Report Project 1
Milestone 3

Prepared by
Fatemah Abdelwahed and Layla Mohsen

SUMMARY

- Project Objectives
- Design of the project
  - Modules
  - Datapath
- Difficulties and their solutions
- Testing
  - Procedure
  - Test Programs
  - Screenshots of Simulations

Project Objectives

The objectives we had for Project 1 Milestone 3 are:

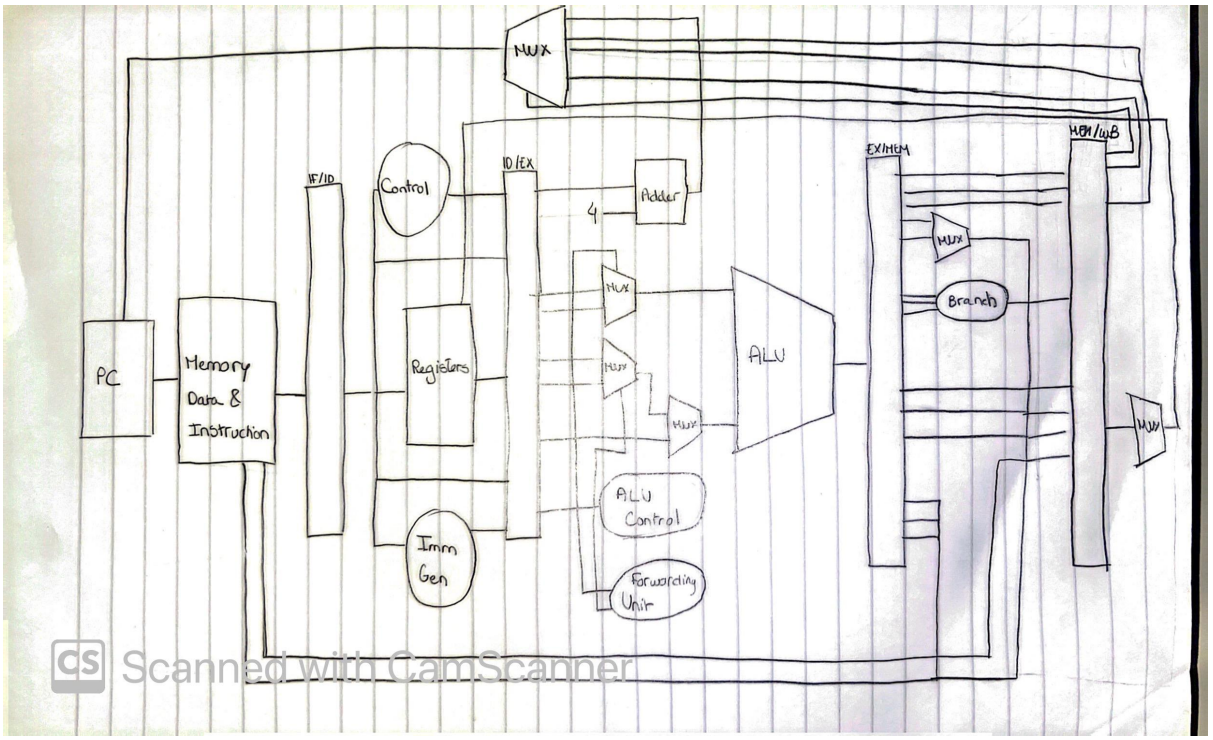
1. Ensure the processor correctly supports all 40 RISC-V instructions.
2. Let the processor be pipelined with correct hazards handling.
3. Use a single memory for both data and instructions.
4. Test, using simulations, the RISC-V instructions and make sure all the outputs are correct.
5. Have an organized datapath for easy understanding.
6. Do the Test Program Generator bonus.

