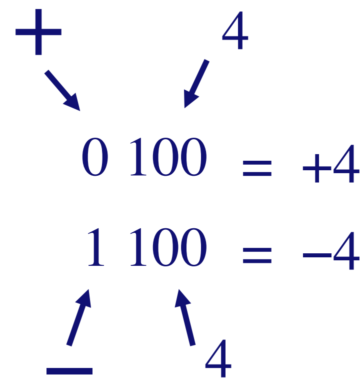
Questions?

Representing negative numbers

- Representation of natural numbers is usually the same for all systems.
- n+1-bits numbers: 2^{n+1} possible combinations, range: $0,\ldots,2^{n+1}-1$.
- Three different ways to represent negative numbers:
  - sign-and-magnitude;
  - 1s-complement;
  - 2s-complement.
- Sign-and-magnitude: used in *floating-point* numbers –*reals*–.
- 1s-complement is inconvenient for integer arithmetic.
- 2s-complement most convenient for integer arithmetic.

Sign-and-Magnitude Representation





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

Three lower order bits are value: 0 (000) t/m 7 (111).

Range for n+1 bits $-(2^n-1) \ldots 2^n-1$, symmetric around 0.

2 representations for 0: +0 and –0.

Complicated addition and subtraction.

Sign-and-Magnitude: Addition and Subtraction

General remark

Subtraction is possible with negation and addition:

Adding number with equal sign bits:

directly add the value

result sign equal to
value signs

$$\begin{array}{r} 1 \\ 3 \\ \hline 4 \end{array}+ \;\Rightarrow\; \begin{array}{r} 0001 \\ 0011 \\ \hline 0100 \end{array} \qquad \begin{array}{r} -1 \\ -3 \\ \hline -4 \end{array}+ \;\Rightarrow\; \begin{array}{r} 1001 \\ 1011 \\ \hline 1100 \end{array}$$

Adding numbers with different sign bits:

how to implement
this?

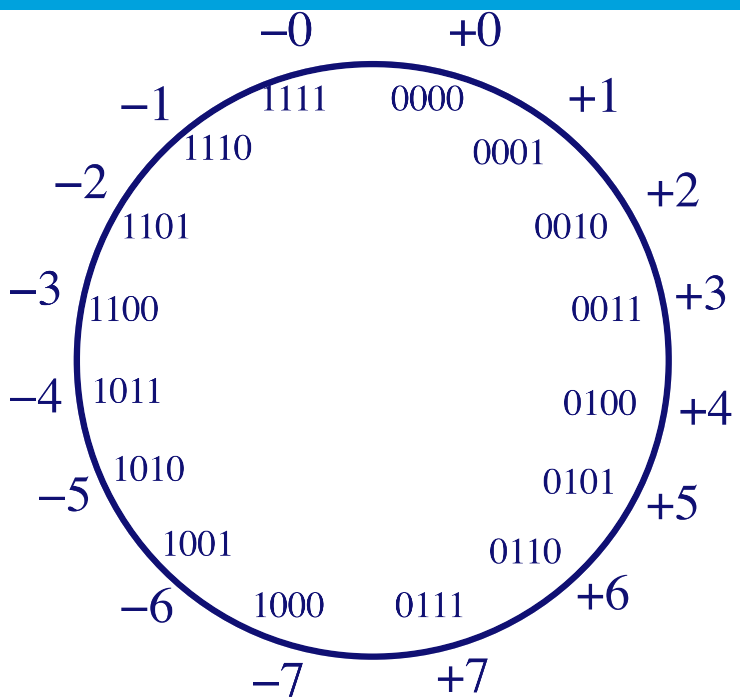
$$\begin{array}{r} 4 \\ -3 \\ \hline 1 \end{array}+ \;\Rightarrow\; \begin{array}{r} 0100 \\ 1011 \\ \hline 0001 \end{array} \qquad \begin{array}{r} -4 \\ 3 \\ \hline -1 \end{array}+ \;\Rightarrow\; \begin{array}{r} 1100 \\ 0011 \\ \hline 1001 \end{array}$$

subtract smallest from the biggest
and use the sign of the latter

requires subtraction AND comparison

Technische Universiteit
Eindhoven
University of Technology

1s-complement



+ 4

$0\ 100_2 = +4$

$1\ 011_2 = -7 + 3 \quad = -4$

–

Highest order bit is sign bit: 0 = positive, 1 = negative.

Range for n+1 bits $-2^n+1,\ldots,2^n-1$, symmetric around 0.

Positive numbers: lower order bits are ''value''.

Negative numbers: lower order bits are: -2^n+1+ ''value'' (with n+1 bits).

