Week 8 Report
Architecture Summary
The WashU-2 is a simplistic cpu, introduced by Prof. Jon Turner (Washington University in St. Louis) as a teaching architecture.
It uses 16-bit registers and signals (wires)

The registers are:
Accumulator (ACC): holds the working data
Program Counter (PC): holds the memory address of the next instruction
Instruction Register (IR): using PC, instruction is loaded in this. (Instructions are 16 bit in this cpu)
Memory Address Register (MAR): stores the address in memory which we want to access at any time
Memory buffer register (MBR): contains data just read from memory, or data which is about to be written to memory

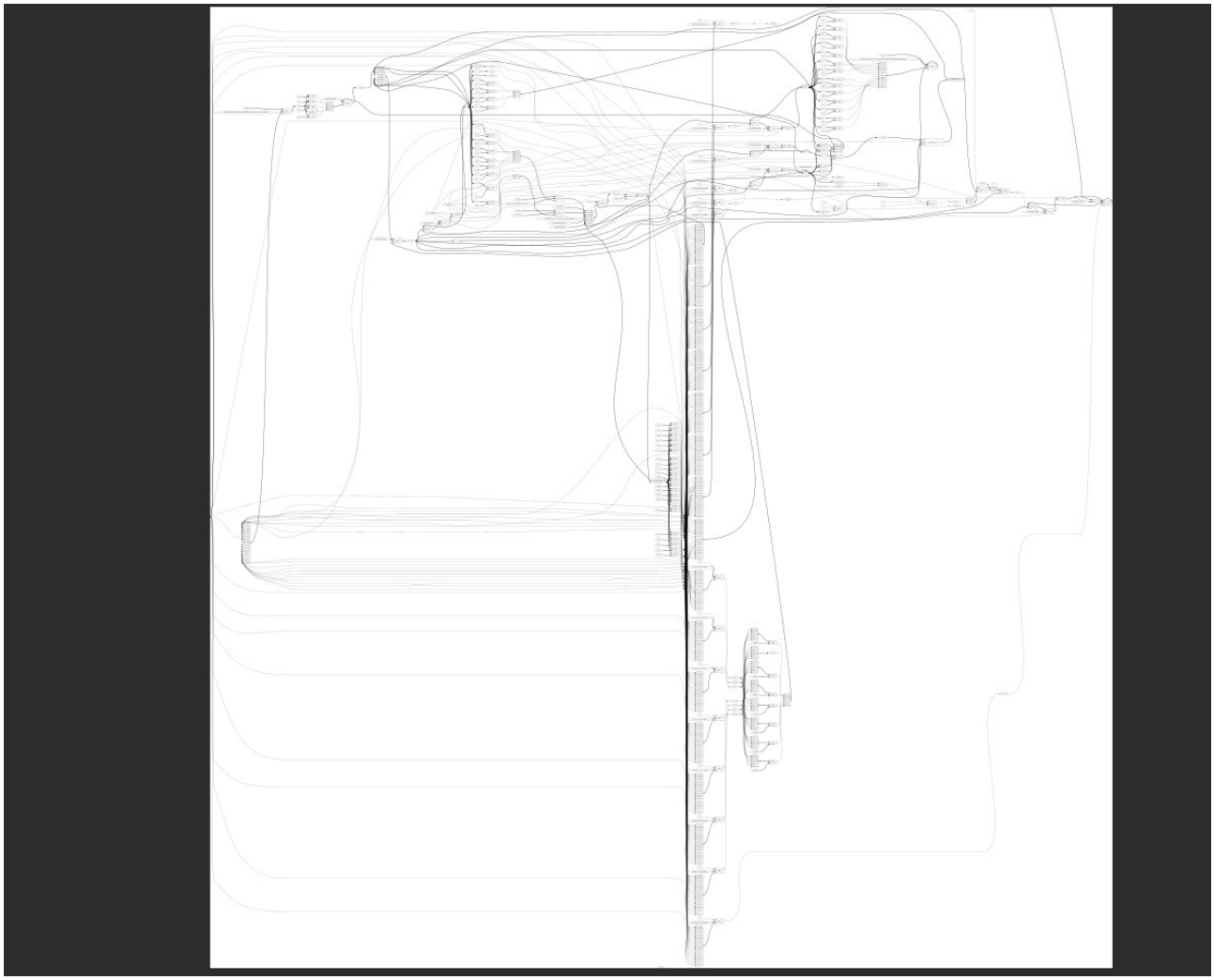
We also have 3 flags: zero_flag, pos_flag and neg_flag, which are assigned a value based on the sign of the quantity in ACC

Format of 16-bit instructions:
first 4 bits → the operation code; next 12 bits → the parameter for the operation
if the operator is binary, the parameter is the one not presently in ACC

To execute the instructions, we use an FSM (Finite State Machine) with 22 states:
3 for fetching instruction from memory
2 for direct access of memory
2 for indirect access of memory (for instructions like pointers in cpp)
and others specific to specific operations

The various registers and the ALU are connected by a multiplexed bus.

