ECE 30
Introduction to Computer Engineering

Lab #7 Problems
Spring 2014

1. **Computer Arithmetic**

   (a) What decimal number do these 32 bit 2's complement binary numbers represent? **Note**: You need not simplify your answer.

   - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 1111_2$
   - $0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 1111_2$
   - $1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 0000\ 1100_2$

   (b) A is an 8-bit integer in the two's complement representation and B is a 4-bit two's complement integer. Which one is larger, if $A = 11011101_2$ and $B = 1101_2$? What is the 8-bit two's complement representation of B?

2. **Logic Design** The *super-majority gate* is a 4-input logic gate whose output is 1 only if three or more of its four inputs are 1.

$$F = SM(A, B, C, D)$$

   For example, if $A = B = C = 1$, then $F = 1$, but if $C = D = 0$, then $F = 0$.

   (a) Fill in the truth table below with all those input values for which the output is 1. (The table may have some extra rows. You can leave them blank).

A	B	C	D	F
				1
				1
				1
				1
				1
				1
				1

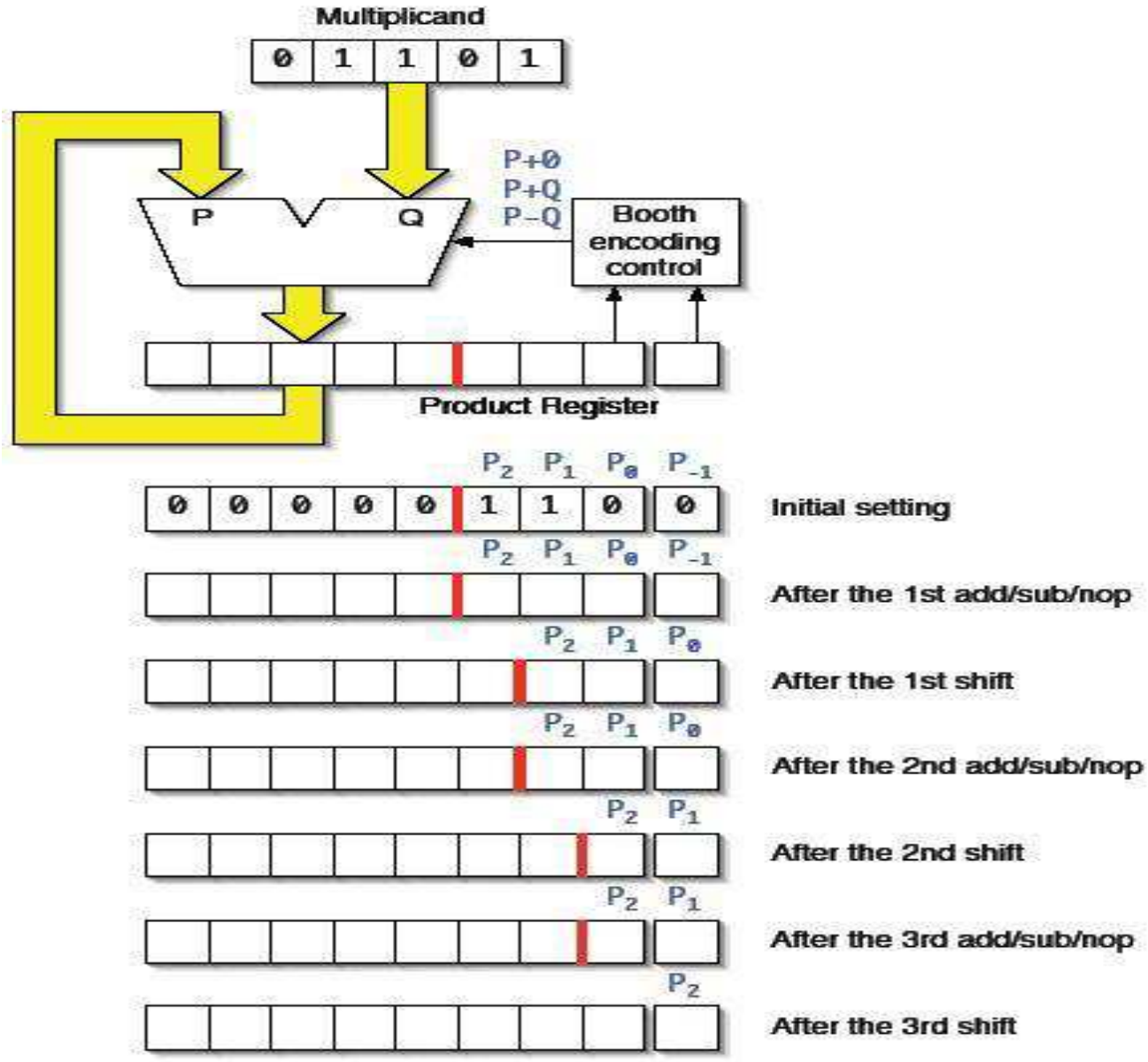
   (b) Draw Karnaugh Map (K-map) for the above truth table.

   (c) Now, write F in terms of its MINIMAL sum-of-products form.

3. **Booth Encoding**

   Trace the steps of multiplying a 5-bit two's complement **multiplicand** 01101_2 with a 3-bit two's complement **multiplier** 110_2, using Booth's algorithm. Assume that the shift-add multiplier shown below is used. Initially, the 5 most significant bits of the 8-bit product

register contain 00000_2 and the 3 least significant bits contain the multiplier (110_2). When the algorithm completes, the product register must contain the 8-bit product in two's complement representation. Complete the table below.

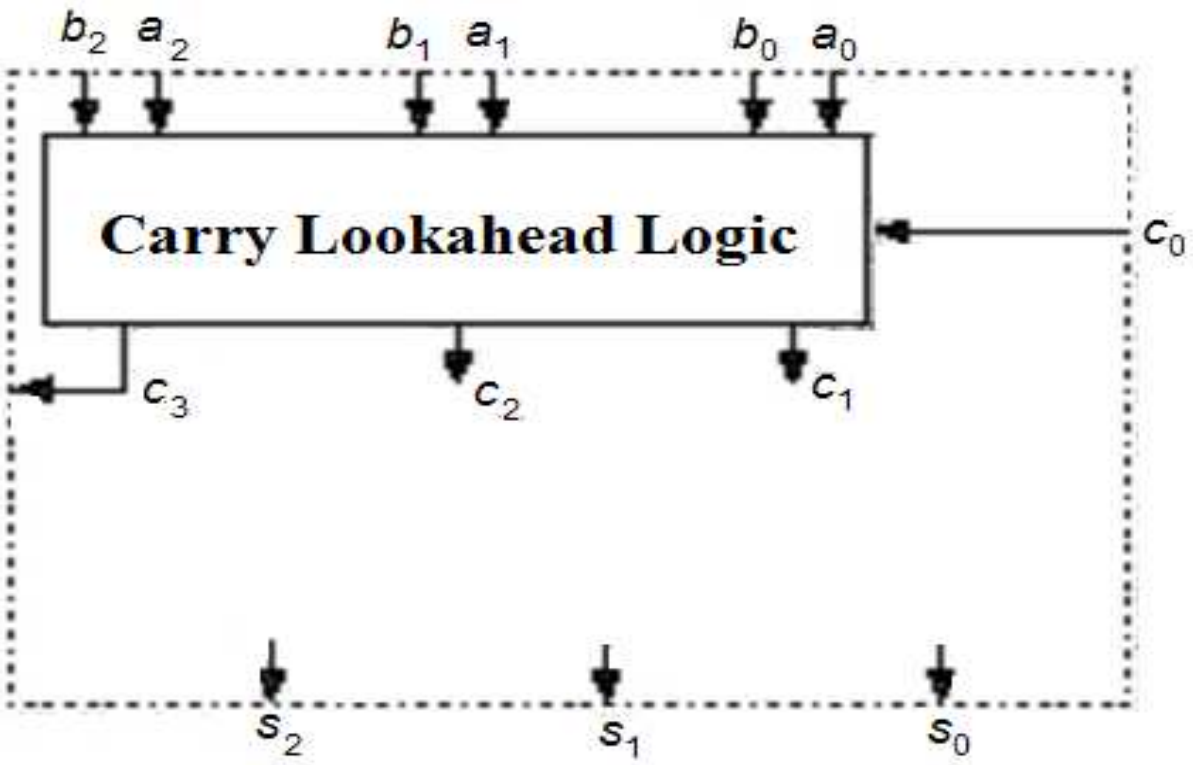


4. **Fast Adder Design**

In this problem, you are to design an 8-bit adder using 3-bit carry lookahead adders (CLA) and 1-bit full/half adders (FA/HA) as building blocks and analyze its performance.

