CEC 16th - Past Year Paper Solution *2015 – 2016 Sem 2*
CE2003 – Digital Circuit Design

Solver: Lim Cheng Siang

E-Mail Address: LIMC0183@e.ntu.edu.sg

1. (a)

(i) *a* has 8 bits in total.
Integer bits: **1**
Fractional bits: **7**

-1 to 0.9921875

This will satisfy $-0.99 \leq a \ \leq 0.99$ (as stated in the question.)

b has 12 bits in total.
Integer bits: **4**
Fractional bits: **8**

-8 to 7.99609375

This will satisfy $-4.5 \leq b \leq 3.5$ (as stated in the question.)

c has 17 bits in total.
Integer bits: **2**
Fractional bits: **15**

-2 to 1.999969482

This will satisfy $-2 \leq c < 2$ (as stated in the question.)

Pro-tip
To get the maximum fractional value: $1 - 2^{-f}$ *where f is the number of fractional bits. To get the minimum fractional value:* 2^{-f}.

(ii) As *a* is multiplied by *b*,
Integer bits: 1 + 4 = 5
Fractional bits: 7 + 8 = 15

In addition to *c*,
Integer bits: 5 + 1 = 6
Fractional bits: 15 (remain the same)

$\therefore SIZE = 6 + 15 = \mathbf{21}$
$\mathbf{-32 \leq tmp \ \leq 31.999969482}$

(iii) Fractional bits: 13 (*tmp*[14:2])
Integer bits: 3 (*tmp*[17:15])

$$\mathbf{0 \le \boldsymbol{out} \le 4.999877929}$$

(iv) There is no need to investigate the minimum possible output value as it is negative which would result in getting zero as intended anyway. On the other hand, the maximum possible output value is positive and the circuit should output this value.

$-0.99 \times -4.5 + 2 = 6.455$
$\therefore tmp < 6.455$

Hence, the result will be misinterpreted as the maximum possible output value (based on the ranges of the original inputs provided by the question) is beyond the maximum range of *out* (i.e. 4.999877929).

FYI: The problem lies with the lack of integer bits as previous parts of the question ask for the *maximum possible precision*, thus the number of fractional bits is favoured over that of integer bits.

(b)

a	*b*	*c*	*d*	*f*	
0	0	0	0	**0**	**6**
0	0	0	1	**1**	
0	0	1	0	**1**	
0	0	1	1	**0**	
0	1	0	0	**0**	**C**
0	1	0	1	**0**	
0	1	1	0	**1**	
0	1	1	1	**1**	
1	0	0	0	**0**	**8**
1	0	0	1	**0**	
1	0	1	0	**0**	
1	0	1	1	**1**	
1	1	0	0	**0**	**A**
1	1	0	1	**1**	
1	1	1	0	**0**	
1	1	1	1	**1**	

