CSE 2021 Computer Organization

Appendix C Part 2

Basic Arithmetic Logic Unit -- Adder

Half Adders

- Need to add *bits* {0,1} of A_i and B_i
- Associate
  - binary bit 0 ↔ logic value F (0)
  - binary bit 1 ↔ logic value T (1)

$$\begin{array}{l} C_{i+1} \\ A: A_n \ldots A_{i+1} A_i \ldots A_0 \\ B: B_n \ldots B_{i+1} B_i \ldots B_0 \\ S_i \end{array}$$

- This leads to the following truth table

A_i	B_i	Sum_i	$Carry_{i+1}$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

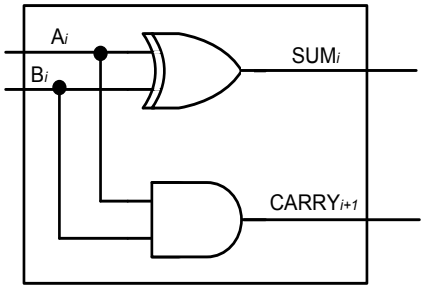
$$SUM_i = \overline{A_i} B_i + A_i \overline{B_i} = A_i \oplus B_i$$

$$CARRY_{i+1} = A_i B_i$$

Half Adder Circuit

$$SUM_i = \overline{A_i}B_i + A_i\overline{B_i} = A_i \oplus B_i$$

$$CARRY_{i+1} = A_i B_i$$



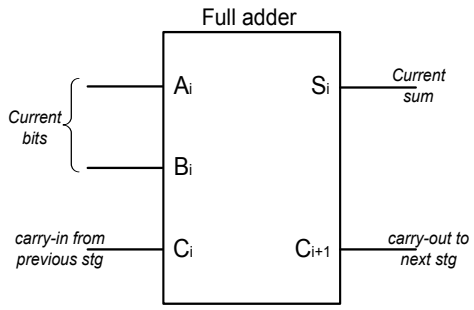
Half Adder Limitations

- Half adder circuits do not suffice for general addition because they do not include the carry bit from the previous stage of addition, e.g.

Carry		0 1 1 0
A		0 1 1 0
B	+	0 0 1 1
SUM		1 0 0 1

Full Adders (1-Bit ALU)

- Full adders can use the carry bit from the previous stage of addition



A_i	B_i	C_i	S_i	C_{i+1}
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

Full Adder Logic Expressions

Sum

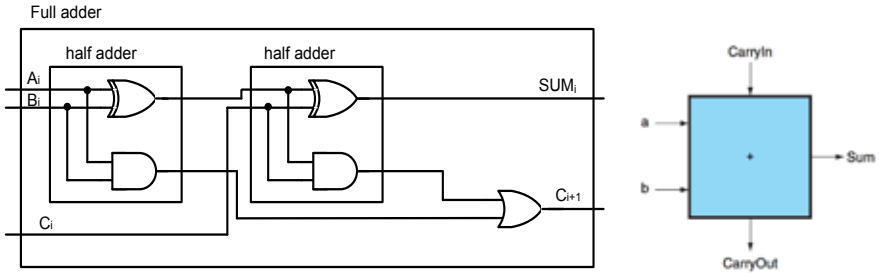
$$\begin{aligned} SUM &= \bar{A}_i\bar{B}_iC_i + \bar{A}_iB_i\bar{C}_i + A_i\bar{B}_i\bar{C}_i + A_iB_iC_i \\ &= \bar{A}_i(\bar{B}_iC_i + B_i\bar{C}_i) + A_i(\bar{B}_i\bar{C}_i + B_iC_i) \\ &= \bar{A}_i(B_i \oplus C_i) + A_i(\overline{B_i \oplus C_i}) \\ &= A_i \oplus B_i \oplus C_i \end{aligned}$$

Carry

$$\begin{aligned} C_{i+1} &= A_iB_i + A_i\bar{B}_iC_i + \bar{A}_iB_iC_i \\ &= A_iB_i + C_i(A_i\bar{B}_i + \bar{A}_iB_i) \\ &= A_iB_i + C_i(A_i \oplus B_i) \end{aligned}$$

Full Adder Circuit

$$SUM = (A_i \oplus B_i) \oplus C_i \qquad C_{i+1} = A_i B_i + C_i (A_i \oplus B_i)$$

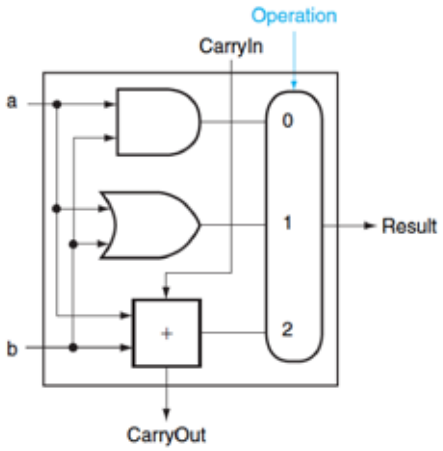


Note: A full adder adds 3 bits. Can also consider as first adding first two and then the result with the carry

Enhancement to 1-bit Adder(1)

- 1-bit ALU with AND, OR, and addition
  - Supplemented with AND and OR gates
  - A multiplexer controls which gate is connected to the output

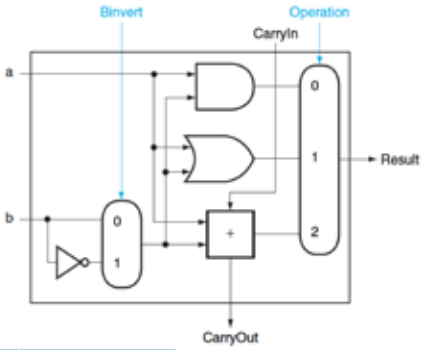
Operation	Result
00	AND
01	OR
10	Addition



