L4, L5 (a)

adders

- signed numbers
- BCD addition

Adders

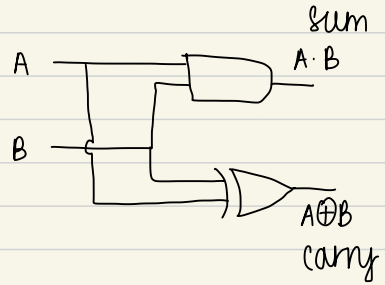
- half adder (2-bit addition)

→ both give a 2 bit result

I/P A	I/P B	msb carry	O/P lsb sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

↑ A·B ↑ A⊕B

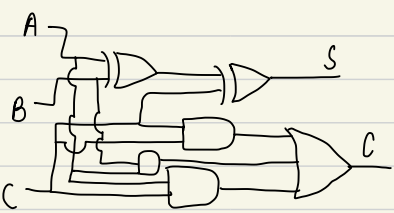
1 AND 1 XOR

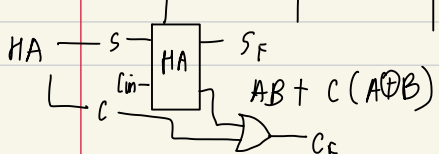


- full adder

A⊕B⊕C
2 XOR

I/P A	I/P B	I/P C	O/P Carry	O/P Sum
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





AB + BC + CA 3 AND

AB + BC(A+A') + CA(B+B') 1 OR

AB + C(AB' + BA')

carry for full adder

$A \cdot B + B \cdot C + C \cdot A$

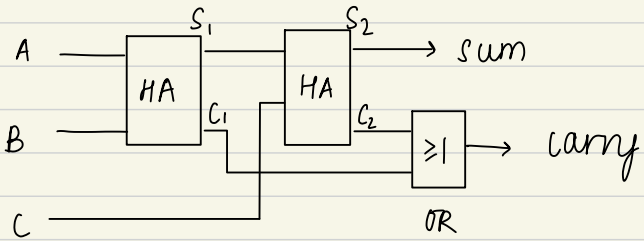
$A \cdot B + C(AB + A'B) + C(AB + B'A)$

$AB + CAB + C(A'B + B'A)$

$AB + CAB + C(A \oplus B)$

$AB + C(A \oplus B)$

↳ implementing full adder using half adder



$S_1 = A \oplus B$ = sum 1/2 adder

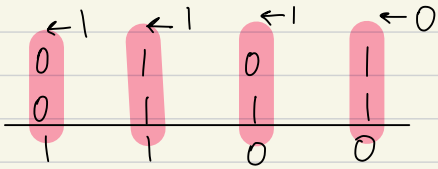
$S_2 = A \oplus B \oplus C$ = sum full adder

$C_1 = AB$

$C_2 = C(A \oplus B)$ carry $= AB + C(A \oplus B)$

- addition of 4 bit no. requires 4 full adders (FA)