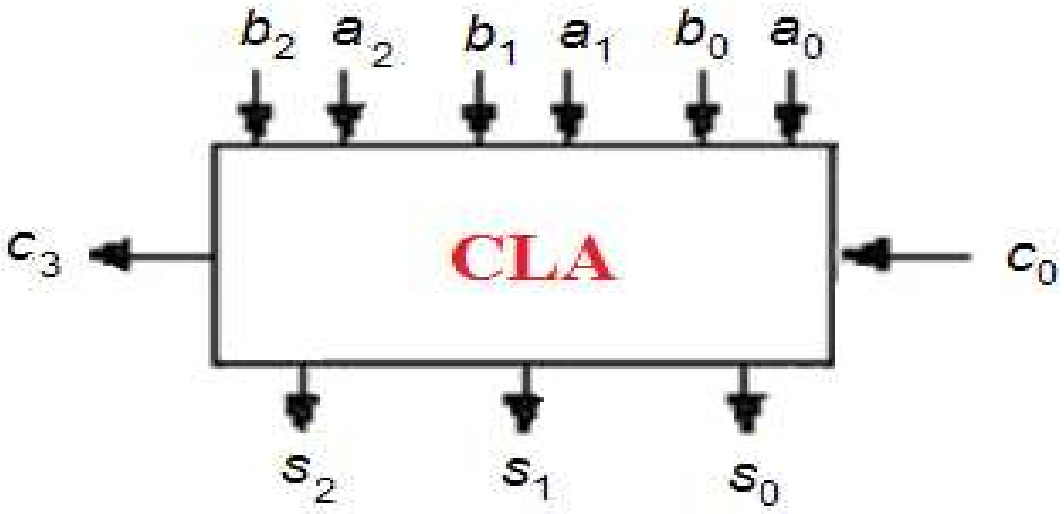
(a) An incomplete block diagram of a 3-bit CLA is shown below. Complete the block diagram by adding the sum logic (use 1-bit full/half adder blocks when constructing the sum logic). Generate the logic equations for c_1, c_2 and c_3 using only g_0, g_1, g_2, p_0, p_1,

p_2 and c_0. Note that $g_i = a_i b_i$ and $p_i = a_i + b_i$.



- $c_1 =$
- $c_2 =$
- $c_3 =$

(b) Construct an 8-bit adder using 3-bit CLA blocks as shown below and 1-bit full adders and half adders as necessary. Optimize your design to minimize the worst-case computation delay.

