A
REPORT
ON

MIPS microprocessor design

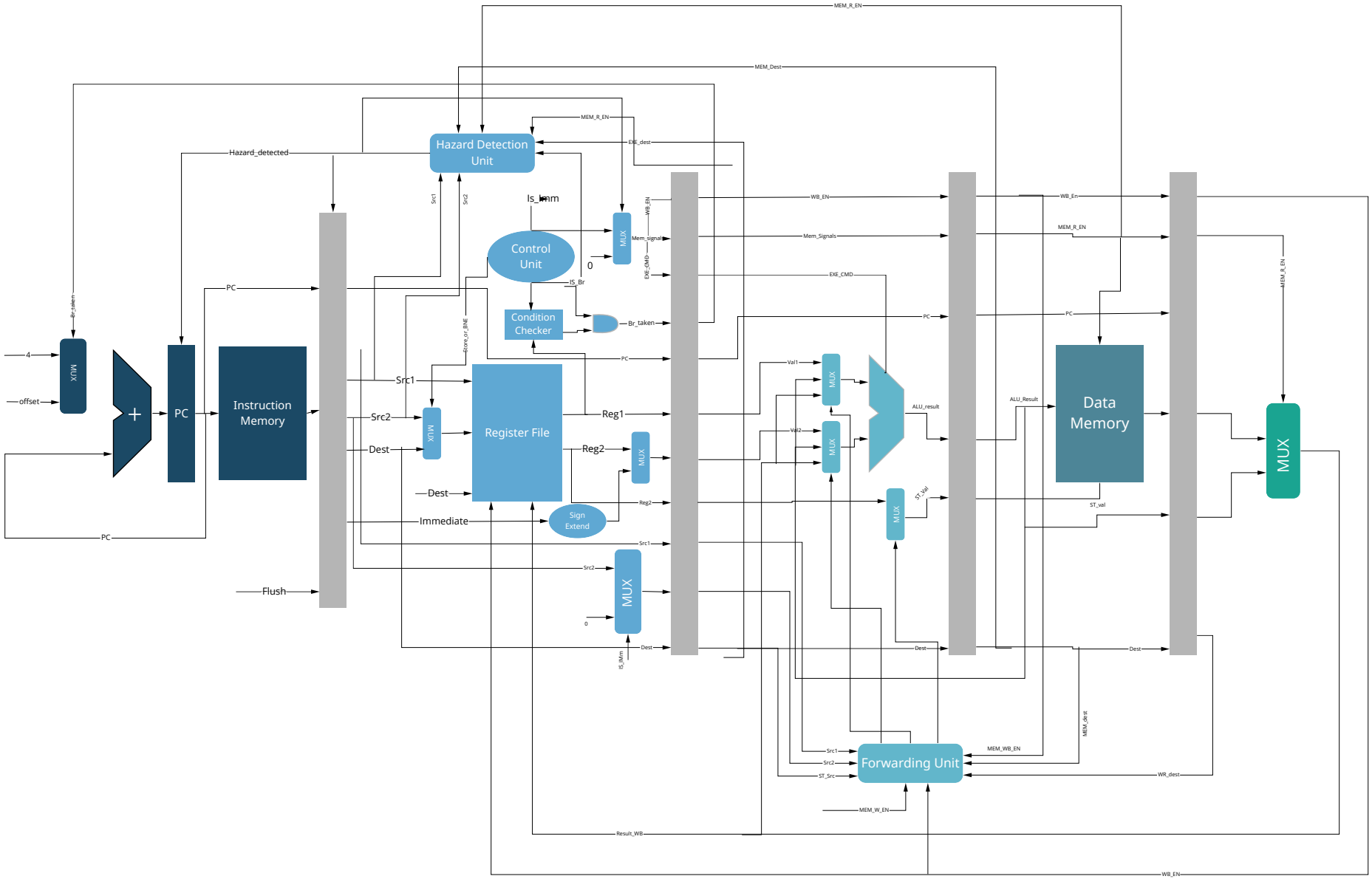
By

Abdultaiyeb Vasanwala	**2019AAPS0279G**
Priyansh Parikh	**2019A3PS0288G**
Dhritiman Sinha	**2019AAPS0005G**

Prepared in partial fulfillment of the
CS F342 - Computer Architecture

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI (GOA)

(Month, Year)



System Architecture

Data hazard mitigation

Considering an instruction to be of the type *operation rd, rs, rt* we can categorize data hazards into two types and the same thing is followed for both operands:

1) The value of the variable calculated in the current instruction is used right after the current instruction has the variable as one of the operands. In this case, while the current instruction is being executed the next instruction is fetched and requires the operand, so the data forwarding unit forwards data to the next instruction from EX/MEM pipeline after getting the 01 control signal from the data forwarding unit.
2) The value of the variable calculated in the current instruction is used after the two instruction has the variable as one of the operands. In this case, while the current instruction is being written back the next instruction is fetched and requires the operand, so the data forwarding unit forwards data to that instruction from MEM/WB pipeline after getting the 10 control signal from the data forwarding unit.