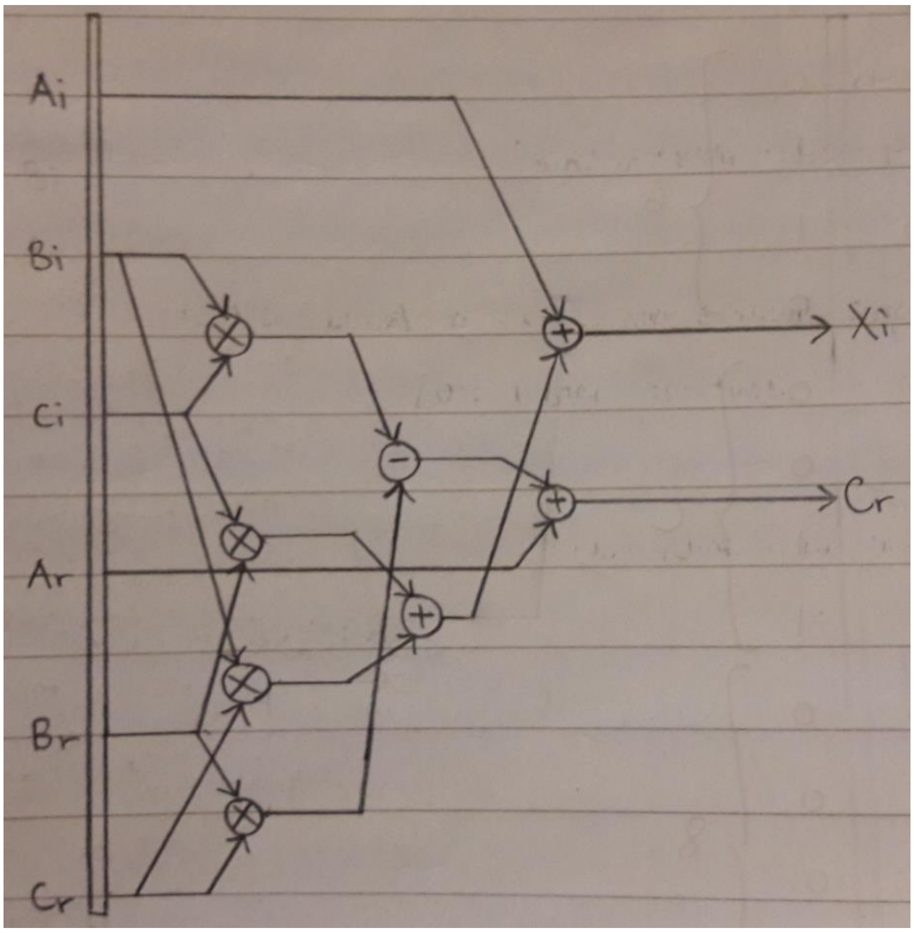
Ans: **16’hA8C6**

Pro-tip
The bottom bit of the LUT content is the most significant bit and the top bit is the least significant bit.

2. (a)

$X = A + C \times B$

$C \times B \Rightarrow (BrCr - BiCi) + j(BrCi + BiCr)$
$A + C \times B \Rightarrow \left(Ar + (BrCr - BiCi)\right) + j\left(Ai + (BrCi + BiCr)\right) = Xr + jXi$



(b)

```
always @ * begin
      Xr = Ar + Br * Cr - Bi * Ci;
      Xi = Ai + Br * Ci + Bi * Cr;
end
```

(c)

(i) Latency: 5 + 2 + 2 = **9 ns**
Throughput: **one every 9 ns**
Max operating frequency: $\frac{1}{9} = \mathbf{111.11\ MHz}\ (to\ 2\ d.p.)$

(ii)

```
module pipelined_complex (input signed [15:0] Ar, Ai, Br, Bi,
                                              Cr, Ci,
                          output reg signed [31:0] Xr, Xi);

reg signed [31:0] BrCr, BiCi, BrCi, BiCr;
reg signed [11:0] Ar_in, Ai_in, Br_in, Bi_in, Cr_in, Ci_in;

always @ (posedge clk) begin

      if (rst) begin

            Xr <= 0; Xi <= 0;

      end

      else begin

            // pipeline inputs
            Ar_in <= Ar; Ai_in <= Ai; Br_in <= Br;
            Bi_in <= Bi; Cr_in <= Cr; Ci_in <= Ci;

            // stage 1
            BrCr <= Br_in * Cr_in; BiCi <= Bi_in * Ci_in;
            BrCi <= Br_in * Ci_in; BiCr <= Bi_in * Cr_in;

            // stage 2
            Xr <= Ar_in + BrCr – BiCi;
            Xi <= Ai_in + BrCi + BiCr;
      end
end
endmodule
```

(iii) Latency: **9 ns**
Throughput: **one every 5 ns**
Max operating frequency: $\frac{1}{5} = \mathbf{200\ MHz}$

Pro-tip
Don't let the diagram scare you!