Design of the project

Modules

- *Datapath:* contains the full datapath, calling all the modules
- *TopSSD:* used for the FPGA to show output of the the Datapath
    - *SevenSegmentDisplay:* used to show the seven segment display
        - *SevenSegmentOptimizer:* divides the thousands, hundreds, tens and ones digit
- *Nbit32Reg:* used for the PC, updates on clock cycle
    - *DFlipFlop:* send input to output on clock cycle
- *NbitALU:* performs operations, such as add, sub, and, or
    - *Shifter:* special module to perform sll, sra and srl instructions
- *InstMem:* instruction memory
- *ImmGen:* immediate generator
- *ControlUnit:* control unit that sends the different operations to the ALUControlUnit, DataMemory, outputs MemoryRead, MemoryWrite, …
- *ALUControlUnit:* defines the operation to be done in the ALU of each instruction
- *DataMem*: data memory, with inputs such as of the address of memory and write data and MemWrite, MemRead flags, and outputs the memory value in the specified address
- *ForwardingUnit:* forward unit that checks forwarding hazard at execute and memory stage and outputs signal.
- *Branch:* module used for branch instructions, that output 1 to branch, and 0 to not branch, based on the output of ALU flags
- *NBitReg:* register file with inputs such as the addresses of the register, WritetoRegister flag, the input data to write to register and outputs the values of the registers
- *NbitShiftLeft:* shifts number, to multiply by 4 for the PC (not used)
- *NbitMUX:*
    - *MUX:* multiplexer (2x1 and 4x1)
- *NbitRCA:* Ripple Carry Adder
    - *FullAdder:* full adder
- *Defines:* file that contains opcodes, ALUOperations…

Datapath


MUX
IF/ID
Control
ID/EX
Adder
4
EX/MEM
MEM/WB
MUX
Branch
PC
Memory
Data &
Instruction
Registers
MUX
MUX
MUX
ALU
ALU
Control
Imm
Gen
Forwarding
Unit
MUX
Scanned with CamScanner

Difficulties and their solutions

- **Difficulty 1:** coordinating to not overwrite each others code
  Solution 1: we used an Excel file to update each other on what instructions we each did, and when we each finish updating the code, we uploaded it on Google Drive for the other partner to use

- **Difficulty 2:** many different MUX and Adders, that become confusing to name and to keep track of (also many different changes in the clocks)
  Solution 2: we organized the code separating each part, and commenting what each mux and adder does.

- **Difficulty 3:** combining the data memory and instruction memory into one single memory
  Solution 3: We made the necessary adjustments to support both memories in one. We did this by dividing it into two divisions, a larger one for the instructions to support large programs. We adjusted the offset necessary to reach the data memory in the correct location. We made changes to the clock cycles (negedge, posedge) in the memory and in the main module (according to the Dr's explanation of the single memory for project support).

- **Difficulty 4:** each time we fix an error, 5 other errors come up
  Solution 4: we fixed the errors

- **Difficulty 5:** connecting the pipeline registers, especially with the many different signals/data and modules that support all 40 instructions
  Solution 5: organizing where every input and corresponding output should be in each register stage, especially the concatenation between the signals that are put in the control part of each stage.

- **Difficulty 6:** implementing forwarding, hazard detection and flushing
  Solution 6: organizing, testing and debugging where all the incorrect values were until we see where errors occur and fixing the corresponding connections (one of the hardest issues to spot was a wrong control signal passed to one of the forwarding mux's)

Testing

Procedure

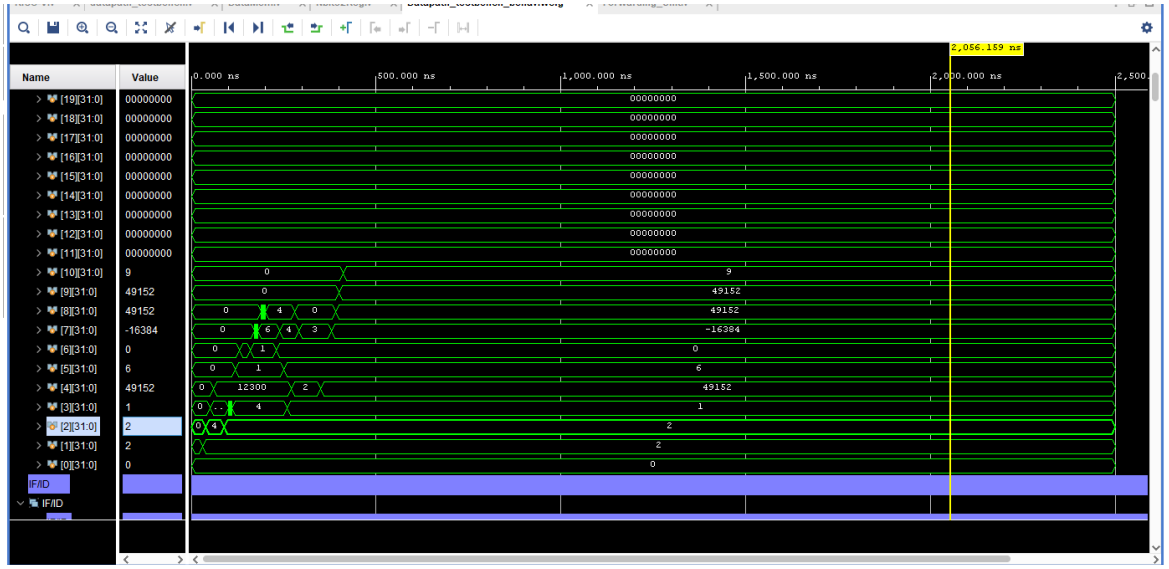
We created a program that uses the 40 instructions, with different dependencies and imports it in the Instruction Memory.

