Concept and Implementation:

At any point in the project, whenever we start the execution of **nth** cycle, we maintain all the instructions in the **(n-1)**th stage in a Hashmap data-structure called ***Stages***.

Data is stored in ***stages*** in a "Key" - "value" pair format as shown below:

Stages

Key	Value
F	SUB R2 R2 R4
D	SUB R1 R1 R3
E	ADD R5 R5 R3
E2	STORE R6 R5 0
B1	NOP
Dly	NOP
M	NOP
W	NOP

Additionally, we also maintain all the instructions in the (n-2)th stage in another Hashmap Data-structure called ***latches*** in "key" - "value" pair format similar to ***stages***.

latches

Key	Value
F	SUB R1 R1 R3
D	ADD R5 R5 R3
E1	STORE R6 R5 0
E2	NOP
B1	NOP
Dly	NOP
M	NOP
W	NOP

While processing for the nth cycle, instructions in stages is processed one by one by each stage. Finally, the data in stages represent the processed instructions of the nth cycle, which are then displayed on the console.

Fetch stage: implemented by the method *fetchStage().*

- Instruction in the decode stage is checked for flow dependency with the instruction in the Execute 1 stage.

- if any of the sources for this instruction are dependent on the destination of the instruction in the E stage then a flag ***isValidSource*** is set to false.

- if ***isValidSource*** is set to false:
  No new instruction is fetched in the pipeline.

- if ***isValidSource*** is set to True:

  A new instruction is fetched, ***currentFilePointer*** is incremented by 1, and ***currentPC*** is incremented by 4 (to point to the next instruction).

  instruction in the "F" of stages is moved to the "F" of latches and the newly fetched instruction is put in the "F" of stages

Decode stage : implemented by the method *decodeStage().*

- if ***isValidSource*** (set in the fetch stage) is false

  add NOP in the "D" of latches: this is done to ensure a NOP in the ALU1 stage of the *stages* when we swap instruction later in the *Execute1* stage

- if ***isValidSource*** = true

  Process the instruction in the fetch stage of the latches and store it in *latches*. (check if valid data is present in the source registers for that instruction)

  MoveInstruction("D", "F");

  Move instruction in the "D" stage of stages to to the "D" stage of latches,
  and move instruction from the "F" stage of latches to the "D" stage of stages.

__Execute 1 stage :__ implemented by the method ***execute1().***

- The "D" instruction in *latches* is read. This instruction represents the instruction in the decode stage of the previous cycle.

- if (instr is not a ***controlFlowInstruction*** i.e any of BNZ|BZ|JUMP|BAL|HALT)
  ```
  {
  If (src-Ex1! = 0) && (src-Ex1 == dest-Mem) && (RegWrite-Mem) then
  Forward operand from MEM stage.
  Elseif (src-Ex1!=0) && (src-Ex1 == dest-WB) && (RegWrite-WB) then
  Forward operand from WB stage
  Elseif (src-Ex1!=0) && (src-Ex1 == dest-Ex2) && (RegWrite-Ex) then
  Forward operand from Execute 2 stage.
  Else
  Use operand from the register file.
  }
  ```

- else
  ```
  {
  set branchFUflag = true.
  move "E" of stages to "E" of latches
  add NOP in "E" of stages.
  }
  ```

- if(branchFUflag = false)
  ```
  {
  Move "E" of stages to "E" of latches,
  and move "D" of latches to the "E" of stages.
  }
  ```

__Execute 2 stage :__ implemented by the method ***execute2Stage().***

- Instruction from the "E" of latches is processed by the ***executeInstruction2Method***().

  moveInstruction("E2", "E");
  move "E2" of stages to "E2" of latches and
  move "E" of latches to "E2" of stages.

- if there is no instruction in "E" of latches then Add NOP in "E2" of stages.

__Branch FU__: Implemented by the method ***branchStage().***

"**D" of latches**: represents the instruction in the decode stage of the previous cycle

control Flow Instruction: means any of BNZ|BZ|JUMP|BAL|HALT

step1: **move instructions**

```
if ("D" of latches is a control Flow Instruction)
 {
         move "D" of latches to the branch stage of the current cycle
  }
else
 {
         add a NOP to the branch stage of the current cycle.
 }
```

step2: process branch instructions

```
if ("D" of latches is a control Flow Instruction)
 {
  • if instruction is BAL: special register is set to (current PC - 1)
  • if instruction is BNZ: result of evaluation of instruction in the E2 stage of the        current cycle is
    stored in a temporary varaible dest .

  • process branch instructions and calculate updated PC
  • if(currentPC is not equal to UpdatedPC)
      {
      Update currentPC and current file pointer values.
      set flag flushRegisterValues equal to true.
      }

  • if(flushRegisterValues = true)
     {
          Add NOPs in the "F" and "D" of stages.
     }

 }
```

Delay Stage: Implemented by the method ***delayStage().***

```
• if(latches contains an instruction at "B1")
    {
     move "Dly" of stages to "Dly" of latches and
     move "B1" of latches to "Dly" of stages.
     i.e. move instruction from Branch stage of the previous cycle to the Delay stage of current cycle
     }
```

- else

  {

    Add NOP in the Delay stage of current cycle.

  }

<u>MEM Stage</u>: Implemented by the method ***memoryStage().***

- Move instruction from Ex-2 or Delay stage of the previous cycle to the Mem stage of the current stage.
- Perform Memory Operations such as Load, Store.

<u>WriteBack Stage:</u> implemented by the ***method writebackStage().***

- Move instruction from the Mem stage of the previous cycle to the W/B stage of the current cycle
  ```
  moveInstruction("W", "M");
  ```

- Perform Register File Operations.
  Write values in register File

- **if** operation is **HALT**
  stop execution.