$0101_2 + 0111_2 = 1100_2$



we use FA here as it can add each of the 2 bit I/P and also the carry

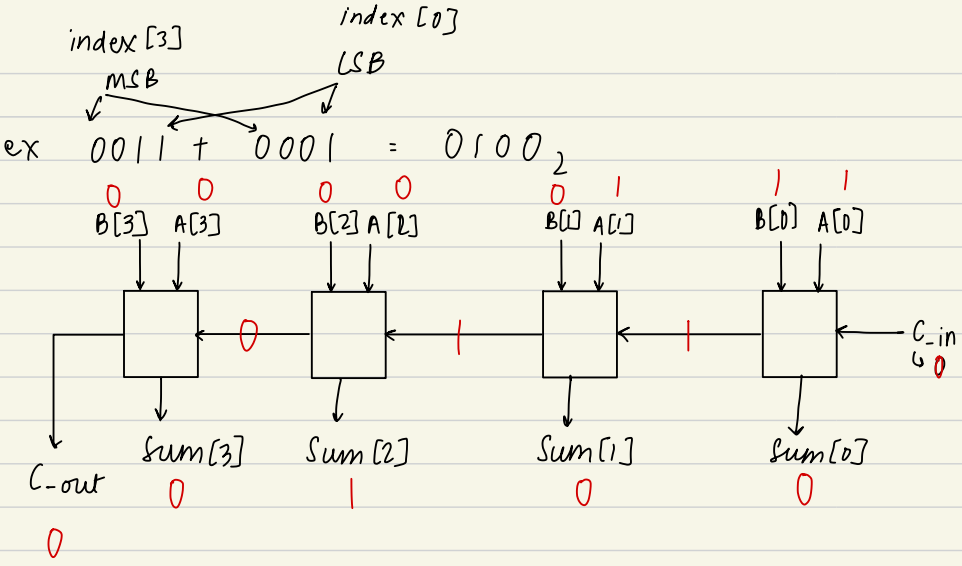
- to add an n-bit number we need n FAs

→ they are linked, as in the carry of the first is an input for the next and so on, hence they are "cascaded" and its called a "parallel adder" or a "ripple adder"

→ augend and addend are just fancy names for numbers to be added.

→ all the bits are fed simultaneously into the adder circuit

→ its very fast and only limited by propagation delay aka carry propagation



- there is a circuit that predicts carry. so that the operation is faster

- for calc propagation delay, we assume each gate to have 1 unit of delay.

signed numbers

- Signed - Magnitude system

→ signed bit = msb

0 → +ve

1 → -ve

$14_{10} = 1110_2$

+14 : 0 1110

-14 : 1 1110

→ equal no. of +ve & -ve values

→ an n-bit no can represent numbers from

$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$ ← disadvantage

→ this doesn't have zero ka representation
actually it has 2 which is fucked up in short

ones complement has the same problem

- 2's complement 2's comp of an n-bit no $= 2^{n+1} -$ magnitude

→ signed bit = msb
0 ⇒ +ve
1 ⇒ −ve

signed bit
↓
and zero is 0000

→ positive rep is same as sign mag.

$14_{10} : 1110_2$

"if a is b's 2's complement
⇒ b is a's 2's complement"

-14 : 1110
↳ 0001 ← taking "1's compliment"
+ 1
0010

subtraction by addition
if we use diff notations
we will get diff answers
so how are we supposed to
know what it is.

$+14$: 0 1110
-14 : 1 0010

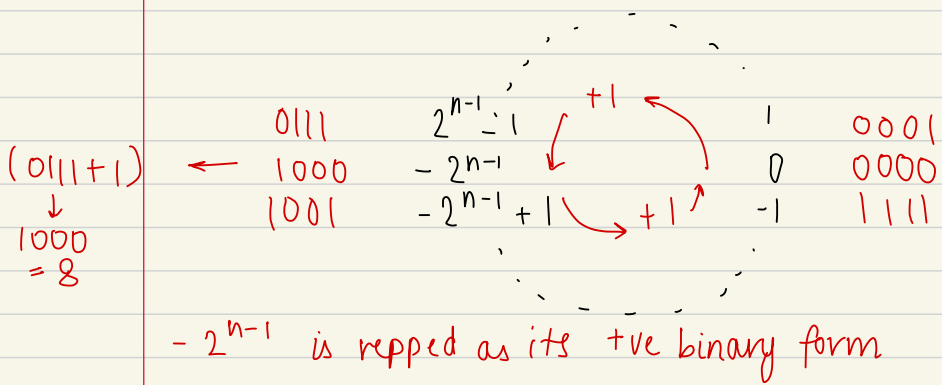
→ shortcut

from LSD copy bits until you hit 1 and then invert
↑
which you also copy

11010 ⟶ 00110
↳ 00101
00110

→ n reps $-(2^{n-1})$ to $(2^{n-1}-1)$ ∴ there is 1 more −ve value than the +ve ones.
(an n bit number)

and there's only one rep. of zero



-2^{n-1} is repped as its +ve binary form

L4 practice problems

a) $1010 \rightarrow A_{16}$

$0011 \rightarrow 3_{16}$

$1111_2 \rightarrow D_{16}$

b) $1000 \rightarrow 8_{16}$

$1011 \rightarrow B_{16}$

$111011_2 \rightarrow 3B_{16}$

binary to dec via division by 10_{10} ↖ but it should be 10 in binary = 1010

10 ↘

1010	1111	← 15
1010	1	1 } 3
	0	1

01000010

4 2

$6+4=10$

2	111	
2	5	1
2	2	1
2	1	0
	0	1

111011

$00000111 - 1 = 00001000$

$0110 + 0100 = 01010$

$1+2+8$

$40 = 2 \times 20 = 2 \times 2 \times 10 = 2 \times 2 \times 5 = 101000$

$1010 + 1110 = 11000$

0010

1101

1110

28

$8 + 32$

2 $8-6-2=-8$

$0110 \quad 0101+1 \quad 1110$

Use only the necessary number of bits in each case.