Enhancement to 1-bit Adder(2)

- 1-bit ALU for subtraction
  - Subtraction is performed using 2's complement, i.e.

$$a - b = a + \overline{b} + 1$$

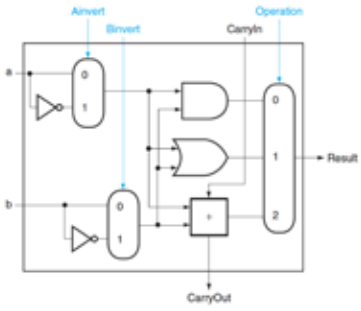


Binvert	CarryIn	Operation	Result
0	0	00	AND
0	0	01	OR
0	0	10	Addition
1	1	10	Subtraction

Enhancement to 1-bit Adder(3)

- 1-bit ALU for NOR operation
- A MIPS ALU also needs a NOR function

$$\overline{(a + b)} = \overline{a} \bullet \overline{b}$$

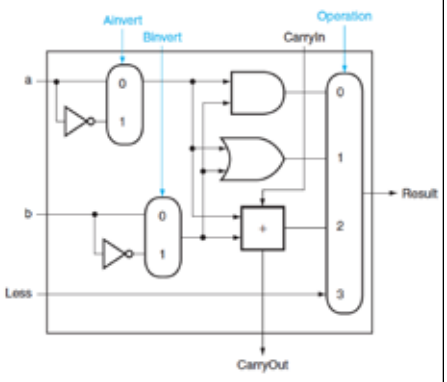


Ainvert	Binvert	CarryIn	Operation	Result
0	0	0	00	AND
1	1	0	00	NOR
0	0	0	01	OR
0	0	0	10	Addition
1	1	1	10	Subtraction

Enhancement to 1-bit Adder(4)

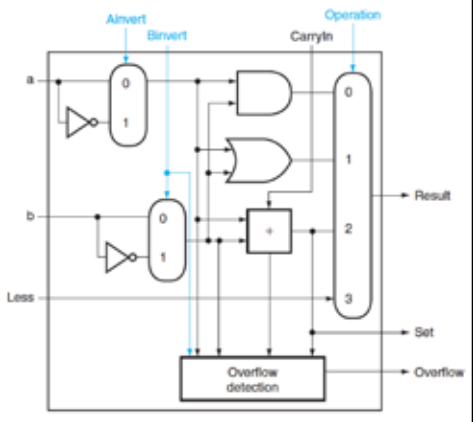
- 1-bit ALU for SLT operations
- slt $s1, $s2, $s3
  - If ($s2<$s3), $s1=1, else $s1=0
- adding one input *less*
  - if (a<b), set *less* to 1

  or if (a-b)<0, set *less* to 1
  - If the result of subtraction is negative, set *less* to 1
- How to determine if the result is negative?

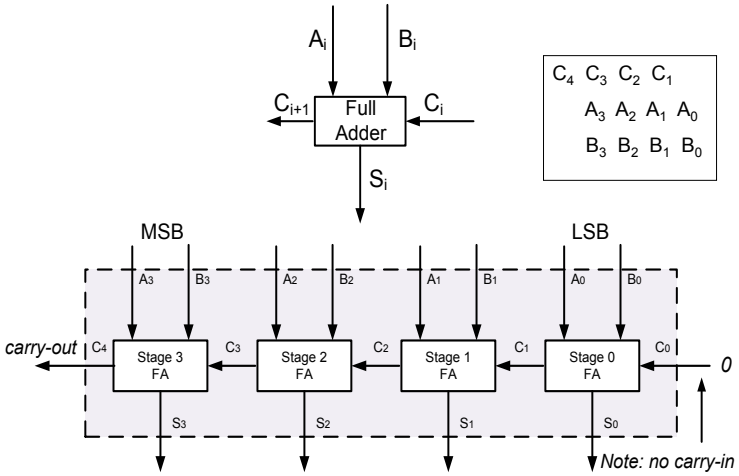


Enhancement to 1-bit Adder(5)

- How to determine if the result is negative?
  - Negative → Sign bit value=1
- Create a new output "Set" and used only for slt
- An overflow detection is included



N-Bit Adders (Ripple Carry)

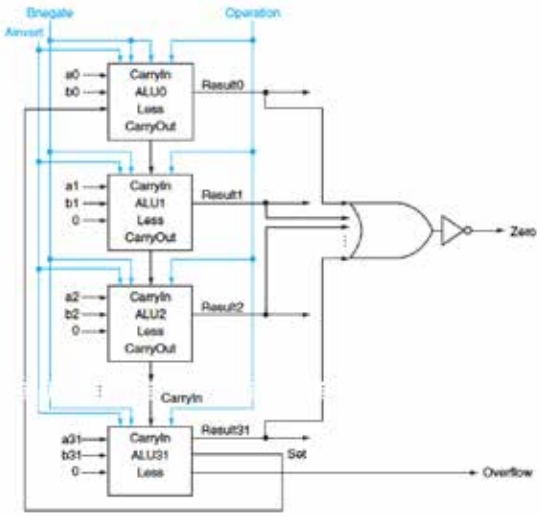


Ripple Carry Adders

- 4 FA's cascaded to form a 4-bit adder
- In general, *N* FA's can be used to form a *N*-bit adder
- Carry bits have to propagate from one stage to the next. Inherent propagation delays associated with this
- Output of each FA is therefore not stable until the carry-in from the previous stage is calculated

32-Bit ALU

- OR and INV gates are added to support conditional branch instruction, i.e. test the result of a-b if the result is 0.



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR