Divya Boggavarapu(1002086719)

Computer Architecture Assignment-2

1. **Complete implementation of the "Store" instruction, show a small test program (example) to show that it works correctly.**
2. **Write a small (CAT) program to add two 1 x N vectors (that is add the corresponding values in two lists of numbers), for example:**

   [1, 2, 3] + [4, 5, 6] -> [5, 7, 9] Where N will be less than 20
   Instrument the CAT VM to count number of instructions executed, clock cycles, as well as number of memory references

Ans:

Please refer execute.py

```
DivyaCompArch/execute.py
[0, 9, 17, 20, 0, 23, 0, 0]
[76546062, 77070353, 60817421, 78118932, 59572224, 9273344, 27262976, 34603008, 35127296, 36175872,
68059136, 102760452, 134742032, 3, 1, 2, 3, 4, 5, 6, 0, 5, 7, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
clock= 146 IC= 29 Coderefs= 0 Datarefs= 0 Start Time= 1669670919.009293 Currently= 1669670919.00981
```

```
;  add two 1 x N vectors
          go   0
0         ldi  2 .v1          ; r2 points to vectors v1
          ldi  3 .v2          ; r3 points to vectors v2
          ld   4 .count       ; r4 has value of counter
          ldi  5 .output
.loop     ld   1 *2           ; load vectors v1 to register 1
          add  1 *3           ; add vectors v2 to register 1
          dec  4              ; decrement count
          inc  2              ; increment vectors v1 pointer to next
elements
          inc  3              ; increment vectors v2 pointer to next
elements
          inc  5              ; increment output pointer
          st   1 *5           ; store value at register 1 to .output
          bnz  4 .loop
          sys  1 16
.count    dw   3
.v1       dw   1
          dw   2
          dw   3
.v2       dw   4
          dw   5
          dw   6
.output   dw   0
          dw   0
          dw   0
26        dw   0
          end
```

3. Implement (CAT) data hazard detection, as if the CAT were pipelined with a 5 stage Pipeline, That is show that you can detect RAW data hazards (for this assignment, for the CAT implementation: Implement a scoreboard (or similar) to detect data hazards.

Show where these hazards exist (in the source asm/binary code), and the scoreboard results, Why these hazards exist, how long it is necessary to stall. All of the preceding should, of course, be implemented in code, show demo code to test implementation,

and results printed.

Ans:

Data Hazards Occurs when instruction executing in one pipeline depends on the output of instruction in the other pipeline.

When data is not available when needed in the pipeline or when data has not been written but needs to be read or similar.

Three types of Data Hazards are:

1. **read after write (RAW)**
   Ex: i1 R2<- R5+R3
   i2 R4<- R3+R2
   When R4 depends on R2 value and if R2 from i, not saved before execution R2.
2. **write after read (WAR)**
   Ex: i1 R4<-R1+R5
   i2 R5<- R1+R2
   If i2 executes before i1 the value of R5 changes to execute R4.
3. **write after write (WAW)**
   Ex: i1 R2<-R4+R7
   i2 R2<- R1+R3
   The Writeback of i2 must be delayed untill i1 finishes executing.

However, there is only possibility of RAW to occur in the CAT 5 stage Pipeline.

4. Implement control (branch) hazard detection on CAT (similar to the preceding). Show where these hazards exist (in the source asm/binary code), and the scoreboard results, Why these hazards exist, how long it is necessary to stall (Similarly, please implement in code, show demo code to test your implementation, and results printed.)

Ans:

When the pipeline chooses an inaccurate branch prediction, instructions that must be discarded enter the pipeline. This is known as a control hazard. A branch hazard is another name for a control hazard.
These occur particularly in pipelined architecture and can be mitigated through various ways.
1. Stall
2. Prediction
3. Dynamic Branch Prediction
4. Reordering Instructions

5. Implement one bit branch prediction for CAT (similar to preceding.)

Ans:
In the 5-stage pipeline, a branch completes in two cycles If the branch went the wrong way, one incorrect instr is fetched One stall cycle per incorrect branch.
When we execute a branch, first check if it was taken when it was last executed: if yes, predict it will be taken this time; if no, predict it will not be taken this time.

6. Describe how a VLIW implementation with two slots VLIW instructions, could be:

(a) implemented

(b) show a demo test program

(c) You DO NOT need to implement this, just describe

Ans:
A VLIW with two slots would have two operation fields, each field specifying the operation to be executed by the primitive instructions. Such instructions are at least 64 bits in width to accommodate the operation fields.
VLIW encoding has two main objectives. The main objective is to represent operation independence, which means that hardware does not have to check for data (and

structural) dangers within an instruction word. Reduced complexity of instruction routing is a secondary objective. The simplest routing of operations, a fixed mapping, is provided by the classic VLIW design, which supports a single instruction word size (as well as simple instruction fetch).