[Based on Question #7 of Summer 1993 Midterm] Remove TARGET register, add ZERO FF:

Please refer to figures 5.39 (page 323) and 5.47 (page 332) in your book (1st edition). To reduce the cost of hardware even at the loss of some performance, the target register has been removed, the mux following the target register has been converted to a 2-to-1 mux as shown below and a status flip-flop has been added to capture the ZERO output from the ALU.

Note: To distinguish from another question (#6) in this assignment (Q#7 of Summer 1996 Midterm), here you are NOT allowed to take the ZERO output of the ALU to the control unit as an input.
The consequent modifications to the state diagram of fig. 5.47 is only partly carried out below. Please carry out the state diagram modification completely to handle the `beq` instruction. Add additional states as needed. Activate \texttt{Z\_FF\_WRITE} control signal in an appropriate state. Revise the \texttt{PCSource} signal values in view of the fact that it is a 1-bit control line now.

Your TA asks you to think critically and decide whether is possible to avoid using the \texttt{Z\_FF\_WRITE} control signal and let the \texttt{ZERO\_FF} write at the end of every clock (like the ALUout and MDR registers of the 2\textsuperscript{nd}/3\textsuperscript{rd} edition design). Your answer:

___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
2  [Based on Question #4 of Summer 2002 Midterm] New Register file with one READ PORT:

Please consider a **modified register file which has only 1 read port**. It requires you to read one register at a time.

2.1 **Scenario 1**: Given on the next page is the completely modified datapath. Given below is a modified *incomplete* state diagram. We assume that the clock has extra slack to account for the reading time of the other register and immediately using it as an operand for ALU operation. Please complete the incomplete state diagram. Simply fill the values for rs/rt and R_Write. Use "X" for "don't care" whenever possible to allow logic minimization.

**Scenario 1 State Diagram**

![Scenario 1 State Diagram](image-url)
Scenario 1 Datapath (Complete)
2.2 **Scenario 2**: In a more realistic design, we do not expect so much extra slack in clock. So perhaps an extra state called **state 1a (after state 1)** can be included to allow fetching of the other register. Please read the discussion below.

Miss Bruin : Every instruction should pass through state 1a

Mr. Bruin : Not all instructions need two source registers. For example, *jump* does not need any source registers. *sw* needs 2 source registers but *lw* needs only one source register.

Miss Trojan: Well *sw* does not need the second source register until state 5 where memory is written. Activity in state 2 (effective address computation) does not depend on the second register. So let us combine state 2 and the proposed state 1a into the new **state 2n (2 new)**.

Mr. Trojan : Do you want an extra state for *jump* also?

Show your modified state diagram for scenario 2 of this question to suit the modified datapath of the 1st edition (scenario 2) of this question.  
Note: Here we use the *temp register* to capture *(rs)* (not *rt)*.

2.3 In scenario 1, we have provided a temporary register to capture contents of *"rt"* where as in scenario 2, we have provided a temporary register to capture contents of *"rs"*. Is this option to capture *(rs)* in temporary register rather than *(rt)* an important choice to this (scenario 2) design or is it an insignificant random choice? Explain.
Scenario 2 - DATAPATH (Complete)
Complete the state diagram. When you fill the values for \( rs/rt \) and \( R_{Write} \) use "X" for "don't care" whenever possible to allow logic minimization.

**Scenario 2 - STATE DIAGRAM**
3  [Question #2 of Spring 1994 Midterm] JR rs implementation:
Given below is the JR rs (jump register) instruction format. It causes an unconditional jump to the address contained in rs register.
The RTL (Register Transfer Language) statement for the same is \((PC) <= (rs)\)
Also given below is the instruction format for ADD rd, rs, rt instruction.

Table 1: Jump Register Format

<table>
<thead>
<tr>
<th>SPECIAL</th>
<th>rs</th>
<th>0</th>
<th>JR</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-26</td>
<td>25-21</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>000000</td>
<td>rs</td>
<td>000000000000000000000000</td>
<td>001000</td>
</tr>
</tbody>
</table>

Table 2: ADD Format

<table>
<thead>
<tr>
<th>SPECIAL</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>0</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-26</td>
<td>25-21</td>
<td>20</td>
<td>15</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>000000</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>0000</td>
<td>10000</td>
</tr>
</tbody>
</table>

Reproduced on the next page is the datapath for the multicycle CPU implementation from your textbook (Fig. 5.39/P323 - 1st edition) modified to include the function-filed connection to the control unit. Please note that the control unit shall now distinguish JR instruction from R-Type instructions by looking at the function-filed.