RTL netlist



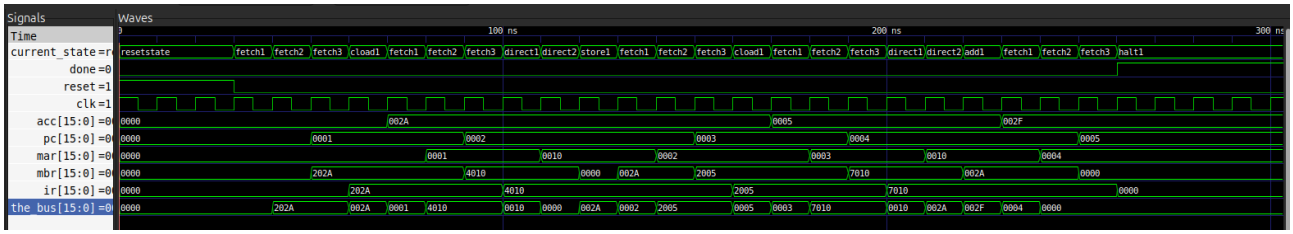
Since things are not very visible, please find week8.yosys_show.png in the cpu directory

Waveform Evidence

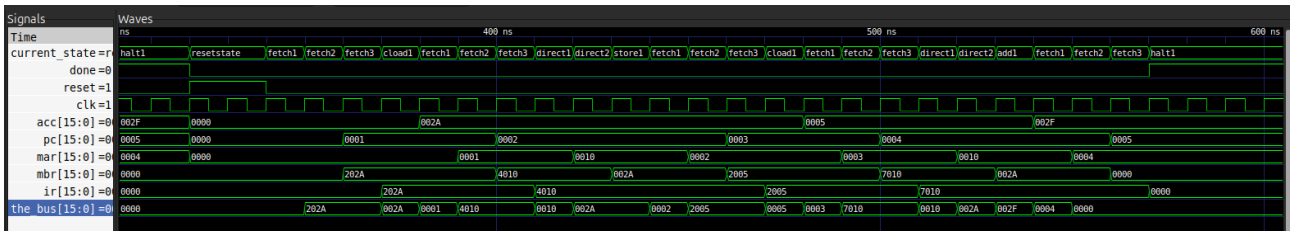
Direct Arithmetic

ADD

I

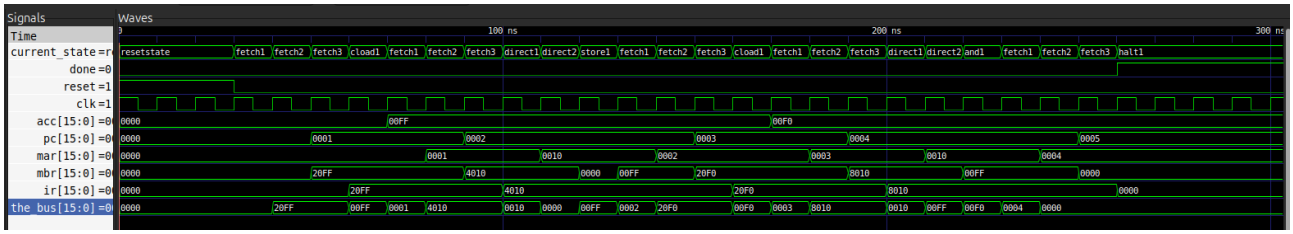



