Pipelined Functional Simulator for Subset of RISC-V instruction set

---

This document describes the design aspect of our pipelined functional simulator for a subset of the RISC-V instruction set.

RUNNING THE CODE:

-Generate Machine Code: g++ PHASE1.cpp (Input File: asmbl.asm) (mcode.mc created)

-Running Simulation : g++ PHASE3.cpp

INPUT & OUTPUT:

A machine code is sent as input to the simulator and its pipelined execution is carried out using the techniques of bit prediction, stalling, flushing and data forwarding. These techniques aim to optimize the program by minimizing the execution time (or the number of cycles) and running the maximum number of instructions possible in the same cycle.

The program outputs the details of all the steps executed per cycle along with the various control changes like jumps, predictions, flushes and stalls.

The following stats are printed at the end of execution:

For both Pipelined & Non-Pipelined Execution:

Stat1: Total number of cycles
Stat2: Total instructions executed
Stat3: CPI (Cycles per Instruction)
Stat4: Number of Data-transfer (load and store) instructions executed
Stat5: Number of ALU instructions executed

Only for Pipelined execution:

Stat6: Number of Control instructions executed
Stat7: Number of stalls/bubbles in the pipeline
Stat8: Number of data hazards
Stat9: Number of control hazards
Stat10: Number of branch mispredictions
Stat11: Number of stalls due to data hazards
Stat12: Number of stalls due to control hazards

The register and memory data is written to files data_r.mc and data_mem.mc respectively. The stats are written to a stats.txt file.

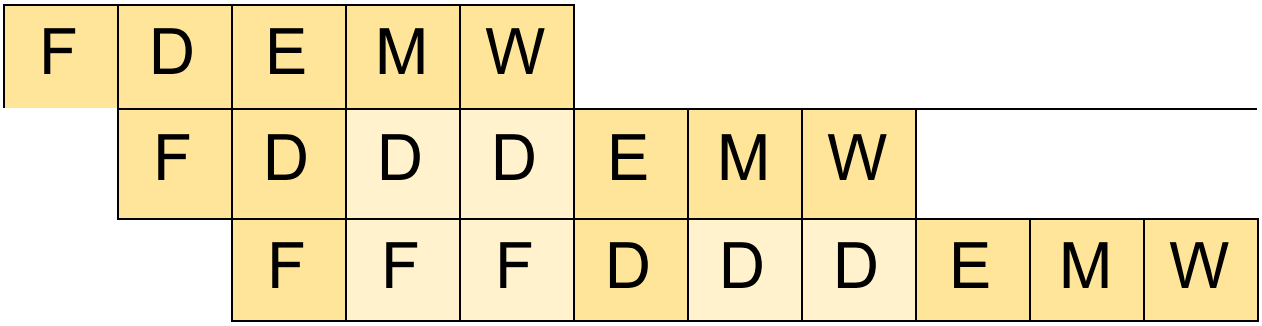
FUNCTIONALITIES & FEATURES:

- Pipelining : Functionality to switch between Pipelined and Non Pipelined execution.
- Data Forwarding: Knob to enable /disable data forwarding.
- Pipeline Registers: Functionality to output register values and values of the inter state buffers
    - At the end of each cycle
    - At the end of a specified cycle number
- Register File: Functionality to print register file at the end of each cycle
- Bit prediction: Knob to enable/disable bit prediction.