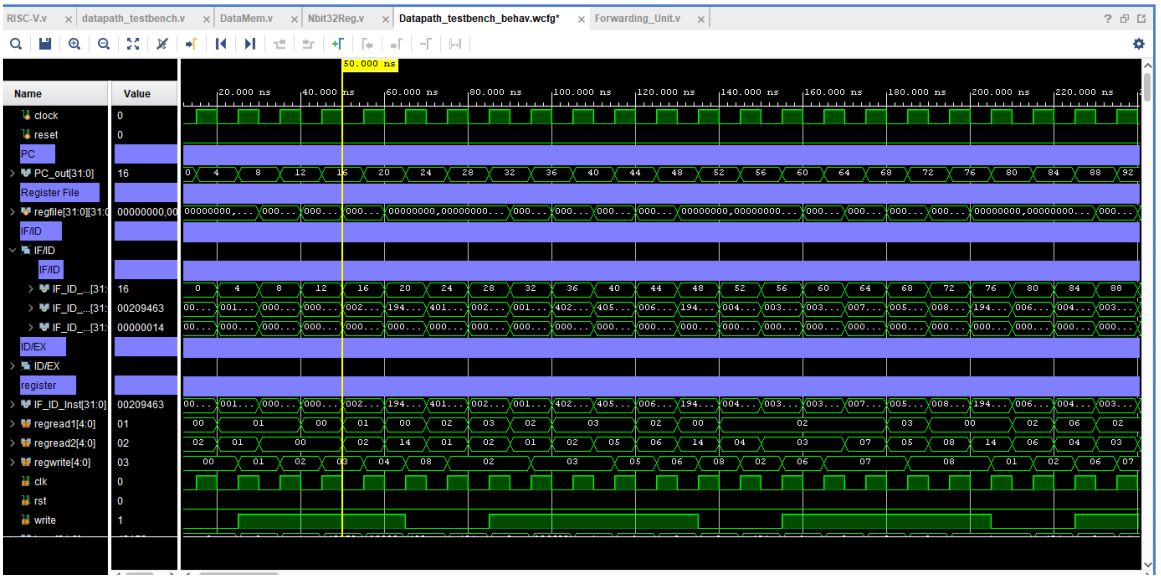
Test Programs & Screenshots of Simulations

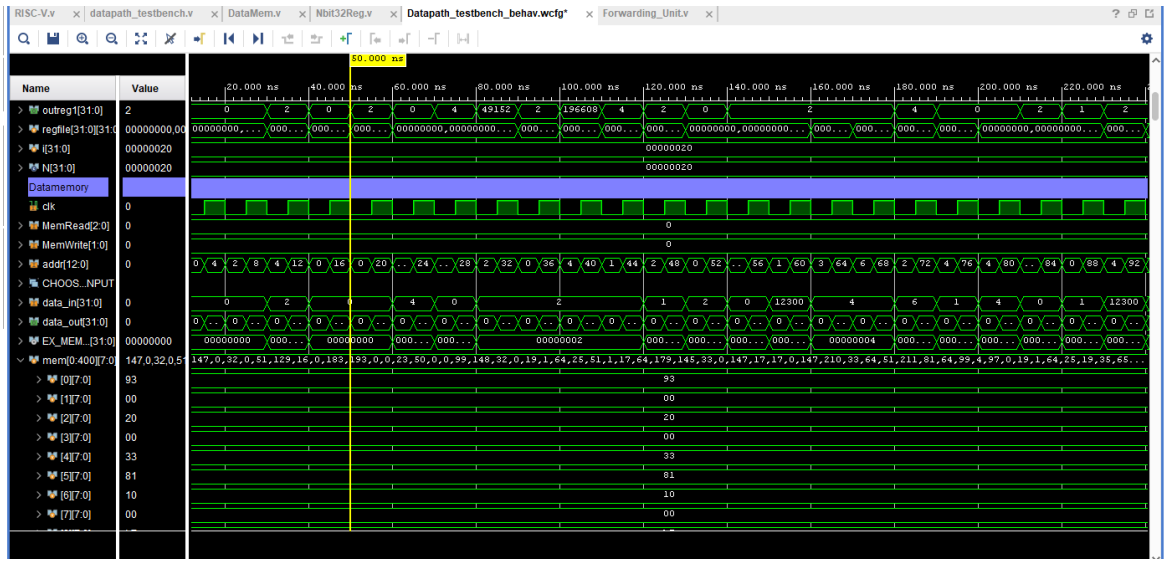
program 1 (36 instructions)