3. (a)

(i) We can reuse the *sw* instruction whereby *rs* is replaced by *sp*, and because for *push* instruction the stack pointer will always decrease by 1, the immediate value will be replaced by -1 (*rs* + *immd*). Unlike the *sw* instruction, the stack pointer has to be updated after the decrement, thus the ALU result has to be written to *rs* where the stack pointer is stored. Hence, we can use a 2:1 mux to choose between *rd* and *rs* as the write register accordingly.

The new instruction will involve all 5 stages of the pipeline to complete as the stack pointer will only be updated at the write-back stage.

S1: **0** (This is not a branch instruction.)
S2: **1** (An immediate value of -1 is required.)
S3: **0** (The ALU result has to be written back to the register source.)

(ii) The stack pointer is only updated after the word has been loaded, thus this is impossible as the ALU is placed before the data memory.

(b) `LLI: ALUResult = ` **`ALU_Data1;`**

`LLU: ALUResult = ` **`{ALU_Data1, ALU_Data2[7:0]};`**

(c) Data forwarding will not work because for the load instruction, the $2 value is only available after the data has been accessed, but it is needed by the next instruction before it is available.

```
add $1, $3;
lw  $2, 0($1);
add $5, $1;
add $4, $2;
```

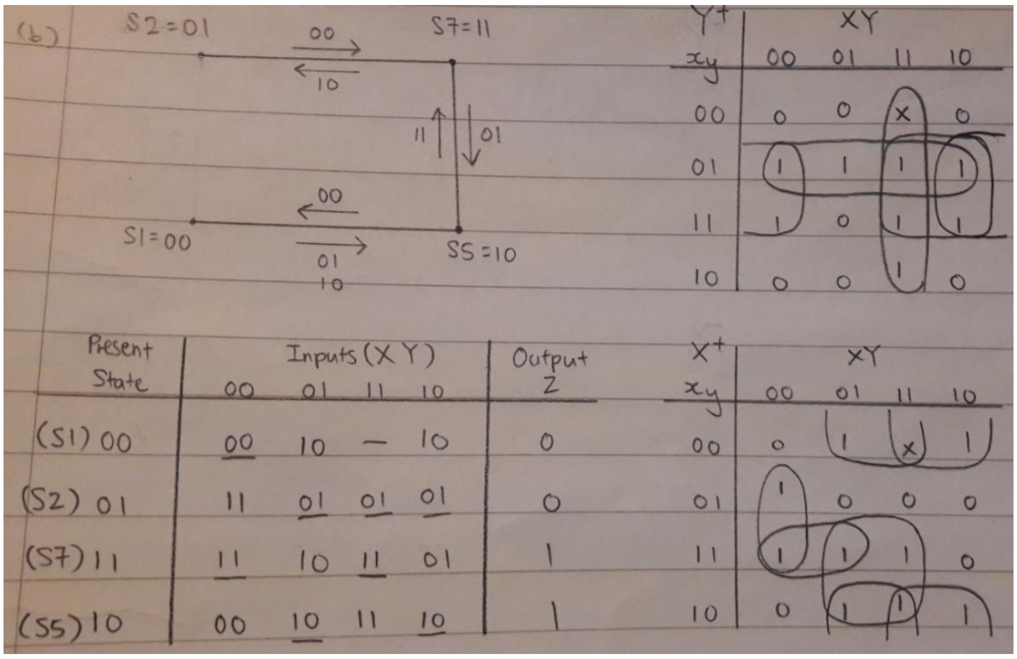
4. (a)

- Merging procedure:
  - Two states, S_i and S_j, (rows of flow table) are compatible if:
    - Both S_i and S_j have the same output condition, AND
      - The successor of S_i and S_j are both the same, OR
      - The successor of S_i and S_j are both stable, OR
      - A successor of S_i and S_j (or both) is unspecified
- States that are compatible can then be merged.