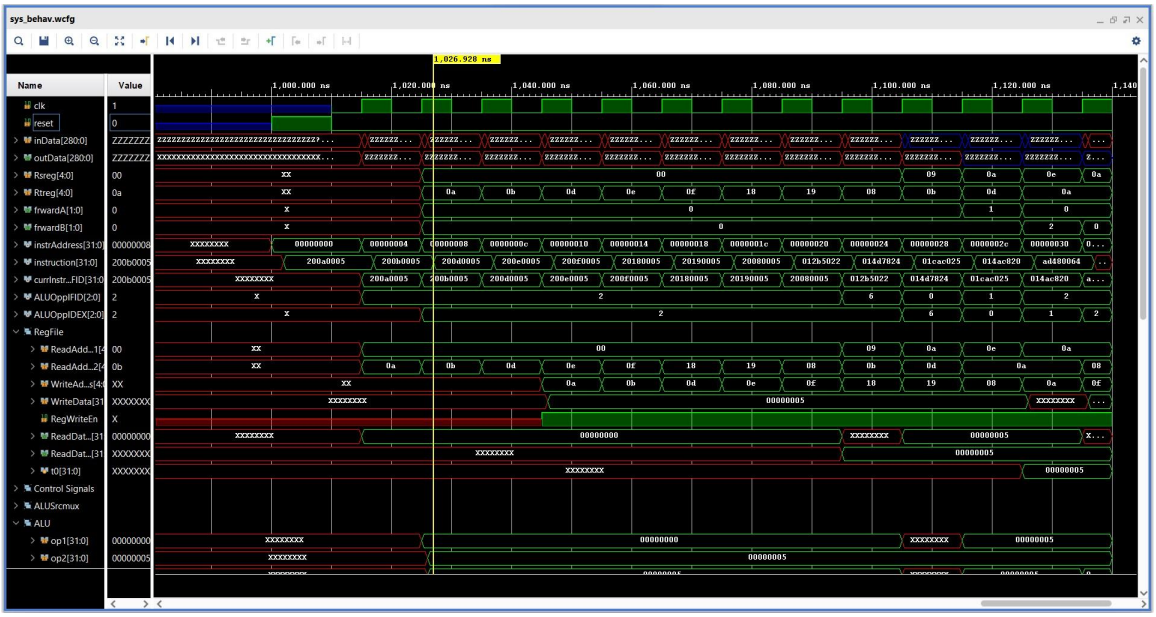
The following code can be used to test the data forwarding unit in the MIPS processor

```
sub     $2 , 11 , $3          # Register $2 written by sub
and     $12 , $2 , $5         # 1st operand($2) depends on sub
or      $13 ,$ 6 , $2         # 2nd operand($2) depends on sub
add     $14, $2 , $2          # 1st($2) &2nd( $2) depend on sub
sw      $15 , 1001 $2         # Base ($2) depends on sub
```

There will be type 1 data hazard in instr2 and type 2 in instr3.

Note: Data forward can mitigate hazard only if dependency is on one of the operand for two operands pipeline will have to be stalled.

Simulation results:



Control Hazard mitigation

Control hazard takes place when a branch is taken and the next two instructions are already in pipeline. In such a case, it is possible that the first instruction at the branch location might use the registers that are currently being used by instructions in the pipeline. Therefore it is necessary that the pipeline is stalled when a branch is taken so that register values do not get updated. The hazard detection unit serves this purpose by checking if the instruction at the new location uses the registers decoded by previous instruction. If there is a match, a hazard detected signal is made high which stalls all the pipelines before the memory stage.

```
36: sub    $1 , $4, $8
40: beq    $1 , $3 , 7
44: and    $12 , $2 , $5
48: or     $13 , $2 , $6
52: add    $14, $4, $2
56: slt    $15 , $6 , $7
72: lw     $14, 50($3)
```

In the code above, the instructions at location 44 and location 48 enter the pipeline but then if the branch is taken, these instructions must not write into registers or memory as the new location might need values stored previously in the registers.