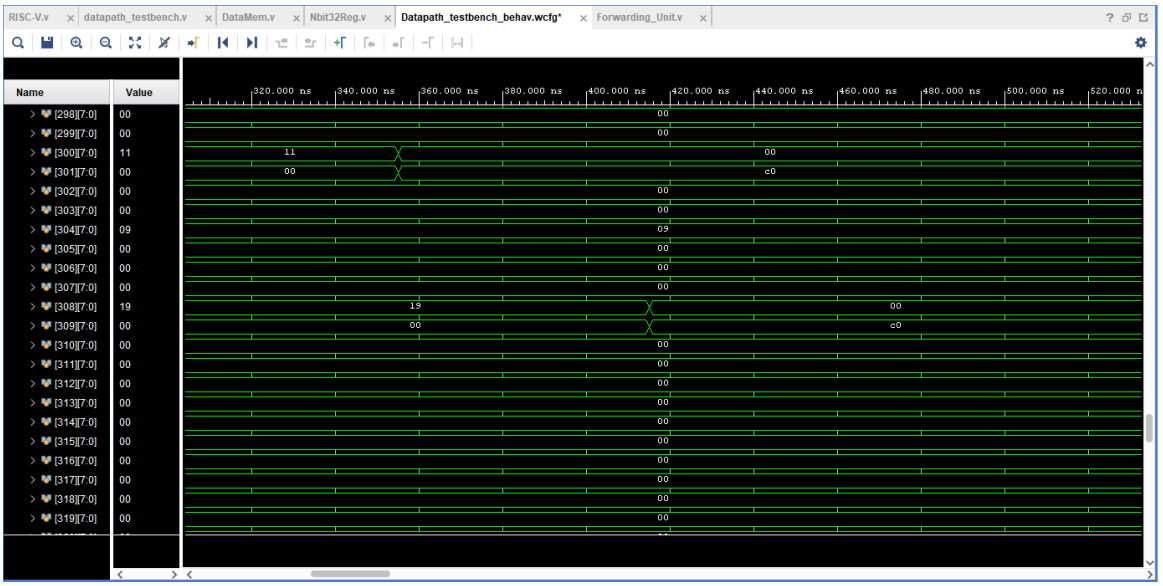
```
addi x1, x0, 2          #x1 --> 2
add x2, x1, x1          #x2 --> 4
lui x3, 12              #x3 --> 49152
auipc x4, 3             #x4 --> 12300
bne x1, x2, l1          #shud branch
addi x2, x0, 404
l1: sub x2, x2, x1      # x2 --> 2
sll x3, x3, x2          # x3 --> 196608
slli x3, x2, 1          # x3 --> 4
srai x5, x3, 2          # x5 --> 1
sra x6, x3, x5          # x6 --> 2
beq x2, x6, l2          # shud branch
addi x2, x0, 404
l2: slti x6, x2, 4      # x6 --> 1
ori x7, x2, 3           # x7 --> 3
or x7, x2, x3           # x7 --> 6
and x8, x2, x7          # x8 --> 2
andi x8, x3, 5          # x8 --> 4
jal l3
addi x2, x0, 404
l3: slt x6, x2, x6      # x6 --> 0
xori x7, x6, 4          # x7 --> 4
xor x5, x2, x3          # x5 --> 6
sltiu x3, x2, 4         # x3 --> 1
srli x4, x7, 1          # x4 --> 2
sltu x8, x7, x4         # x8 --> 0
srl x7, x5, x3          # x7 --> 3
blt x4, x8, l1          # shud not branch
bge x8, x4, l1          # shud not branch
bltu x4, x8, l1         # shud not branch
bgeu x4, x8, l4         # shud branch
addi x2, x0, 404
```