P1 = (S1 S2 S3 S4 S5 S6 S7 S8)		(all states)
P2 = (S1 S2 S3 S4) (S5 S6 S7 S8)		(same output conditions)
P3 = (S1) (S2 S3 S4) (S5 S6) (S7 S8)		(compatible states)

Present State	Inputs (X Y)				Z
	00	01	11	10	
S1	**S1**	S5	—	S5	0
S2	S7	**S2**	**S2**	**S2**	0
S5	S1	**S5**	S7	**S5**	1
S7	**S7**	S5	**S7**	S2	1

(b)



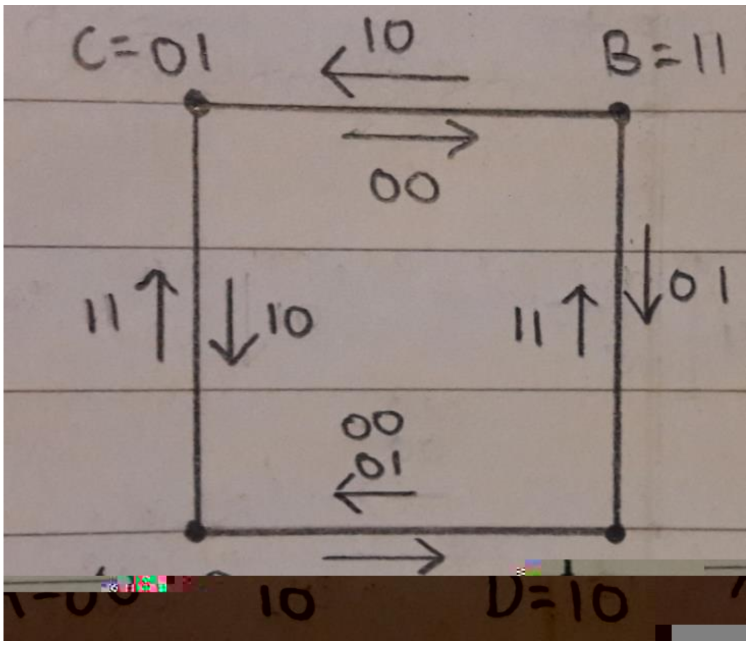
Present State	Inputs (X Y)				Output Z
	00	01	11	10	
(S1) 00	00	10	—	10	0
(S2) 01	11	01	01	01	0
(S7) 11	11	10	11	01	1
(S5) 10	00	10	11	10	1

Y+ xy \ XY	00	01	11	10
00	0	0	X	0
01	1	1	1	1
11	1	0	1	1
10	0	0	1	0

X+ xy \ XY	00	01	11	10
00	0	1	X	1
01	1	0	0	0
11	1	1	1	0
10	0	1	1	1

$$Y^+ = yY' + x'y + XY + yX$$
$$X^+ = yX'Y' + xyX' + xY + y'Y + y'X$$
$$Z = x$$

(c) If **A = 00**, then **C = 01**, **D = 10**, and **B = 11**. This would be an issue for B as it cannot transit to A when the input is 01. Fortunately, the next state for D is a "don't care" so B can actually transit to D and then from D transit to A. The above assignment would also be an issue for C as it cannot transit to D when the input is 10. Fortunately, again, the next state for A is D, so C can actually transit to A and then from A transit to D. With these modifications, the state assignments would then be hazard free. Below is the evidence of this.



END OF SOLUTION

*I would like to thank **<u>Jiang Xiaoxuan</u>** for encouraging me to be a PYP solver! Also, thank you for always believing in me! As promised, there I've done it. ☺*

Some words of wisdom from me…

<u>THIS MODULE IS IMPORTANT FOR CE/CZ3001 ADVANCED COMPUTER ARCHITECTURE, SO BETTER GET USED TO IT!</u> GOOD LUCK AND ALL THE BEST TO YOUR EXAMS!!! LOOK FORWARD IT!!! HAVE FUN!!! WOOOOOOOOOOOOOOHOOOOOOOOOOOOOO!!! ☺