	Signed decimal	Unsigned binary	**sign-magnitude** representation	**2's complement** representation
a	-127	1111111	1 1111111	1 0000001
b	-112	0111 0000	1 0111 0000	1 0001 0000
c	-60	0011 1010	1 0011 1010	1 1000110
d	-30	1 1110	1 11110	1 00010
e	30	1 1110	0 11110	0 11110
f	60	0011 1010	0 0011 1010	0 0011 1100
g	112	0111 0000	0 0111 0000	0 0111 0000
h	127	1111111	0 1111111	0 1111111

- adding 1 to a 2's complement form of the number actually increases its magnitude by 1

dec	bin mag	2's comp
-8	1000	1000
-7	0111	1001
-6	0110	1010
		⋮

- generalised n-bit system.

$-(2^{n-1})$	100...00
⋮	
-1	111...11
0	00...00
1	000...01
$2^{n-1}-1$	0111...11

- decimal value of a 2's complement number

if +ve : simply convert normally
if -ve : perform 2's complement conversion, and then convert normally

eg. 01001 (0 ⇒ +ve) → 9

10011 (1 ⇒ -ve) ⇝ 01101 → 14

- observations on 2's complement

→ to rep an N-bit number, we need (N+1) bits

→ if theres extra bits, it'll be filled in as the sign bit

→ 2's comp changes a number from +ve to -ve w/o changing its magnitude.
except for the most negative number as its positive counterpart can't be represented in that many bits

→ 2's comp is used because it allows us to carry out subtraction in the same way as addition thereby allowing us to use the same circuit for addition and subtraction

Addition

- align signed bit (no of bits should be the same)
- sign bits are added normally, carry is ignored
- $A + B \equiv A - (-B)$
- n bit
  + n bit
  ---
  n+1 bit we drop the msb
- arithmetic overflow : when the addition of 2 n-bit numbers gives an answer that is greater than n-bits
- detection ↑ : → never if +ve and -ve are being added.
  → if 2 same signed no. are being added, and the result has the opposite sign, overflow is detected
  → never if same signed are being subtracted.

L5 practice problems

2's complement representation

1. The following are signed numbers in 8-bit 2's complement representation.

   a) 0010 0110 = 26 + (32+4+2)
   b) 0111 1111 = 7F + 127
   c) 1000 0001 = 81 − 127
   d) 1111 1111 = FF − 1

   (i) Rewrite the numbers in hexadecimal notation

   (ii) Determine their signed decimal values.

Arithmetic overflow

2. Determine whether arithmetic overflow occurs in each of the following 8-bit 2's complement arithmetic operations:

   E2 + 42 = 24

   a) 1110 0010 + 0100 0010 — no (+ve + -ve)

   imp → b) 0101 1000 – 1000 0010 — yes (+ve − −ve ≡ +ve +ve)

   58 − 82 = 58 + 7E = D6 ≈ 11100110 ← -ve

   0111 1111

   c) 0111 1111 + 0000 0010 — yes

   d) 1110 0001 – 1111 1101 — no (−ve − −ve) ≡ (−ve + +ve)

3. Repeat Q2 using hexadecimal digits.

BCD (binary coded decimal)

- any sum (of digits) can't exceed 9_{10}, otherwise it needs a correction

- ex. 24 + 47 = 71

$11+6 \rightarrow 17 \equiv 16+1$

```
  0010  0100
  0100  0111
  ----------
  0110  1011   <- >9 (not allowed as a single decimal) digit can't exceed 9)
+    1  0110   (11)
  ----------
  0111  0001
```

carry →

correction