A *double* representation for 0.

General formula

The value of a number $a_{n-1}\ a_{n-2}\ \ldots\ a_0$ in 1-complement is:

If a_{n-1}=0 then:

$$a_{n-2}2^{n-2} + a_{n-3}2^{n-3} + \ldots + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$$

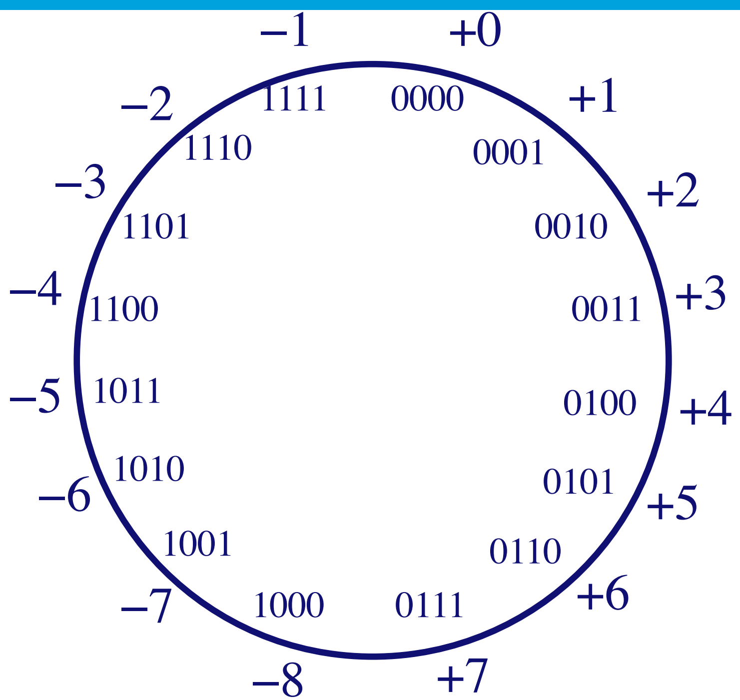
If a_{n-1}=1 then:

$$-\ (1-a_{n-2})2^{n-2} + (1-a_{n-3})2^{n-3} + \ldots + (1-a_3)2^3 + (1-a_2)2^2 + (1-a_1)2^1 + (1-a_0)2^0$$

$$\begin{cases} \displaystyle\sum_{i=0}^{n-2} a_i 2^i & \text{if } a_{n-1}=0, \\ \displaystyle -\sum_{i=0}^{n-2} (1-a_i)2^i & \text{if } a_{n-1}=1. \end{cases}$$

TU/e Technische Universiteit Eindhoven University of Technology

2s-complement (1)



$$\overset{+}{0}\;\overset{4}{100}_2 = +4$$

$$\underset{-}{1}\;\underset{3}{011}_2 = -8 + 3 = -5$$

Highest order bit is sign bit: 0 = positive (or 0), 1 = negative.

Range for n+1 bits $-2^n,\ldots, 2^n-1$, *not* symmetric around 0.

Positive numbers: lower order bits are ''value''.

Negative numbers: lower order bits are: -2^n + ''value'' (with n+1 bits).

A *single* representation for 0 and no complications with addition.

General formula

The value of a number $a_{n-1}\ a_{n-2}\ \ldots\ a_0$ in 2-complement is:

$$[a_{n-1}a_{n-2}\ldots a_0] = -a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + a_{n-3}b^{n-3} + a_{n-4}b^{n-4} + \ldots + a_3b^3 + a_2b^2 + a_1b^1 + a_0b^0$$

$$[a_{n-1}a_{n-2}\ldots a_0] = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_ib^i$$

Technische Universiteit
Eindhoven
University of Technology

2-s complement (3)

Let m be a 2s complement number in n bits.

What is the 2s-complement n-bit representation of $-m$?

$$-m = -(-a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + \ldots + a_1 \times 2^1 + a_0 \times 2^0)$$

$$= a_{n-1} \times 2^{n-1} - a_{n-2} \times 2^{n-2} - \ldots - a_1 \times 2^1 - a_0 \times 2^0$$

$$= a_{n-1} \times 2^{n-1} - a_{n-2} \times 2^{n-2} - \ldots - a_1 \times 2^1 - a_0 \times 2^0 \underbrace{+ \sum_{i=0}^{n-2} 2^i + 1 - 2^{n-1}}_{0}$$

$$= -(1-a_{n-1}) \times 2^{n-1} + (1-a_{n-2}) \times 2^{n-2} + \ldots + (1-a_1) \times 2^1 + (1-a_0) \times 2^0 + 1$$

This number is representable as an 2s-complement number, except if m is the smallest negative number.