II

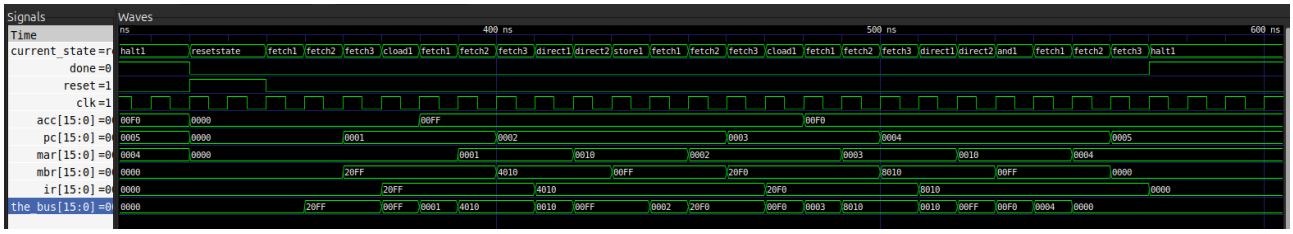



AND

I



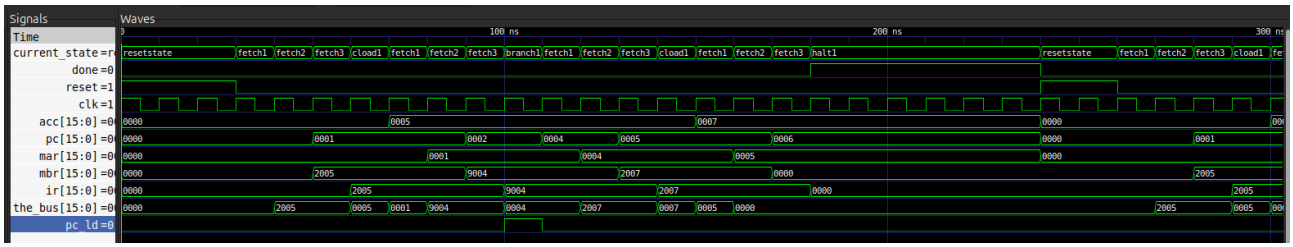

II



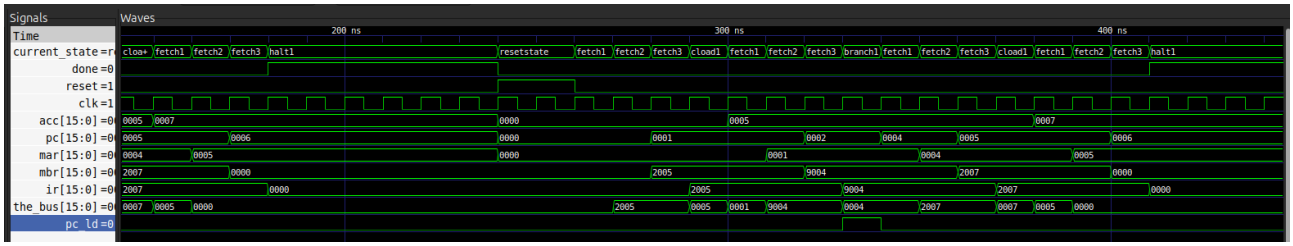
Conditional branch

BRANCH

I

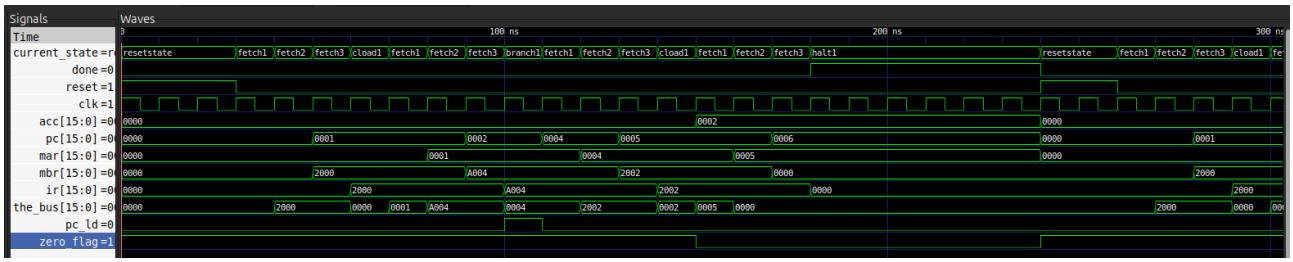



II

