Computer Architecture

Fall, 2023

Week 7

2023.10.23

組別：_____ 簽名：_____

[group5]

1. True or False?
   (a) We can use sub and its signed bit to implement 'set on less than' in hardware.
   (b) MIPS addu and addi instructions ignores overflow.
   (c) When an exception occurs the PC is saved into the EPC register.

Ans:

(a) True, subtract the two numbers and check the signed bit, the signed bit is the result.
(b) False, addi will check for overflow.
(c) True.

[group4]

2. Please select the correct statements.

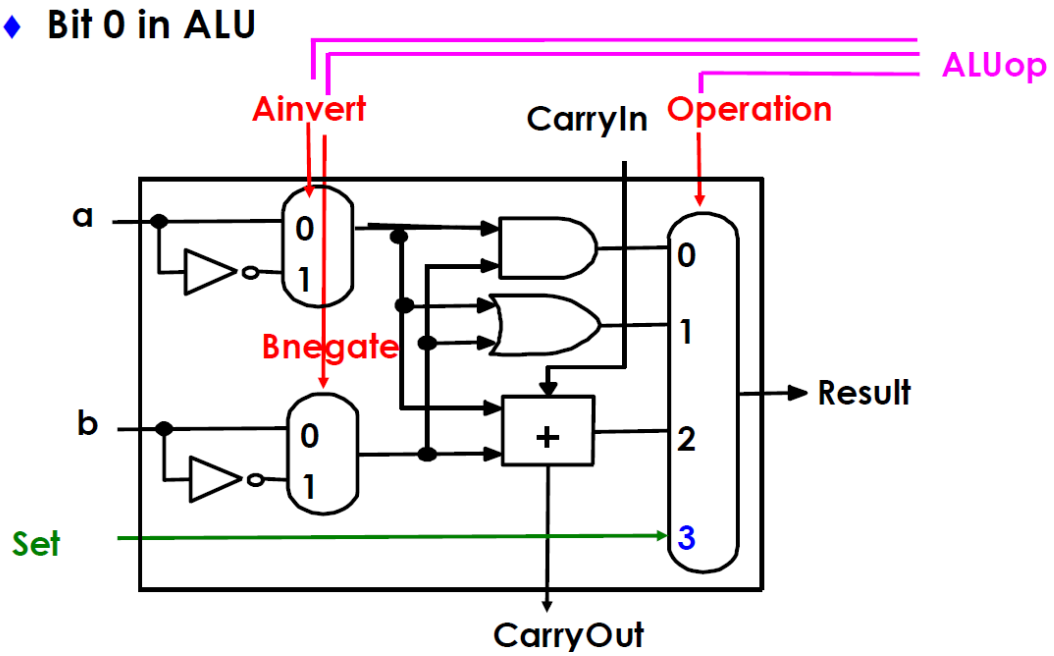
a. We can detect overflow in ALU by using XOR gate.
b. ALUOP 0000 is the control signal of add.
c. We can implement NOR with AND gate and inverters in ALU.
d. The operation of slt is subtract two input numbers and output the result of sign bit in ALU.

Ans:

a. True, since if the Carryin and Carryout of the last bits are different, the overflow happens.
b. False, it is the control signal of AND
c. True, A NOR B is the same as NOT A AND NOT B
d. True, slt is implemented by A subtract B, if the result is negative, then A is less than B. And we can tell the result by sign bit.

[group14]

3. Refer to the given figure. What is the following operator's ALUop code (4 bits)?



Function
and
or
add
subtract
set-on-less-than
nor

Ans:

And	0000
Or	0001
add	0010
substract	0110
Set on less than	0111
nor	1100

[group13]

4. Please explain why the ripple carry adder is slower than the carry look-ahead adder?

Ans.

In the ripple carry adder, the carry must be computed one by one. In the carry look-ahead adder, we can compute the carry for each bit without waiting for the carry to propagate from previous stages. Therefore, the ripple carry adder is slower than the carry look-ahead adder.

[group12]

5.

(1) Please give a brief explanation of the following three design tricks.

(a) divide and conquer.

(b) take pieces you know and try to put them together.

(c) solve part of the problem and extend.

(2) The ALU control of slt(set in less than) is 1110 and the ALU control of nor is 1100, true or false?

(3) Selectors will determine whether the functional gate will operate, so that we can get the correct result directly. True or false?

Ans.

1

(a) Break the problems into simpler problems, solve them and glue together the solution.

(b) solve the basis case and then extend to the whole problem. (ex. recursive/iterative)

(c) use the function we understand to implement complex concepts. For example, A nor B = (not A) and (not B)

2

False. slt refers to 0111, not 1111.

3

False. No matter what the control signal is, the gates still operate. However, the usage of selector is to select the ideal output.

[group8]

6. For these 4-bit signed number, which of them are overflow?

   (a)0010+1010

   (b)1111+1000

   (c)0011+0111

   (d)1010+0100

Ans:

(a)1100 carry in 1 = carry out 1, no overflow

(b)0111 carry in 1 ≠ carry out 0, overflow

(c)1010 carry in 0 ≠ carry out 1, overflow

(d)1110 carry in 1 = carry out 1, no overflow

[group7]

7. True or false, and give a brief explanation.

   (A) Using MIPS instructions such as addu, addui, subu might cause exceptions on overflow.

   (B) “Less” takes on the value of “set” for every bit when performing set-less-than.