LOGIC AND DESIGN FLOW

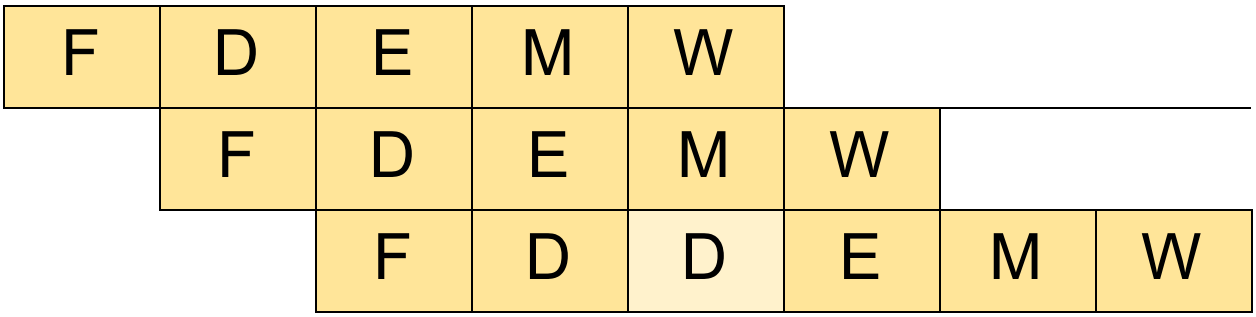
I1: add x1 x0 x0
I2: sub x2 x1 x0
I3: and x3 x2 x0

- There are two data dependencies in the above example ,i.e. between I1 , I2 and I2 , I3.So, in this case there are two possibilities to overcome these dependencies.
- **Stalling :** I2 gets stalled in the Decode stage till x1 gets updated in 5th cycle. I3 gets stalled for 2 cycles in the Fetch stage and for 2 cycles in Decode stage.
- **Data Forwarding :**Forwarding path is added from Execute of I1 to Execute of I2 thereby preventing the loss of 2 cycles . Further one more Forwarding path from Execute of I2 to Execute of I3 is added.



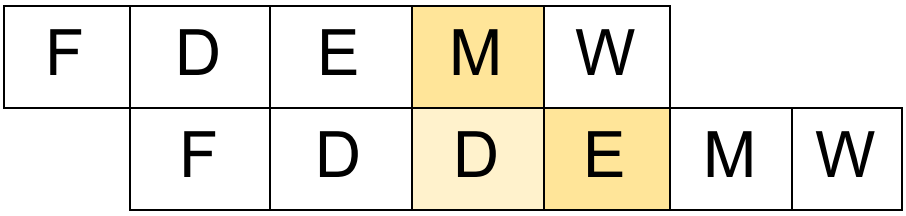
I1: addi x1 x0 1
I2: addi x2 x0 2
I3: add x3 x1 x1

- There is data dependency between I1 and I3 . Here also we can overcome the dependency by stalling and data forwarding.
- **Stalling:**
  Since dependency is between I1 and I3 instruction ,no stalling will be for I2. Only I3 gets stalled in the Decode stage for one cycle .
- **Data Forwarding**:
  A Forwarding path between Memory stage of I1 and Execute stage of I3 is added to prevent the loss of cycle due to stalling.



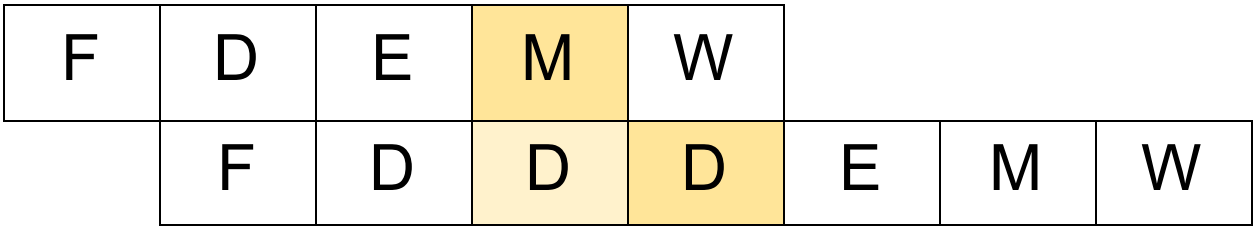
I1: lw x1 0(x2)
I2: and x3 x1 x0

- Since the updated value of x1 will be available only after the Memory stage of I1, therefore I2 gets stalled for one cycle in the Decode stage and then a forwarding path is added between Memory of I1 and Execute of I2.