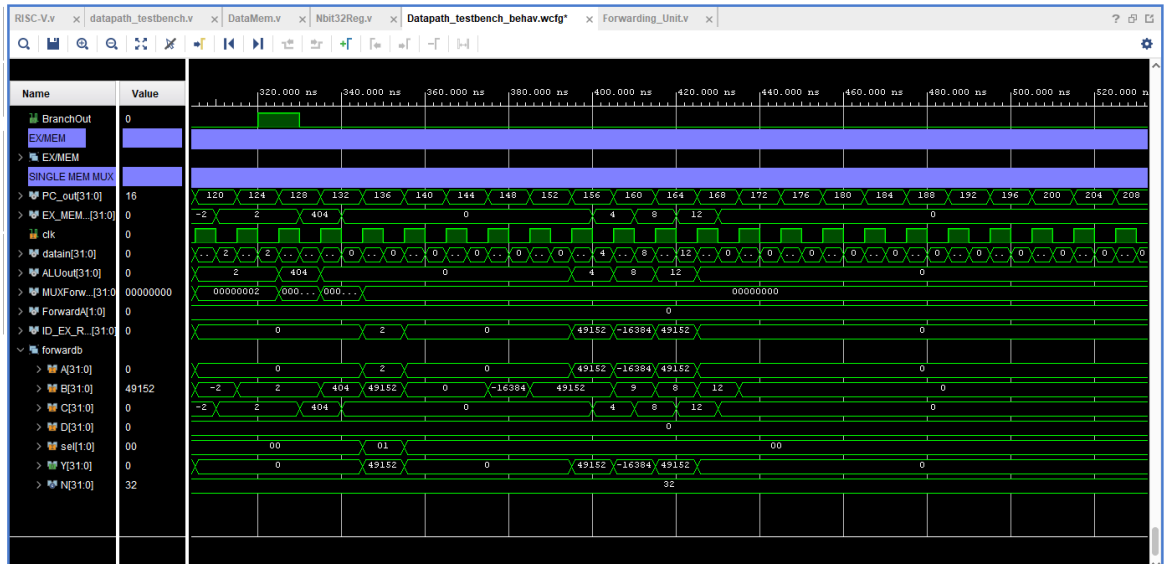
l4: lui x4, 12 #x4 --> 49152
sw x4, 0(x0) #0(x0) --> 49152
lb x6, 0(x0) #x6 --> 0
lh x7, 0 (x0) #x7 --> -16384
lw x8, 0(x0) #x8 --> 49152
lhu x9, 0(x0) #x9 --> 49152
lbu x10, 4(x0) #x10 --> 51
sh x7, 8(x0)
sb x8, 12(x0)






RISC-V.v
datapath_testbench.v
DataMem.v
Nbit32Reg.v
Datapath_testbench_behav.wcfg*
Forwarding_Unit.v
Name
Value
outreg1[31:0]
regfile[31:0][31:0]
i[31:0]
N[31:0]
Datamemory
clk
MemRead[2:0]
MemWrite[1:0]
addr[12:0]
CHOOS...NPUT
data_in[31:0]
data_out[31:0]
EX_MEM...[31:0]
mem[0:400][7:0]
[0][7:0]
[1][7:0]
[2][7:0]
[3][7:0]
[4][7:0]
[5][7:0]
[6][7:0]
[7][7:0]
50.000 ns


RISC-V.v
datapath_testbench.v
DataMem.v
Nbit32Reg.v
Datapath_testbench_behav.wcfg*
Forwarding_Unit.v
Name
Value
[298][7:0]
[299][7:0]
[300][7:0]
[301][7:0]
[302][7:0]
[303][7:0]
[304][7:0]
[305][7:0]
[306][7:0]
[307][7:0]
[308][7:0]
[309][7:0]
[310][7:0]
[311][7:0]
[312][7:0]
[313][7:0]
[314][7:0]
[315][7:0]
[316][7:0]
[317][7:0]
[318][7:0]
[319][7:0]


RISC-V.v
datapath_testbench.v
DataMem.v
Nbit32Reg.v
Datapath_testbench_behav.wcfg*
Forwarding_Unit.v
Name
Value
BranchOut
EX/MEM
EX/MEM
SINGLE MEM MUX
PC_out[31:0]
EX_MEM...[31:0]
clk
datain[31:0]
ALUout[31:0]
MUXForw...[31:0]
ForwardA[1:0]
ID_EX_R...[31:0]
forwardb
A[31:0]
B[31:0]
C[31:0]
D[31:0]
sel[1:0]
Y[31:0]
N[31:0]

Program 2 (remaining 4 instructions)

addi x2, x0, 2
fence
addi x3, x0, 40
ecall
addi x4, x0, 50
ebreak
addi x2, x0, 404
addi x3, x0, 404
addi x4, x0, 404