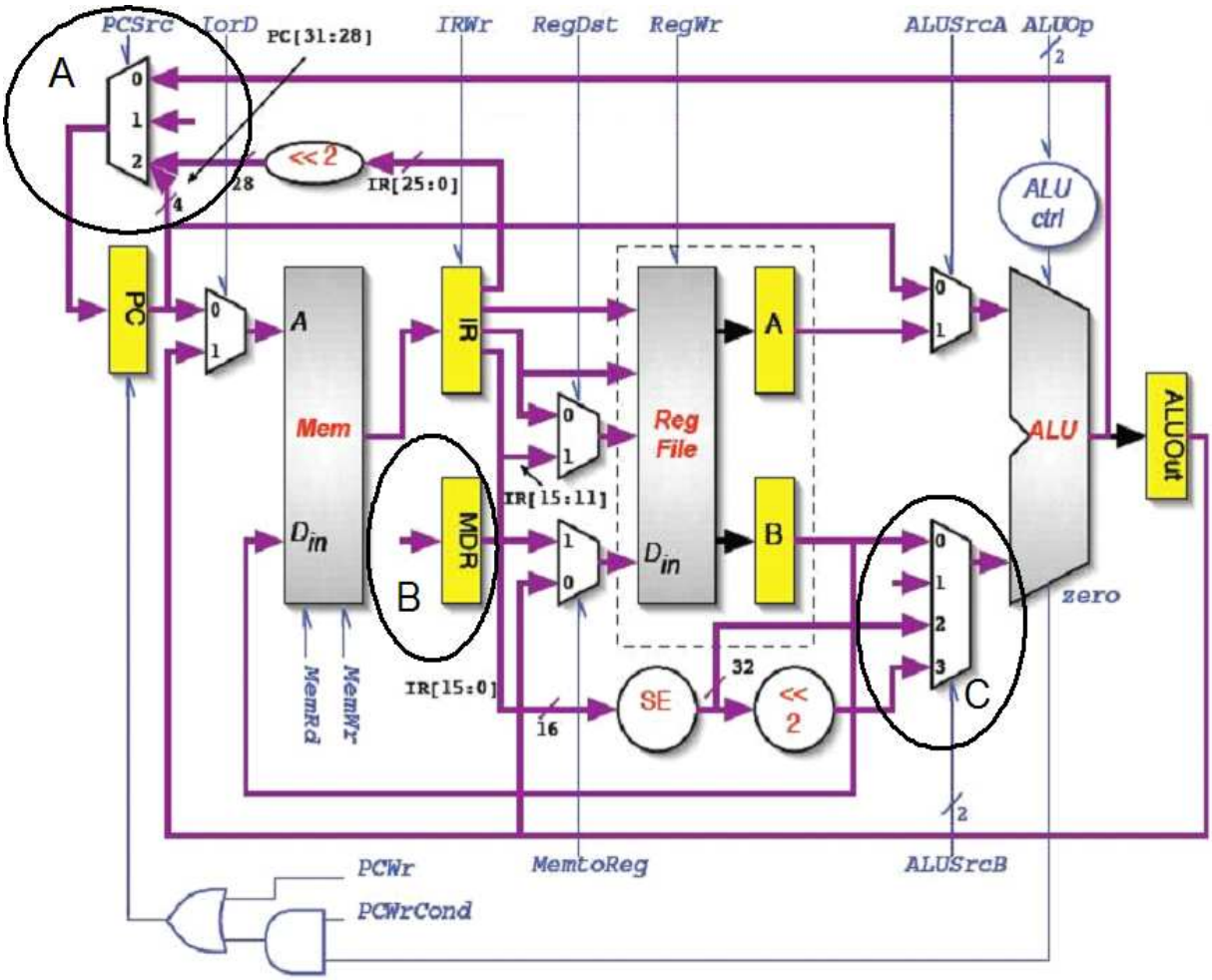


(c) Compute the worst-case delay from changes in inputs to the last sum bit changing for the 8-bit adder in (b). Assume that

- all inputs change simultaneously;
- delay from inputs to $g_i/p_i = \mathsf{T}$;
- delay from $g_i/p_i/c_0$ to $c_i = 2\mathsf{T}$ in the carry lookahead logic;
- delay from a_i/b_i to $c_{i+1} = \mathsf{T}$ in the half adder;
- delay from a_i/b_i to $c_{i+1} = 2\mathsf{T}$ in the full adder;
- delay from c_i to $s_i = \mathsf{T}$.

Give an example of two operands which would cause the worst-case delay.

5.



Shown above is a multi-cycle CPU. This multi-cycle CPU breaks instructions down into the following stages, each of which takes exactly one clock cycle:

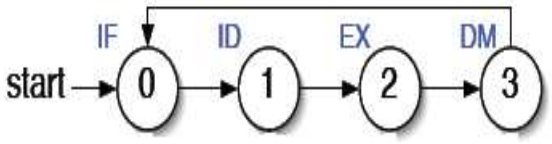
- **IF**: fetch instruction; compute `PC+4` (computed regardless of the instruction type)
- **ID/RF**: decode instruction; fetch source operands; compute branch target (computed regardless of the instruction type)
- **EX**: perform arithmetic / logic operation or compute data memory address
- **DM**: data memory access
- **WB**: write result to register file

Note, however, that there are 3 items/connections missing from the diagram. Each circle (A, B, C) highlights an area missing a single connection/item. In this problem you will fill in the three missing items from the CPU diagram. For each circle:

- Draw the missing connection/item onto the CPU diagram.

- Briefly explain what you added to the diagram.
- List ALL instructions that use the missing connection/item that you just drew (choose from the following basic instructions: `j, beq, add/sub, lw, sw`). For each instruction you listed, specify which stage of the instruction (IF, ID/RF, EX, DM, WB) uses the missing connection/item.

6. Shown below are the different stages of the multi-cycle CPU execution for the `sw` instruction, starting with IF as step 0. Complete the state table by specifying the control signals in binary. Use dashes ("-") for unspecified or don't care values.



	0	1	2	3
IorD				
MemRd				
MemWr				
IRWr				
ALUSrcA				
ALUSrcB				
ALUOp				
PCSrc				
PCWr				
PCWrCond				
RegWr				
RegDst				
MemtoReg				
	IF	ID/RF	EX	DM

7. Compute the number of clock cycles it takes to "execute" the following instructions in a *multi-cycle* implementation, assuming that each of the following phases takes exactly one clock cycle:

IF: fetch instruction; compute `PC+4`
ID/RF: decode instruction; fetch source operands; compute branch target
EX: perform arithmetic / logic operation; compute data memory address; compute branch condition and determine the next PC; determine jump address
DM: data memory access
WB: write result to register file

Complete the table below. Write the phases in the correct order.

Instruction type	No. of clock cycles	Phases (IF, ID/RF, EX, DM, WB)
`lb`		
`sb`		
`beq`		
`j`		

8. Assuming the instruction mix shown below for a *multi-cycle* implementation, what is the average CPI? If the clock period of an alternative *single-cycle* implementation is equal to the time it takes to execute the longest instruction, how much faster is the *multi-cycle* implementation?

Instruction	Frequency	No. of clock cycles
arithmetic / logical	30%	4
lw	20%	5
sw	25%	4
beq	15%	3
j	10%	3