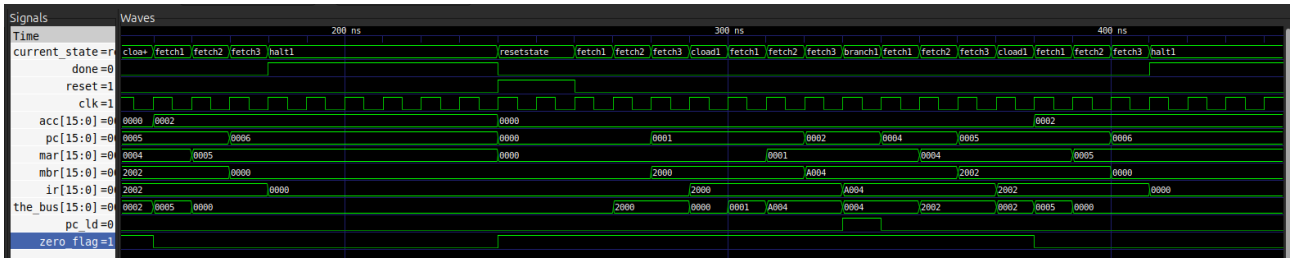



BRZERO: taken

I

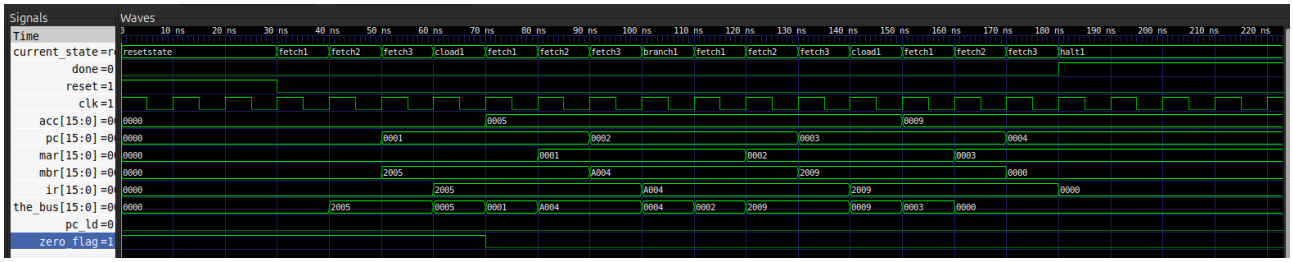



II

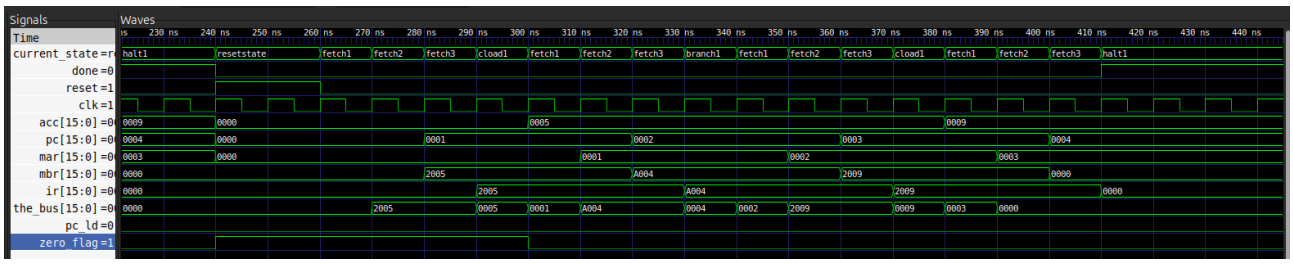



BRZERO: not taken

I

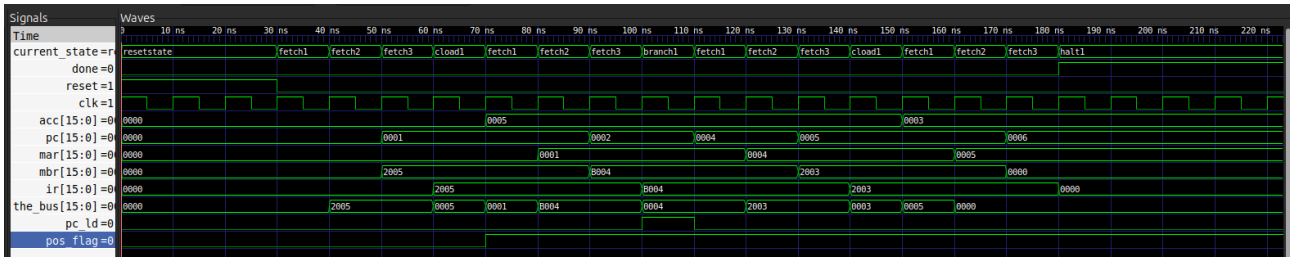



II

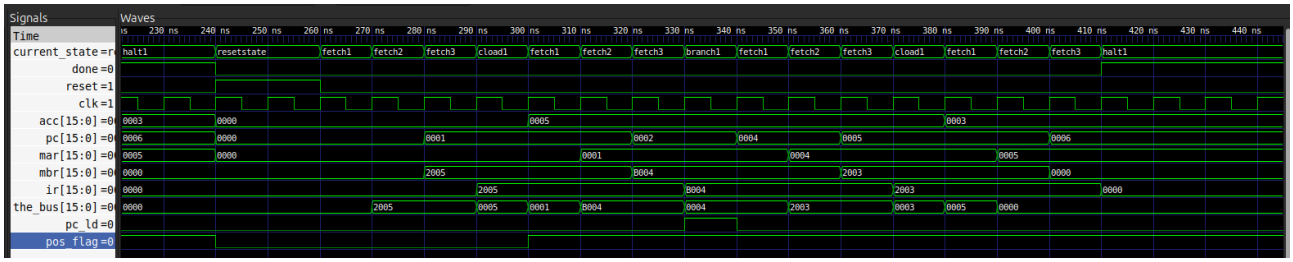



BRPOS (taken)