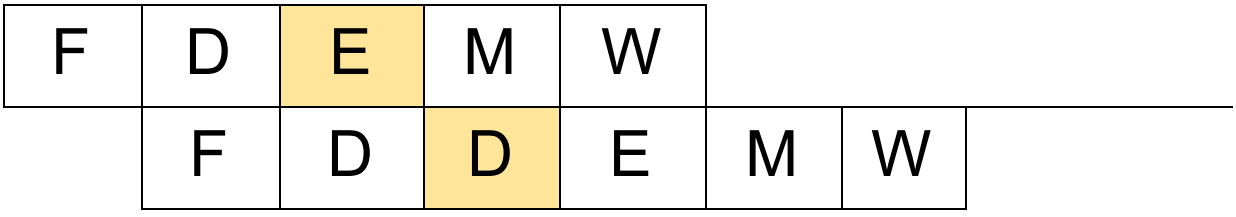
I1: lw x1 0(x2)
I2: beq x1 x0 exit

- In this case , I2 gets stalled for two cycles in the Decode stage and a forwarding path is added from Memory of I1 to Decode of I2 ,so that we can get the updated value of x1 in I2.

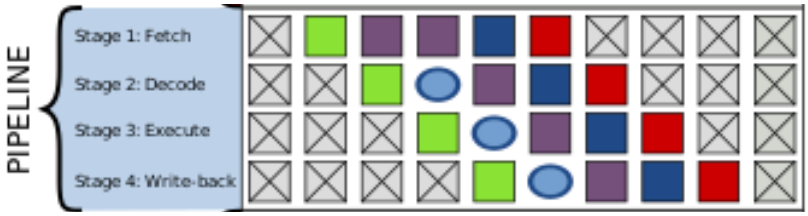


I1: and x2 x1 x0
I2: bge x2 x0 loop

- Unlike the load instruction ,here the updated value of x2 is available in Execute of I1 , therefore I2 gets stalled only for one cycle in Decode stage and a forwarding path is added between Execute of I1 to decode of I2.



- **Flushing-**
  - **Bit prediction turned off**- The cpc is incremented by 4 in every iteration, to fetch the following instruction in the next clock cycle.In the decode phase, if it's found out that the instruction was any branch instruction, (sb type, jal or jalr), we check if the following fetch is of the required instruction. In this case, for jal and jalr, we always end up choosing the incorrect instruction(if the offset is not 4). In case of sb type, if branch is not taken, the pc update would be correct. If an incorrect pc was chosen, flushing is required. Before the next fetch, the pc is updated accordingly, resulting in a bubble in the pipeline, such that in the next cycle, no decode stage will be run, instead the correct instruction will be fetched. This way, we have 1 bubble per flush in the pipeline.



  - **Bit prediction turned on**- Now, if we have bit prediction turned on using a provided knob, we construct a BHT table for the branch instructions.

BHT STRUCTURE

PC	PREDICT BIT	TARGET ADDRESS
44	0	60
58	1	20

    - To achieve this, we have made a map in the program, queried by the PC.In the fetch stage, the pc is looked up in the BHT. If present, cpc is updated according to the target address, else by 4. In the decode stage, if instruction is sb type, jal or jalr, we again compare predicted pc and the correct next pc , checking if the correct prediction was made. If not, the corresponding entry is made in the BHT table and the CPC is updated before the next fetch, resulting in 1 bubble in the pipeline. If the prediction was correct, it does not require flushing and the control flow continues.
    - Key: cpc is the pc of the next instruction to fetch.

Test plan

We have successfully tested the simulator with following assembly programs:

- Fibonacci Program
- Bubble Sort
- Factorial Program
- Palindrome

Team Members:

GURSEERAT KAUR	2018CSB1093
NIKITA AGARWAL	2018CSB1109
PRANJALI BAJPAI	2018EEB1243
GOVIND BANSAL	2018EEB 1147
DAVID DAHIYA	2018EEB1143

Work Split:

CONCEPT BUILDING AND BASIC PIPELINING	DAVID
BIT PREDICTION	NIKITA & GOVIND
FLUSHING	NIKITA & GOVIND
STALLING	GURSEERAT & PRANJALI
DATA FORWARDING	GURSEERAT & PRANJALI