COL216 Assignment-2 Part 3

by Harshit Mawandia 2020CS10348

Report:

I have made 8 files + 1 testbench.

1. ALU.vhd
2. register.vhd
3. Memory.vhd
4. programCounter.vhd
5. flags.vhd
6. condChecker.vhd
7. processor.vhd
8. FSM.vhd

ALU, Register, Memory, PC, flags and Condition Checkers are components, while FSM and Processor are used for binding logic and gluing the components together.

While most of the components have been reused from the previous part of the assignment, I have completely re-written the code for “processor.vhd”. Also, FSM is a new addition which is used for managing the state of the multi-cycle processor. ProgramMemory is no longer used.

Program Counter is now only storing the incoming value of PC into the register that is PC_out.

FSM.vhd also controls all the multiplexers like PW, IorD, Rsrc, M2R, RW, Asrc1, Fset, IW, DW, AW, BW, ReW, Asrc2, op and MW. It has a total of 9 states starting from 0 and ending at 8. 0 is the initial state for every instruction and after every instruction is executed the FSM goes back to initial state 0.

In the previous assignment, processor.vhd did most of the work and controlled the logic, while here we only use it to control the input signals to various components.

I have used the following libraries:

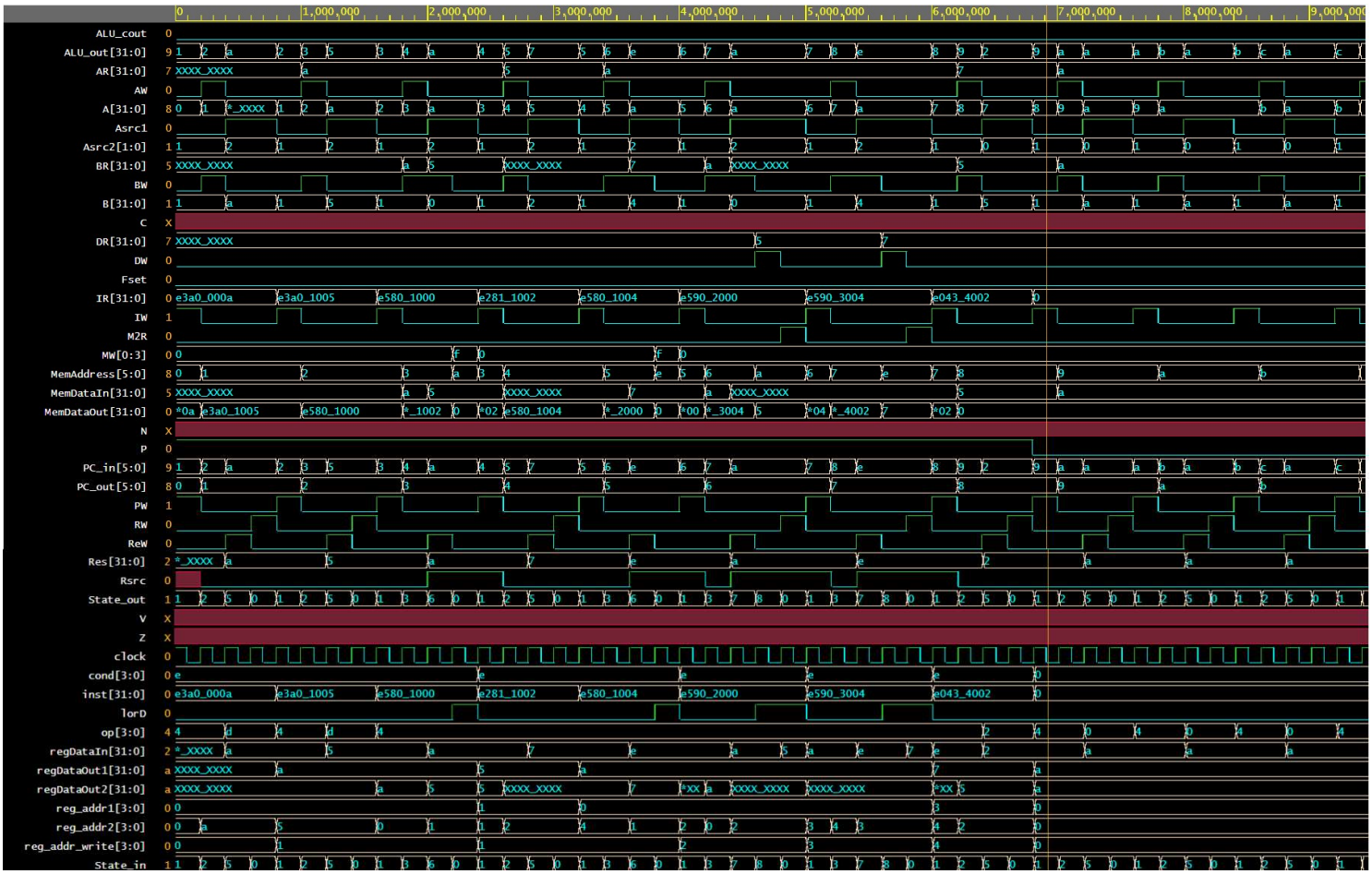
```
library IEEE;
use ieee.NUMERIC_STD.all;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

My program is running correctly for the given test cases as well as a few of my own. All have worked correctly uptill this point.

I have attached the EPWave profile for the first test program that was given in Part 2 of the Project.

```
signal mem_array : memory_type :=
                   (0 => X"E3A0000A",
                    1 => X"E3A01005",
                    2 => X"E5801000",
                    3 => X"E2811002",
                    4 => X"E5801004",
                    5 => X"E5902000",
                    6 => X"E5903004",
                    7 => X"E0434002",
                    others => X"00000000"
                   );
```

```
.text
mov r0, #10
mov r1, #5
str r1, [r0]
add r1, r1, #2
str r1, [r0, #4]
ldr r2, [r0]
ldr r3, [r0, #4]
sub r4, r3, r2
.end
```



The program ends when the instruction (IR) becomes 0, we can also use a reset state to set all signals to 0 after this point.