I

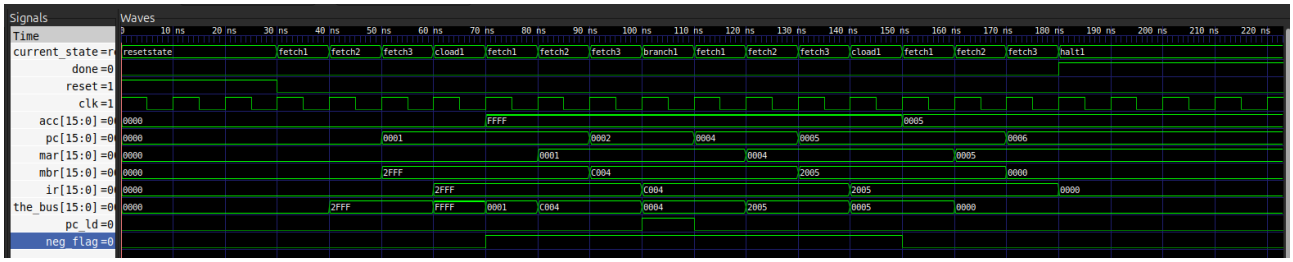



II

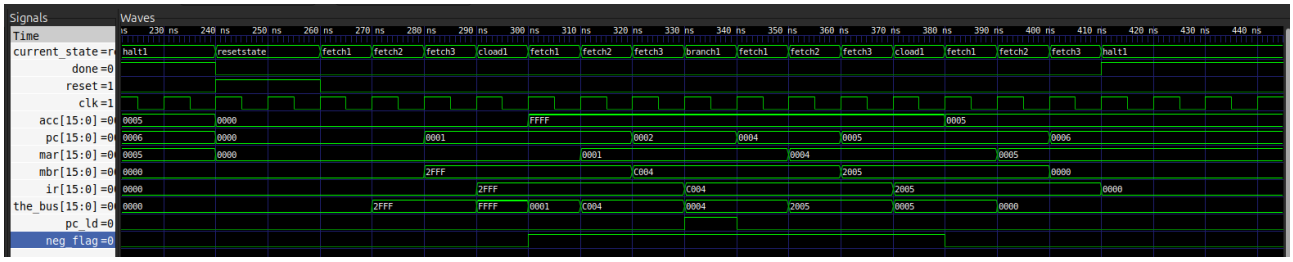



BRNEG

I



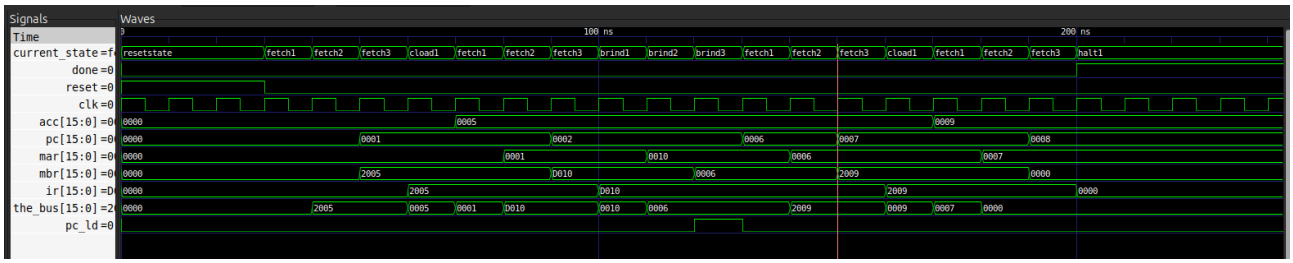

II



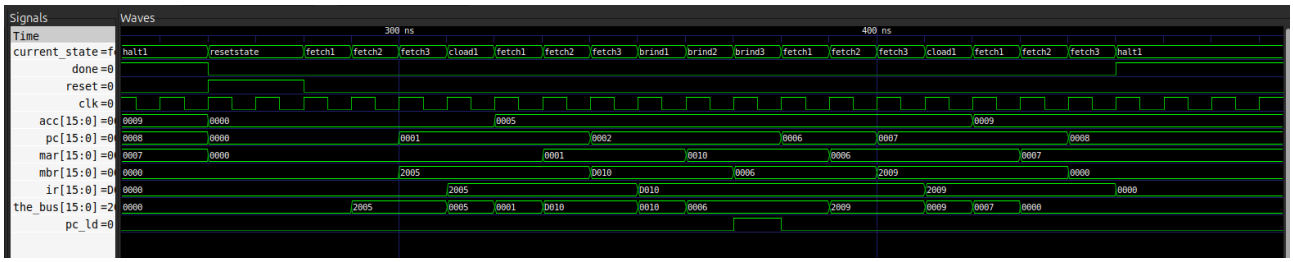
Indirect branch

BRIND

I



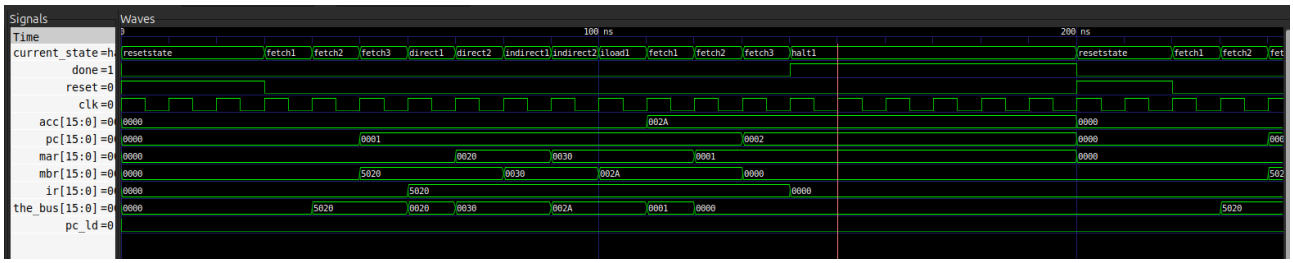

II



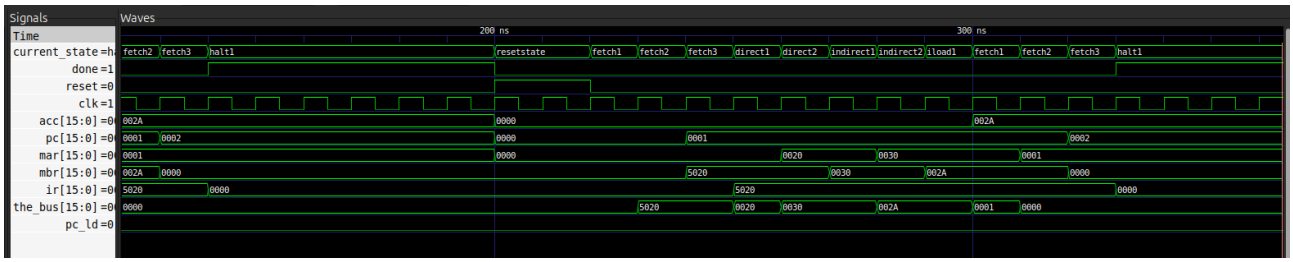
Double-indirect

ILOAD

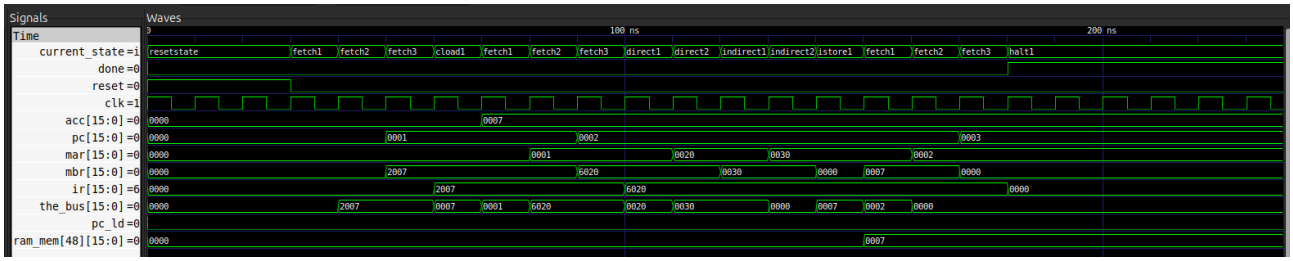
I




II

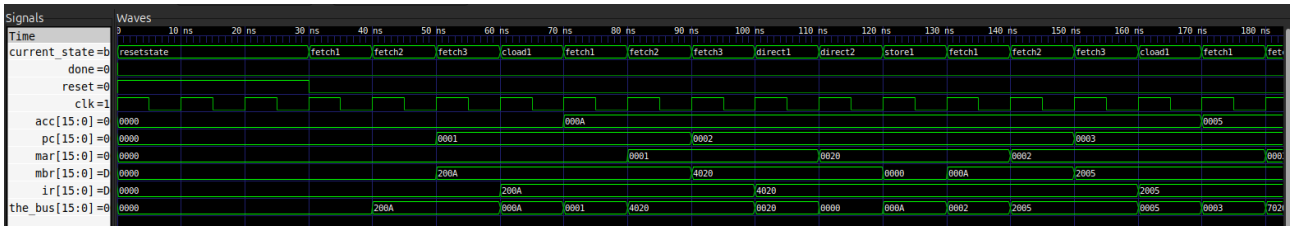



ISTORE



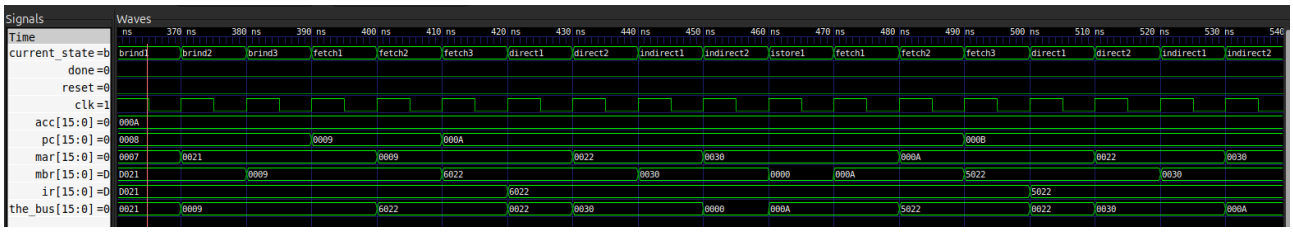
# Integration

## I

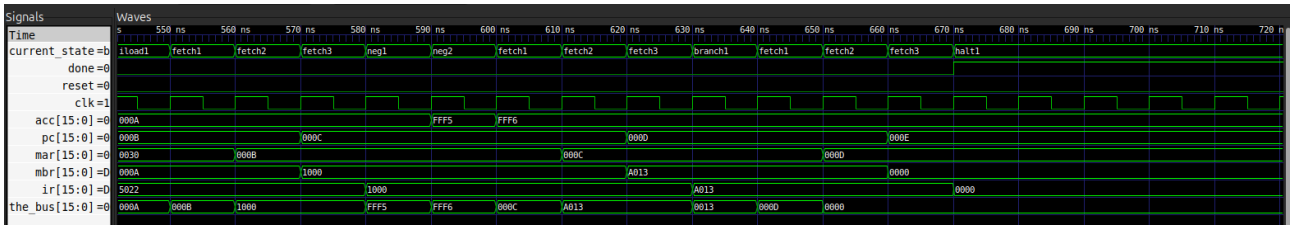



## II

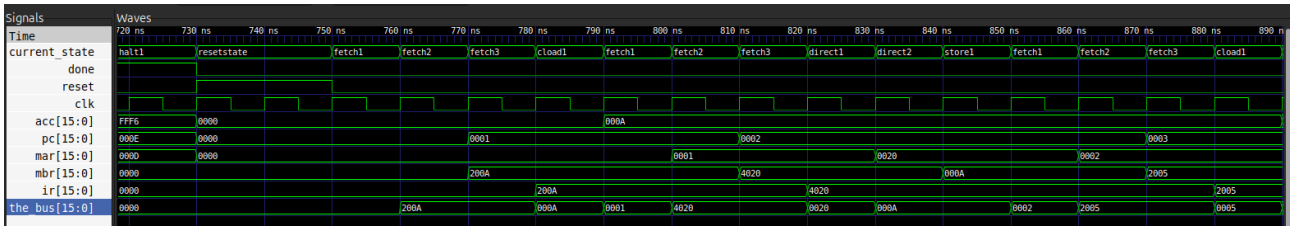
## III



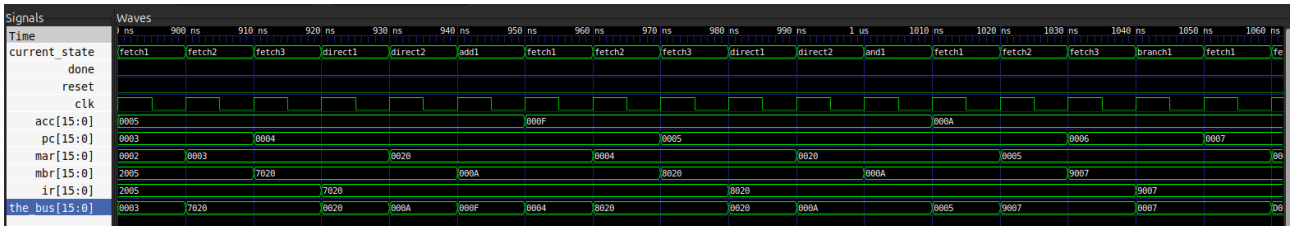

## IV



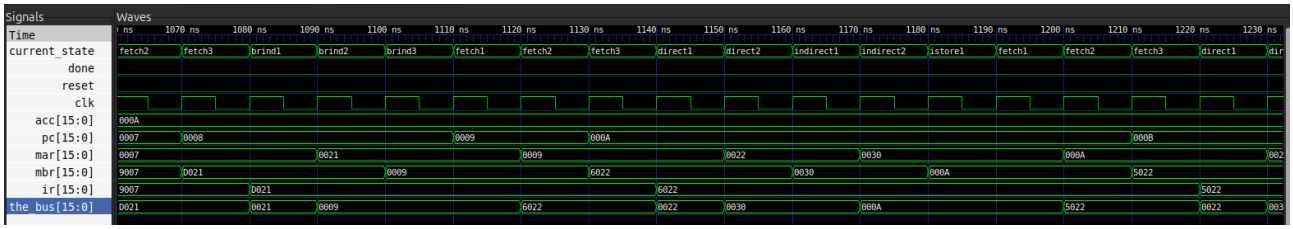

## V



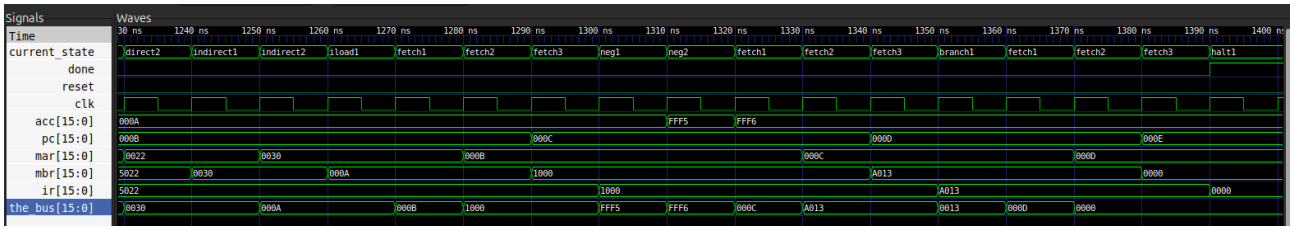

## VI



VII


Signals
Waves
Time
current_state
done
reset
clk
acc[15:0]
pc[15:0]
mar[15:0]
mbr[15:0]