In 4 bits: If m=1000, then $-m$ has value 0111 + 1, which is 8, which cannot be represented.
If m=1111, then $-m$ has value 0000 + 1, which is 1, represented by 0001.

2-s complement (3)

Let m be a 2s-complement number in n bits.

What is the 2s-complement n-bits representation of $-m$?

Calculate 2^n-m where m is interpreted as a positive number and the result is interpreted as an n digit number.

$$2^n - m = 2^n - \sum_{i=0}^{n-1} a_i \times 2^i$$

$$= 2^n - \sum_{i=0}^{n-1} a_i \times 2^i - 2^n + \sum_{i=0}^{n-1} 2^i + 1$$

$$= \sum_{i=0}^{n-1} (1-a_i) \times 2^i + 1$$

$$-m = -(1-a_{n-1})2^{n-1} + \sum_{i=0}^{n-2} (1-a_i) \times 2^i + 1$$

If 1-a_{n-1}=0, they are the same.

If 1-a_{n-1}=1, we interpret the first digit of the result at the left as negative and they are also equal.

Technische Universiteit Eindhoven University of Technology

2-s complement (3)

Let m be a positive number.

What is the 2s-complement n+1-bits representation of $-m$?

Example: 2s-complement of –7

$$(2^4)_{10} = (1)0000_2$$
$$(7)_{10} = 0111_{2c} \; -$$
$$1001_{2c} = (-7)_{10}$$

also works with negative numbers!

$$(2^4)_{10} = (1)0000_2$$
$$(-7)_{10} = 1001_{2c} \; -$$
$$0111_{2c} = (7)_{10}$$

Fast method:
invert bits + 1

$$\underset{(7)}{0111} \longrightarrow 1000 + 1 \longrightarrow \underset{(-7)}{1001}$$

as does the fast method

$$\underset{(-7)}{1001} \longrightarrow 0110 + 1 \longrightarrow \underset{(7)}{0111}$$

TU/e Technische Universiteit Eindhoven University of Technology

2-s complement: sign extension and truncation

- Question: what is –4 represented in 4 bits? Answer: 1 1 0 0 .
- Question: what is –4 represented in 8 bits?
- Answer: just *sign-extend* the 4-bits answer, by replicating the sign bit:

  1 1 1 1 1 1 0 0

- Question: what is –4 represented in 3 bits?
- Answer: just *truncate* the previous 4-bits answer, by deleting the sign bit:

  1 1 1 1 1 1 0 0

- If the sign bit is not changed by truncation the result is correct, but

  if the sign bit is changed the result is not representable: overflow!

2s-complement: Addition and Subtraction

As before,
Subtraction by addition and negation.

adding directly

$$\begin{array}{r} 1 \\ 3 \\ \hline 4 \end{array} + \;\Rightarrow\; \begin{array}{r} 0001 \\ 0011 \\ \hline 0100 \end{array} + \qquad \begin{array}{r} -1 \\ -3 \\ \hline -4 \end{array} + \;\Rightarrow\; \begin{array}{r} 1111 \\ 1101 \\ \hline 1\,1100 \end{array} +$$

ignore carry–out

$4 - 3 =$
$4 + (-3)$:
$-0011=1101$

$$\begin{array}{r} 4 \\ -3 \\ \hline 1 \end{array} + \;\Rightarrow\; \begin{array}{r} 0100 \\ 1101 \\ \hline 1\,0001 \end{array} +$$