3.1 Show how the JR rs instruction can be implemented without any modification to the datapath (except for the function-filed connection to the control unit).
Hint: How about \((PC) <= (rs) + ($0)\) ?

In the case of the JR rs instruction format, what do you have in the place of rt field position of the ADD instruction? Does it help?

Note: If you can not think of a way to implement JR rs instruction without any modification to the datapath, then you are free to suggest and show the modifications to datapath desired by you. Briefly explain your modifications to the datapath only if you modify the datapath.
3.2 Complete the modification to the state diagram for the control unit of the multicycle CPU (Fig. 5.47/P332 of the 1st ed. reproduced below) to accommodate JR rs instruction. Note: Your modification to the state diagram will be judged based on your datapath.

You need to decide if the ALU Control block needs to be modified or not and explain why it needs to be modified if it needs to be modified or why it does not need to be modified. If you need to modify it, you do not need to show the details of the proposed modification. Simply narrate the proposed modification in words.
4 [Question #2 of Summer 1995 midterm] **SWAP instruction implementation:**

In this question we are trying to implement the *SWAP* instruction on the multi-cycle CPU. Instead of using one of the general purpose registers in the Register File as a temporary register, Mr. Trojan suggests that the *TARGET* register is used as a temporary register. Though the *TARGET* register holds the Branch Target Address by the end of the state 1, once you decode and recognize that the current instruction is a *SWAP* instruction, you may safely overwrite the contents of the target register.

Proposed format of the SWAP instruction: Note that the *rs* field is repeated twice in the format.

<table>
<thead>
<tr>
<th>opcode swap</th>
<th><strong>rs</strong></th>
<th><strong>rt</strong></th>
<th><strong>rs</strong></th>
<th>xxxxxx</th>
<th>xxxxxx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>31</td>
<td>26</td>
<td>25</td>
<td>21</td>
<td>20</td>
</tr>
</tbody>
</table>

4.1 A modified data path is provided in Fig 4.1. Please notice the *muxes I, II, and III*. State the use/purpose of these muxes so far as the execution of the *SWAP* instruction is concerned.

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

4.2 Add a set of states to the state diagram in Fig. 4.2 for the execution of the *SWAP* instruction. Label the states as State 10, 11, ..., etc. State what is achieved in each of the states in the space provided below. You do NOT need to find the actual values of the control signals. It is enough.
to state something like, "access \((\text{\$rs})\) and \((\text{\$0})\) from the register file", "compute \((\text{\$rs}) + (\text{\$0})\) using ALU", etc. Note that, in the original implementation of an R-type instruction, we used state 1 to access registers, state 6 to perform the operation by the ALU, and state 7 to place the result in the destination register. We did not perform all the three operations in one clock cycle/state as one clock would not be long enough at the high frequency of our CPU clock. Please follow the same philosophy here also.

**State 1:** Decode and identify the swap instruction

**State 10:**

**State 11:**

**State 12:**

**State 13:**

**State 14:**

**State **:

4.3 Consider the \textit{SWAP} instruction and the single-cycle CPU of chapter 5. Explain why it is \textbf{not} possible to implement the swap instruction \textbf{without} a modification of the Register file.
Add more states if required

\[ \text{Add more states if required} \]
Assume that the ALU is SLOW and needs two clock cycles to produce its result. Accordingly on the next page, we have added extra states called 0A, 1A, 2A, 6A, and 8A after states 0, 1, 2, 6, 8.

Instruction decoding can now be done in 0A state. So the anticipatory BTA (Branch Target Address) calculation is no more anticipatory. It is done only if the instruction is a BEQ instruction.