ir[15:0]
the bus[15:0]
1070 ns 1080 ns 1090 ns 1100 ns 1110 ns 1120 ns 1130 ns 1140 ns 1150 ns 1160 ns 1170 ns 1180 ns 1190 ns 1200 ns 1210 ns 1220 ns 1230 ns
fetch2 fetch3 brind1 brind2 brind3 fetch1 fetch2 fetch3 direct1 direct2 indirect1 indirect2 istore1 fetch1 fetch2 fetch3 direct1
000A
0007 0008 0009 000A 000B
0007 0021 0009 0022 0030 000A 002
9007 D021 0009 6022 0030 000A 5022
9007 D021 6022 5022
D021 D021 0009 6022 0022 0030 000A 5022 0022 003

VIII


Signals
Waves
Time
current_state
done
reset
clk
acc[15:0]
pc[15:0]
mar[15:0]
mbr[15:0]
ir[15:0]
the bus[15:0]
1240 ns 1250 ns 1260 ns 1270 ns 1280 ns 1290 ns 1300 ns 1310 ns 1320 ns 1330 ns 1340 ns 1350 ns 1360 ns 1370 ns 1380 ns 1390 ns 1400 ns
direct2 indirect1 indirect2 iload1 fetch1 fetch2 fetch3 neg1 neg2 fetch1 fetch2 fetch3 branch1 fetch1 fetch2 fetch3 halt1
000A FFF5 FFF6