$-4 - -3 =$
$-4 + (--3)$

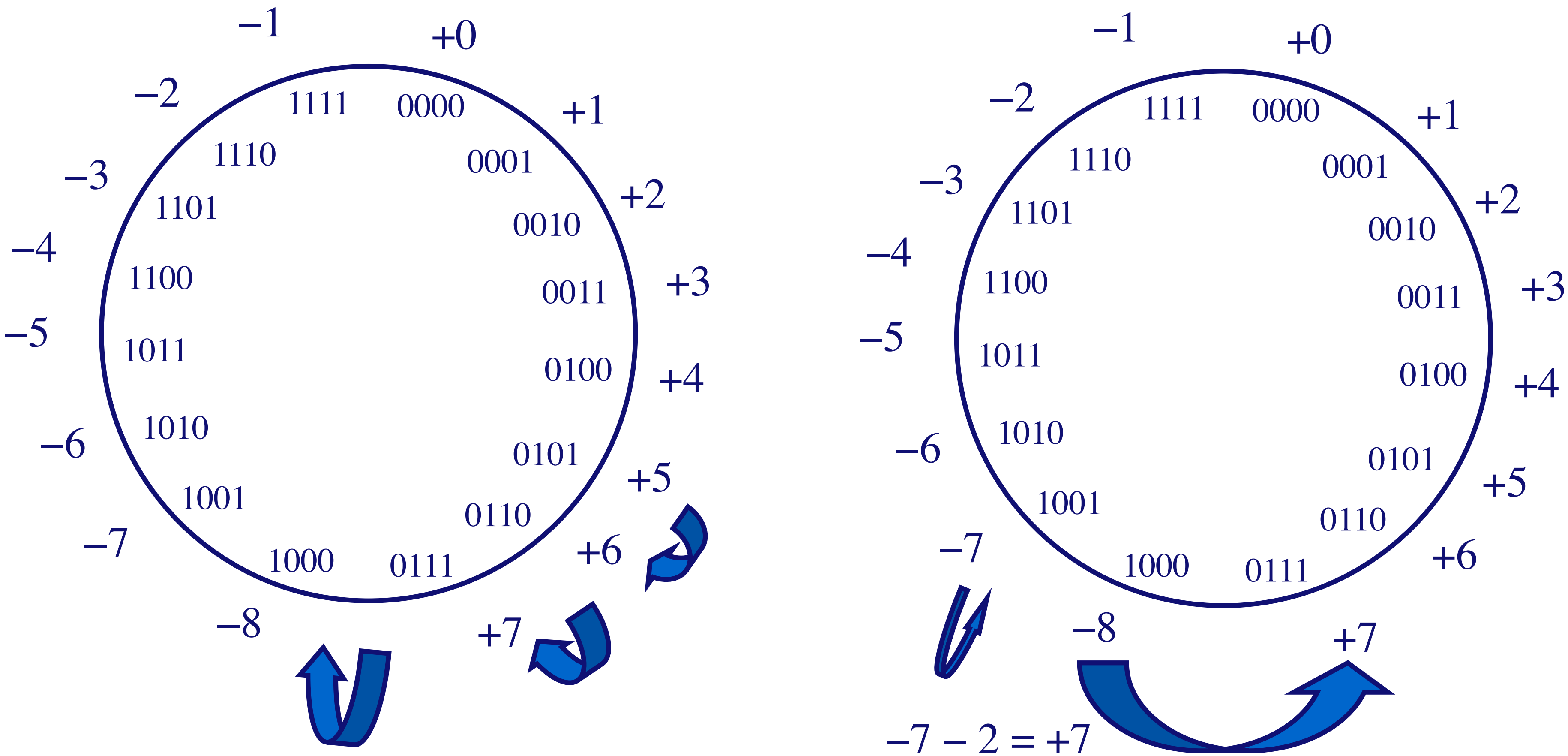
$$\begin{array}{r} -4 \\ 3 \\ \hline -1 \end{array} + \;\Rightarrow\; \begin{array}{r} 1100 \\ 0011 \\ \hline 1111 \end{array} +$$

Adder and bit inverter suffice for implementing addition AND subtraction.

Because of the simpler addition scheme, 2s-complement is the predominant choice for computations with integer number in digital systems.

Result not Representable: Overflow

Adding two positive numbers yields a negative number
or
adding two negative numbers yields a positive number.



−7 − 2 = +7

5 + 3 = −8

with addition, overflows only occur in numbers with equal signs!

Technische Universiteit Eindhoven University of Technology

Detecting overflow (1)

carry-in ≠ carry-out

overflow

	0 1 1 1	(carries)
+5	0 1 0 1	
+3	0 0 1 1	
−8	0 1 0 0 0	

overflow

	1 0 0 0
−7	1 0 0 1
−2	1 1 1 0
+7	1 0 1 1 1

no overflow

carry−in = carry−out

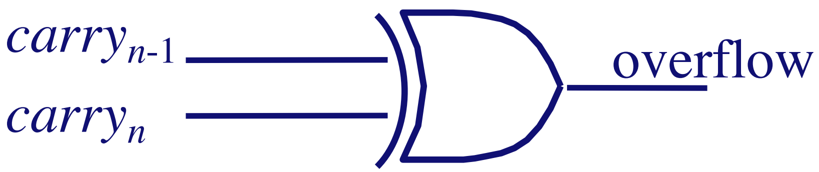
	0 0 0 0
+5	0 1 0 1
+2	0 0 1 0
+7	0 0 1 1 1

	1 1 1 1
−3	1 1 0 1
−5	1 0 1 1
−8	1 1 0 0 0

There is an overflow if and only if most significant carries are different.

Detecting overflow (1)

Overflow occurs when carry-in ≠ carry-out.



Detecting overflow (method for use by hand)

- *Overflow* occurs if the result of the addition/subtraction is *not representable* in the number of bits used.

- If we are working with n-bits 2s complement numbers:

  Sign-extend the two numbers with 1 additional bit, and calculate the result in n+1 bits. The result is representable in n+1 bits, say:

  $a_n \; a_{n-1} \; . \; . \; . \; a_2 \; a_1 \; a_0$ represents the result in n+1 bits; now:

  if $a_n = a_{n\text{-}1}$ the result is *also* representable in n bits: no overflow!

  if $a_{n+1} \neq a_{n\text{-}1}$ the result is *not* representable in n bits: overflow!

- Use?
- Calculations by hand: simple, but *don't forget* the sign-extension!
- Circuit implementations, if the last-but-one carry is not available.

Detecting overflow (method for use by hand)

overflow

	sign extend		sign extend
+5	0 0 1 0 1	−7	1 1 0 0 1
+3	0 0 0 1 1	−2	1 1 1 1 0
−8	0 1 0 0 0	+7	1 0 1 1 1

bits different? overflow!

no overflow

	sign extend		sign extend
+5	0 0 1 0 1	−3	1 1 1 0 1
+2	0 0 0 1 0	−5	1 1 0 1 1
+7	0 0 1 1 1	−8	1 1 0 0 0

bits equal? no overflow!

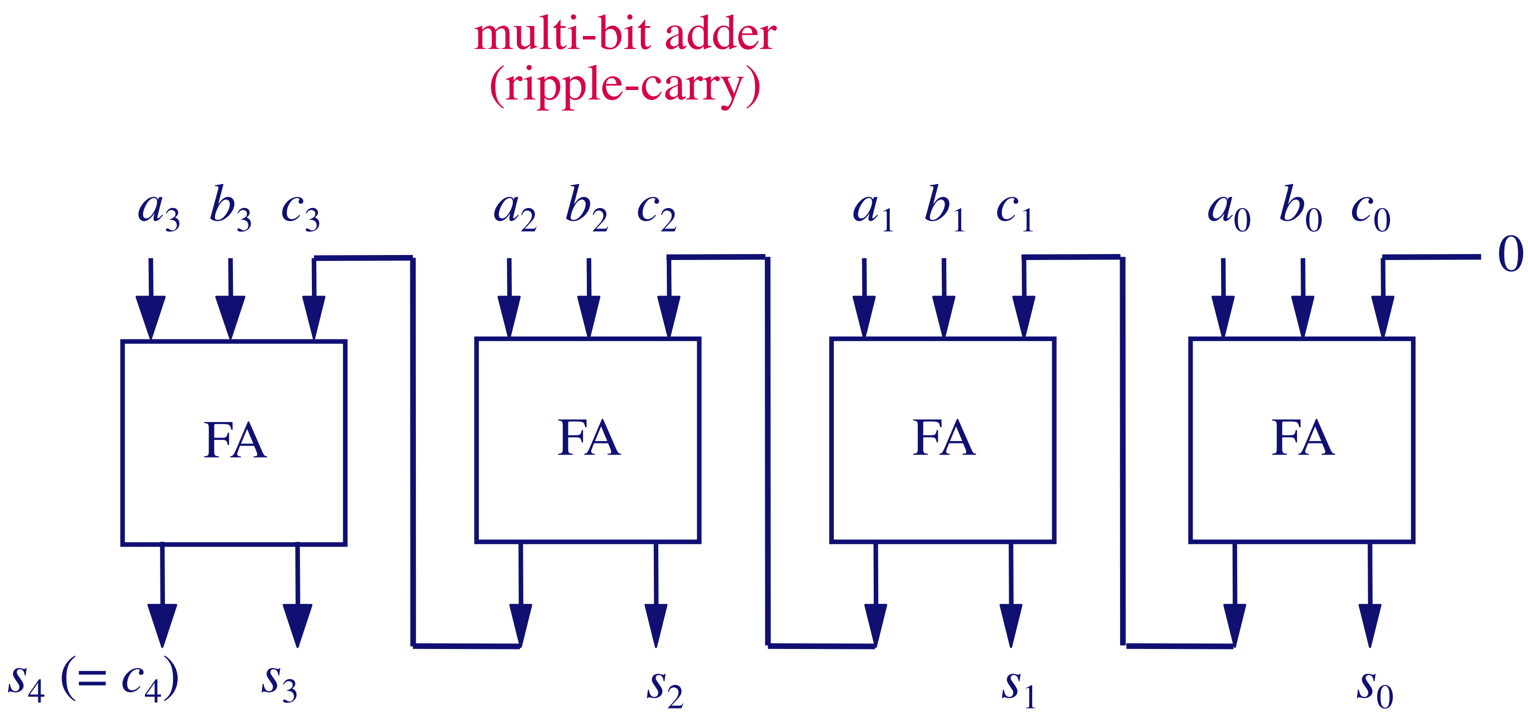
overflow if and only if the left-most 2 bits are *different*

Questions?