Complete the state diagram on the next page by completing the states 0A, 1A, 2A, 6A, and 8A. You can write in the state circles, "same as in state ABC", "all signals of state XYZ except signal PQR", etc. If you need to modify any of the original states (0, 1, 2, 3, 4, 5, 6, 8, 9), feel free to do so.
Instruction Decode
Register Fetch
PC incre. completion

0A

Start

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

1A

Instruction decode/ register fetch

ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
TargetWrite

2A

Memory address computation

(ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00)

6A

Execution

(ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10)

(ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00)

(ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00)

8A

Jump completion

(ALUSrcA = 1
ALUSrcB = 00
PCWriteCond
PCSource = 01)

4

Write-back step

(ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00)

(ALUSrcA = 1
IorD = 1
ALUSrcB = 10
ALUOp = 00)

(ALUSrcA = 1
MemWrite
IorD = 1
ALUSrcB = 10
ALUOp = 00)

(ALUSrcA = 1
RegDest = 0
RegWrite
MemtoReg = 1
ALUSrcB = 10
ALUOp = 00)

7
5.1 Complete the "Clocks Taken" columns in the table on the side.

5.2 Target Write Control signal: Can we remove the write control signal on the Target register and write to it at the end of every clock? Answer for each of the two cases separately. Provide brief explanation.

(a) in the case of the original design

___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________

(b) in the case of this revised design.

___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________

5.3 Now suggest optimizing the BEQ execution.

___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
___________________________________________________________________________
6. [Based on Question #7 from Summer 1996 Midterm] Anticipatory "EA" calculation for lw/sw in state 1 instead of anticipatory "BTA" calculation for beq in state 1:
The CPIs under the current CPI column are based on our current multi-cycle CPU implementation.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Frequency f_i</th>
<th>Current CPI_i</th>
<th>New CPI_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>R-type</td>
<td>50%</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td>15%</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Jump</td>
<td>5%</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Mr. Trojan noted that, gaining one clock in load/store instruction is worth even at the cost of losing one clock in branch instructions as load/store occur more often compared to branch. He recommended not to perform the anticipatory calculation of BRANCH ADDRESS (and storing it in the TARGET register) in state 1. Instead, he wanted that we compute the memory address (normally done in state 2) early in state 1 thereby gaining a clock in load/store execution. To complete branch instruction, you need an extra state to calculate the target address.

6.1 Implementation of Trojan’s design:

The datapath on the next page has been revised. The TARGET register has been removed, hence the 3-to-1 mux after the target register was reduced to a 2-to-1 mux. The ZERO inference from the ALU has been conveyed to the control unit and the PCWrite control logic has been modified. Complete the changes to the state diagram. State 2 has been cancelled. New states, 1 (New) and 8 (New), were added in the place of the original states 1 and 8. An additional state 8A has been added. Fill up the control signals as appropriate to these new states 1 (New), 8 (New), and 8A.

6.2 Is it possible that the apparent advantage in reducing the number of clocks needed to perform lw and sw instructions is offset by need for slowing the clock? Did we pack too much stuff in state 1? Can the memory address computation start at the beginning of state 1 or does it need to wait for anything? Explain.

6.3 Assuming that Mr. Trojan’s design finally works out without having to slow down the clock, what is the factor of improvement/speed-up it achieves? Fill-up the new CPI column and calculate. If needed, assume that 50% of the branches are taken branches (successful branches).
No PCWriteCond signal anymore
[Based on Question #2 Summer 2003 Final Exam]: Separating Data Memory:

Reproduced on the next two pages are the datapath and state diagram from the first edition of the text book used in your current lab #4 design.

You are aware that the so-called memory is actually an on-chip cache for high frequency operation. Your boss wants to experiment with different parameters for the instruction cache as compared to the data cache. For example, he wants to use direct mapping for the instruction cache whereas set-associative mapping for the data cache. So your boss asked you to replace the single unified memory in the datapath with separate instruction memory and data memory. Please modify both the datapath and the state diagram as needed. If you wish to use two sets of MemRead and MemWrite control signals use signal names with suffix "_I" for instruction memory and suffix "_D" for data memory (MemRead_I, MemWrite_I, and MemRead_D, MemWrite_D). Your assistants, Bruin #1 and Bruin #2, are arguing about the need for IR (Instruction Register) in the modified datapath and they are "duly" confused!

You decided to __________________ (remove/retain) IR because ____________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________