000B 000C 000D 000E
0022 0030 000B 000C 000D
5022 0030 000A 1000 A013 0000
5022 1000 A013 0000
0030 000A 000B 1000 FFF5 FFF6 000C A013 0013 000D 0000

Bugs encountered:
Error:
washu2_ram.vhd:76:27:error: no declaration for "ram_mem"
```
   attribute ramstyle of ram_mem : signal is "block";
                         ^
```
Fix: put the line later after declaration

Error:
washu2_cpu.vhd:105:14:error: choice subtype is not locally static
```
       with std_logic_vector'(bus_pc      &
           ^
```
Fix: need to declare a new signal; error is because std_l… isn’t constrained by size.

Error:
fetch2: mbr reads from bus, mar not updated yet.
Fix: The ram_addr signal was missing. Made changes accordingly.
This fixes the error because now we don’t need to wait an extra cycle for mar to update, ram_addr gets the info from the_bus and doesn’t have to wait for clock rising edge.

Cycle Count Table

| Instruction | Cycle Count |
|---|---|
| HALT | 4 |
| NEGATE | 5 |
| CLOAD | 4 |
| DLOAD | 6 |
| DSTORE | 6 |
| ADD | 6 |
| AND | 6 |
| BRANCH | 4 |
| BRZERO | 4 |
| BRPOS | 4 |
| BRNEG | 4 |
| BRIND | 6 |
| ILOAD | 8 |
| ISTORE | 8 |

Self-assessment

The hardest part of the project was managing the latencies since signals in a clocked process would update on the next clock cycle. These latencies were mainly an issue in reading from memory, namely the latency in the update of the MAR register. The ram_addr signal was a work-around which allows the cpu to work properly without needing an extra cycle. Instead of waiting for the rising clock edge, whenever MAR is supposed to read from the bus, we also let ram_addr directly read from the bus and carry the address to the ram, which takes care of the latency.