   (C) If we connect several 8-bit look-ahead adders to form a 32-bit cascaded look-ahead adder, there will be a total of 8 gate delays.

   (D) Given (p1, g1, p2, g2) = (1,0,1,0) we can know for sure that Cin3 = 0.

   (E) If Cin XOR Cout = 1 for MSB, there is an overflow.

Ans:

(A) F, they do not cause exceptions on overflow

(B) F, less = set for bit 0. Less is 0 for other bits.

(C) T, we need four 8-bit look-aheads in total. There are 2 gate delays per 8-bit, so 8 gate delays in total.

(D) F, Cin3 = g2 + (p2*g1) + (p2*p1*g0) + (p2*p1*p0*Cin0) = 0 + 0 + g0 + p0*Cin0. Because (p0, g0, Cin0) are unknown, we can not know for sure that Cin3 = 0.

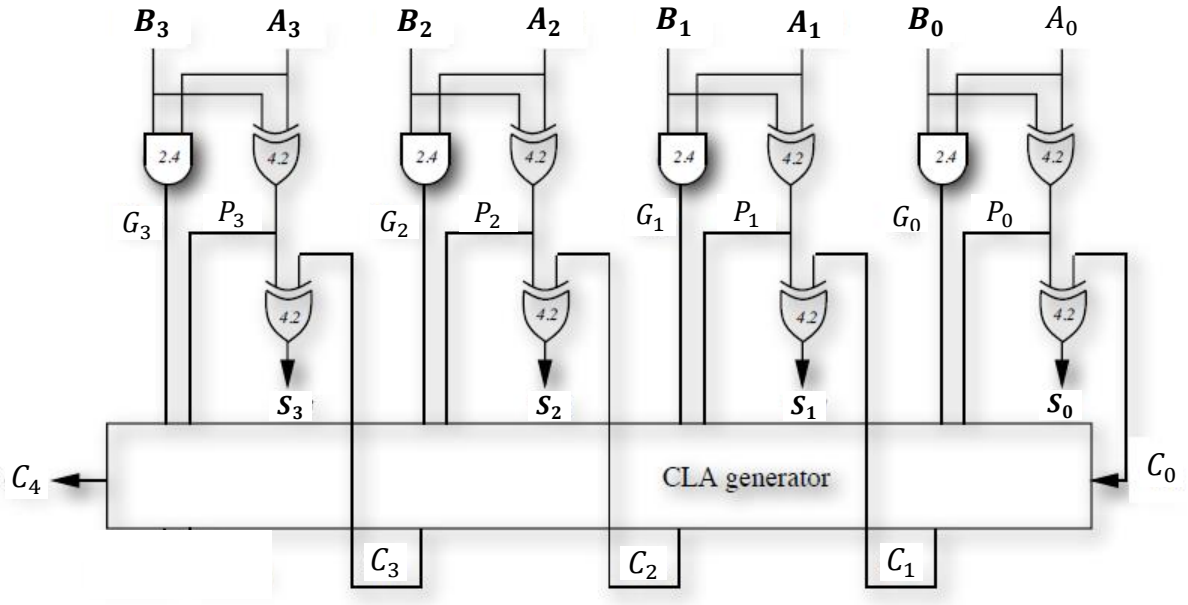
(E) T, overflow if Cin MSB != Cout MSB

[group2]

8. The figure below is a 4-bit Carry Lookahead Adder. The symbol representations go as follows:

- $\boldsymbol{A[3:0], B[3:0]}$: 4-bit inputs
- $\boldsymbol{P_i}$: Carry-Propagate for each 1-bit adder
- $\boldsymbol{G_i}$: Carry-Generate for each 1-bit adder
- $\boldsymbol{S_i}$: Sum for each 1-bit adder
- $\boldsymbol{C_i}$: Carry for each 1-bit adder

	$\boldsymbol{A_i}$	$\boldsymbol{B_i}$	$\boldsymbol{C_i}$	$\boldsymbol{C_{i+1}}$
	0	0	x	0
$\boldsymbol{P_i} = 1$	0	1	0 or 1	$\boldsymbol{C_{i+1}}$
$\boldsymbol{P_i} = 1$	1	0	0 or 1	$= \boldsymbol{C_i}$
$\boldsymbol{G_i} = 1$	1	1	x	1



(1) Please express $\boldsymbol{P_i}$ and $\boldsymbol{G_i}$ (in Boolean expressions) using $\boldsymbol{A_i}$ and $\boldsymbol{B_i}$.

(2) The carry-out for each bit, $\boldsymbol{c_{i+1}}$, can be determined by $\boldsymbol{P_i}$ and $\boldsymbol{G_i}$. Express $\boldsymbol{C_{i+1}}$ using $\boldsymbol{P_i}$ and $\boldsymbol{G_i}$.
*(*HINT: $\boldsymbol{C_{i+1} = A_iB_i + (A_i \oplus B_i)C_i}$)*

(3) Using the result in (2), write down the formula for $\boldsymbol{C_1}$.

(4) Write down the formula for $\boldsymbol{C_2}$ (need to use the result in (3)).

Ans:

(1) $\begin{cases} \boldsymbol{P_i = A_i \oplus B_i} \\ \boldsymbol{G_i = A_i \cdot B_i} \end{cases}$

(2) $\boldsymbol{C_{i+1} = A_iB_i + (A_i \oplus B_i)C_i = G_i + P_iC_i}$

(3) $\boldsymbol{C_1 = G_0 + P_0C_0}$

(4) $\boldsymbol{C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0)}$