Nice theory: now what?

- We will implement all of this with logical gates
- but only for 2s-complement numbers
- and only for addition and subtraction

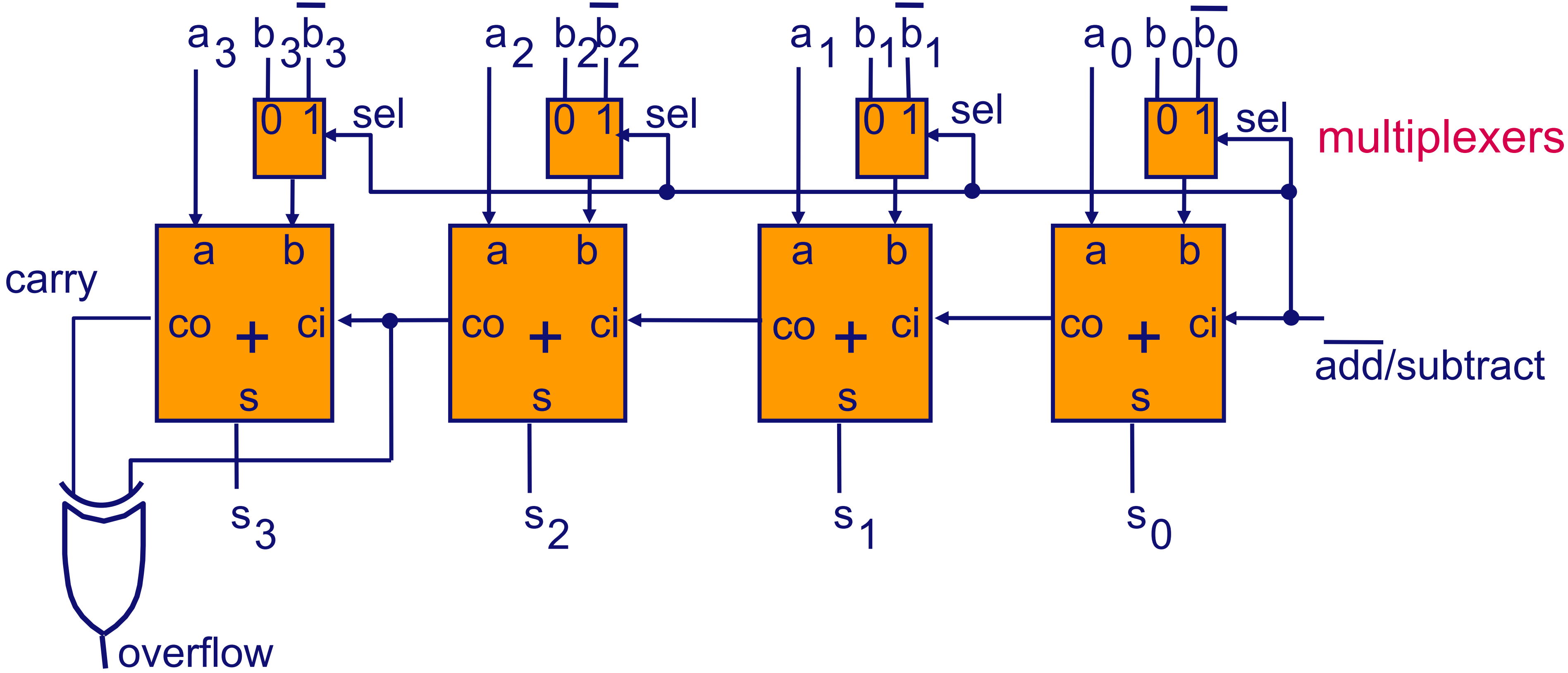
Binary addition of n-bit numbers



Addition and Subtraction in one circuit

$$(a + b)_{10} = (a + b)_{2c}$$

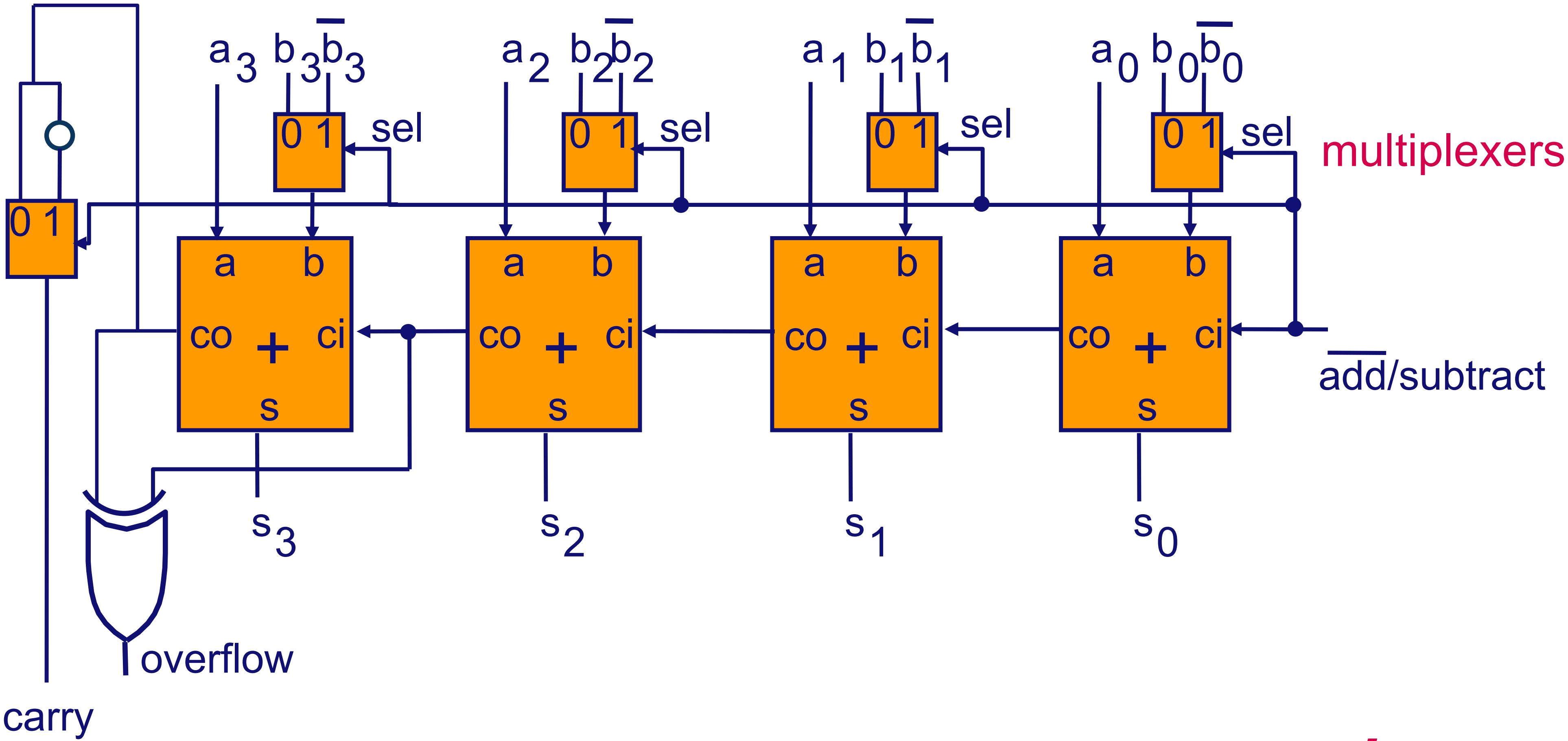
$$(a - b)_{10} = (a + (-b))_{10} = (a)_{2c} + \overline{b}_{2c} + 1$$



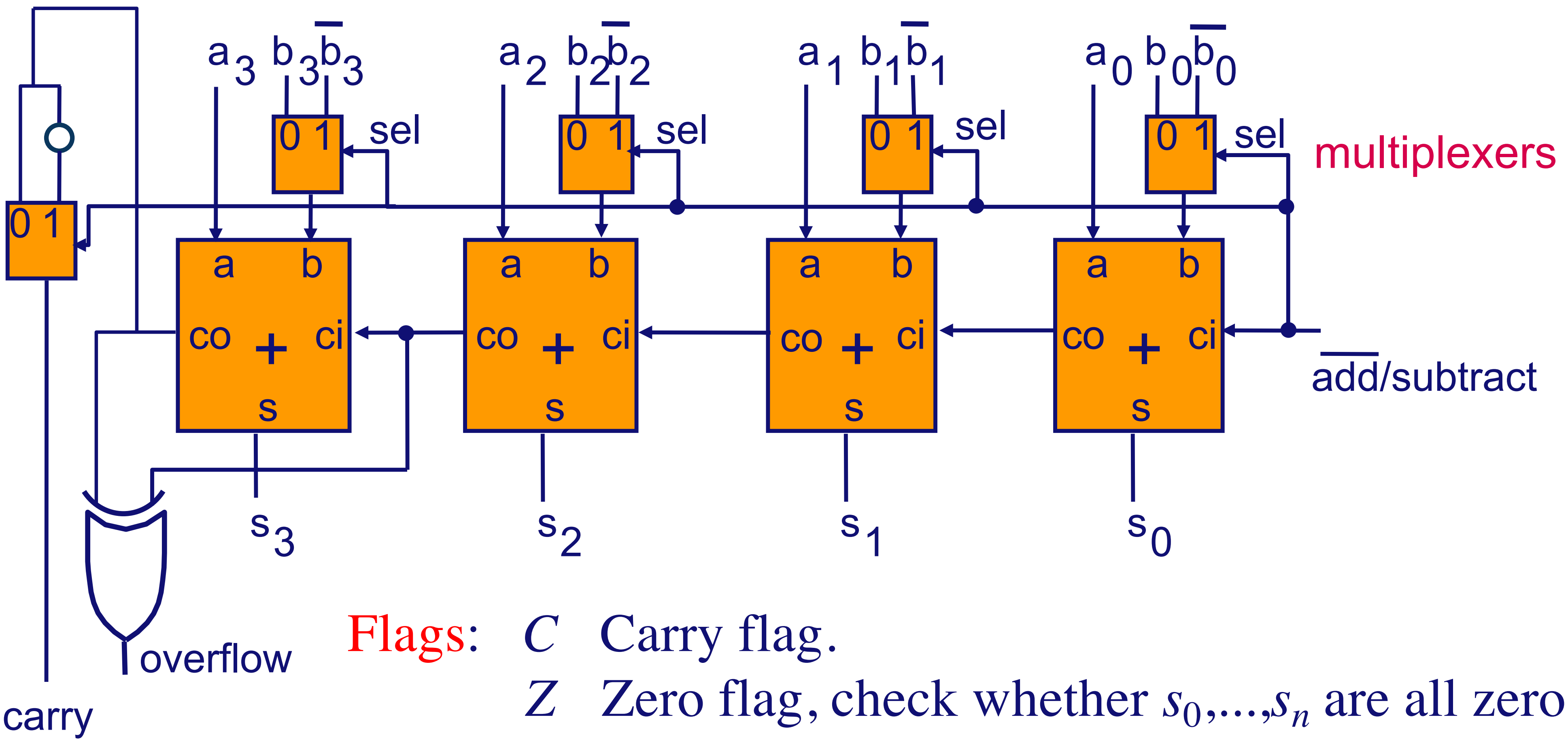
Addition and Subtraction in one circuit

$$(a + b)_{10} = (a + b)_{2c}$$

$$(a - b)_{10} = (a + (-b))_{10} = (a)_{2c} + \overline{b}_{2c} + 1$$



Addition and Subtraction in one circuit



Flags:

- C Carry flag.
- Z Zero flag, check whether $s_0,\ldots,s_n$ are all zero.
- N Negative flag. Equal to s_n.
- V Overflow flag.

Condition Codes: How to be used? (1)

Unsigned and Signed Comparisons

Bit Patterns	Unsigned Values	Two's complement values
0100 1101	+4 < +13 ??	+4 < -3 ??
1100 0011	+12 > +3 ??	-4 > +3 ??

Conclusion: the result of a comparison *depends* on how the bit patterns are interpreted: *unsigned* or *two's-complement*.

Calculate:

α-β

	unsigned	two's complement
$\alpha \leq \beta$	$C=1 \vee Z=1$	$N \neq V \vee Z=1$
$\alpha < \beta$	$C=1$	$N \neq V$
$\alpha = \beta$	$Z=1$	$Z=1$
$\alpha \neq \beta$	$Z=0$	$Z=0$
$\alpha > \beta$	$C=0 \wedge Z=0$	$N=V \wedge Z=0$
$\alpha \geq \beta$	$C=0$	$N=V$

Questions?

Summary

- We know how unsigned numbers are generally represented in other bases and can translate these to each other. Most interesting bases are: decimal, octal, hexadecimal and binary.
- We can design addition on numbers using a ripple adder.
- We know that negative numbers can be represented in (at least) three different ways: sign and magnitude, ones and two’s complement.
- We can add/subtract with negative numbers represented two’s complement, and know how to build a circuit to do that for two’s complement.

Technische Universiteit Eindhoven University of Technology