COL216 ASSIGNMENT 2 STAGE 3

Manshi Sagar - 2020CS50429

February 2022

1 Introduction

This stage involves modifying the design of stage 2 to make it multi-cycle design. The set of instructions to be executed and the variants to be supported remain same.

The implementation and structure of modules designed in Stage 1(register file, ALU) remains same as described in Stage 1 Report.

The implementation and structure of modules designed in Stage 2(flags, condition checker and instruction decoder) remains same as described in Stage 2 Report.

The new / updated modules in this stage are :

- Memory
- Program counter
- Finite State Machine
- Gluelogic

2 Program Counter

Two changes are done in the component containing PC.

- Logic for addition and sign extension is removed. ALU will provide the next address that goes into PC.
- A write enable signal(PW) will be required since PC will not clock in every cycle.

3 Memory

Program memory and Data Memory are now combined in one module : Memory.
Memory is internally divided into 2 parts:

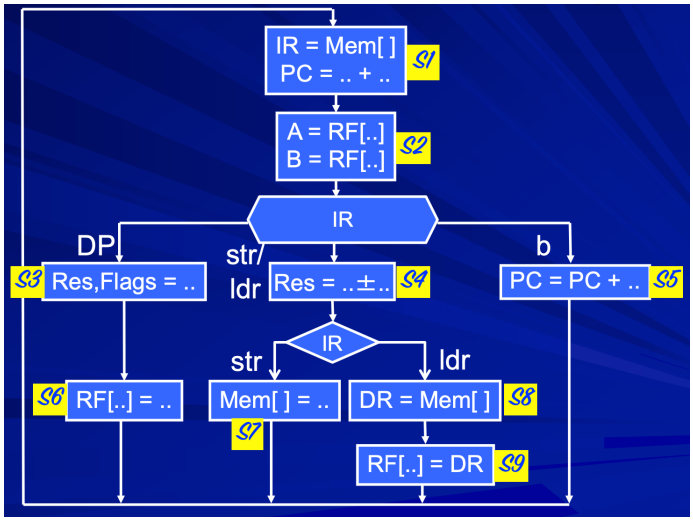
- Instructions are stored in an array(of size 64) named as prog-memory
- Data is stored in an array (of size 64) named as data-memory.

To switch between data memory and program memory, a select signal called **IorD** is used.

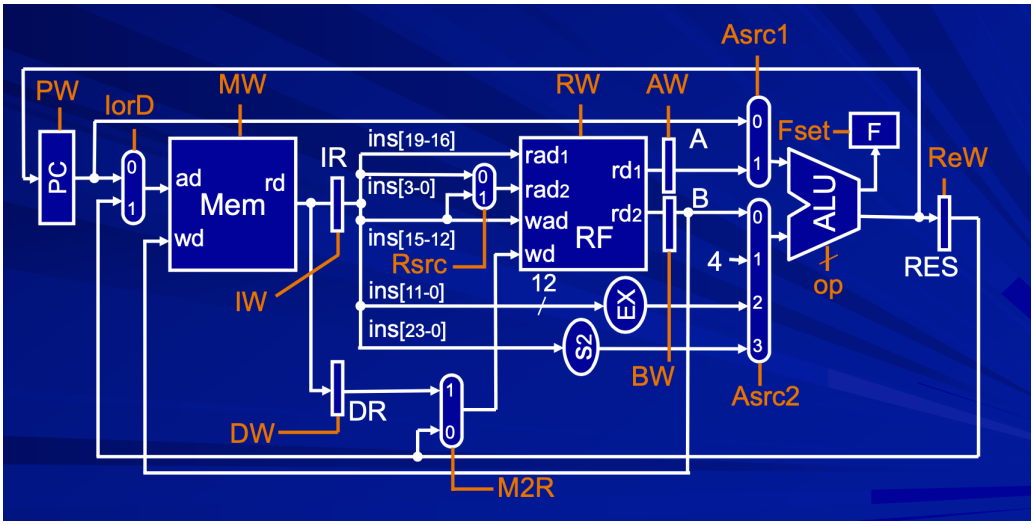
- IorD = '0' : to access Program memory
- IorD = '1' : to access Data memory

4 Finite State Machine

The states referred to in the code for FSM are named as follows:



The control signals set in each state of FSM are as follows:



- State : S1
  IorD = 0 to access program memory
  PW = 1 to write PC + 4 in PC
  IW = 1 to write in IR register
  Asrc1 = 0 to choose output from PC as operand 1
  Asrc2 = 1 to add 1 to word address
  fsm-opcode = opcode for add
  Next state is S2 for all instructions

- State : S2
  AW = 1 to write in register A
  BW = 1 to write in register B
  Next state = S3 for DP instructions
  Next state = S4 for DT instructions
  Next state = S5 for BRN(branch) instructions

- State : S3
  ReW = 1 to write in RES register

F-set = 1 to set flags
Asrc1 = 1 to set operand 1 as register A
if(DP-operand2-src=register) then Asrc2 = 00 ; else Asrc2 = 10 ;
Next state = S6

- State : S4
  ReW = 1 to write in RES register
  Asrc1 = 1 to set operand 1 as register A
  Asrc2 = 2 to set operand 2 as immediate field
  RW = 0 to read from Register file
  RSRC = 1
  Next state = S7 for Store instruction
  Next state = S8 for load instruction

- State : S5
  PW = 1 if branching is needed , else PW = 0
  Asrc1 = 1 to set operand 1 as register A
  Asrc2 = 3 to set second operand as branch offset
  IorD = 0 to write in instruction memory
  Next state = S1

- State : S6
  RW = 1 to write in register file
  M2R = 0. to choose ALU result as write-data
  Next State = S1

- State : S7
  MW = 1 to write in register memory
  IorD = 1 to access data memory
  Next state = S1

- State : S8
  DW = 1 to write inDR register
  IorD = 1 to access data memory
  Next state = S1

- State : S9
  RW = 1 to write in register file
  M2R = 1 to choose DR as write-data
  Next state = S1

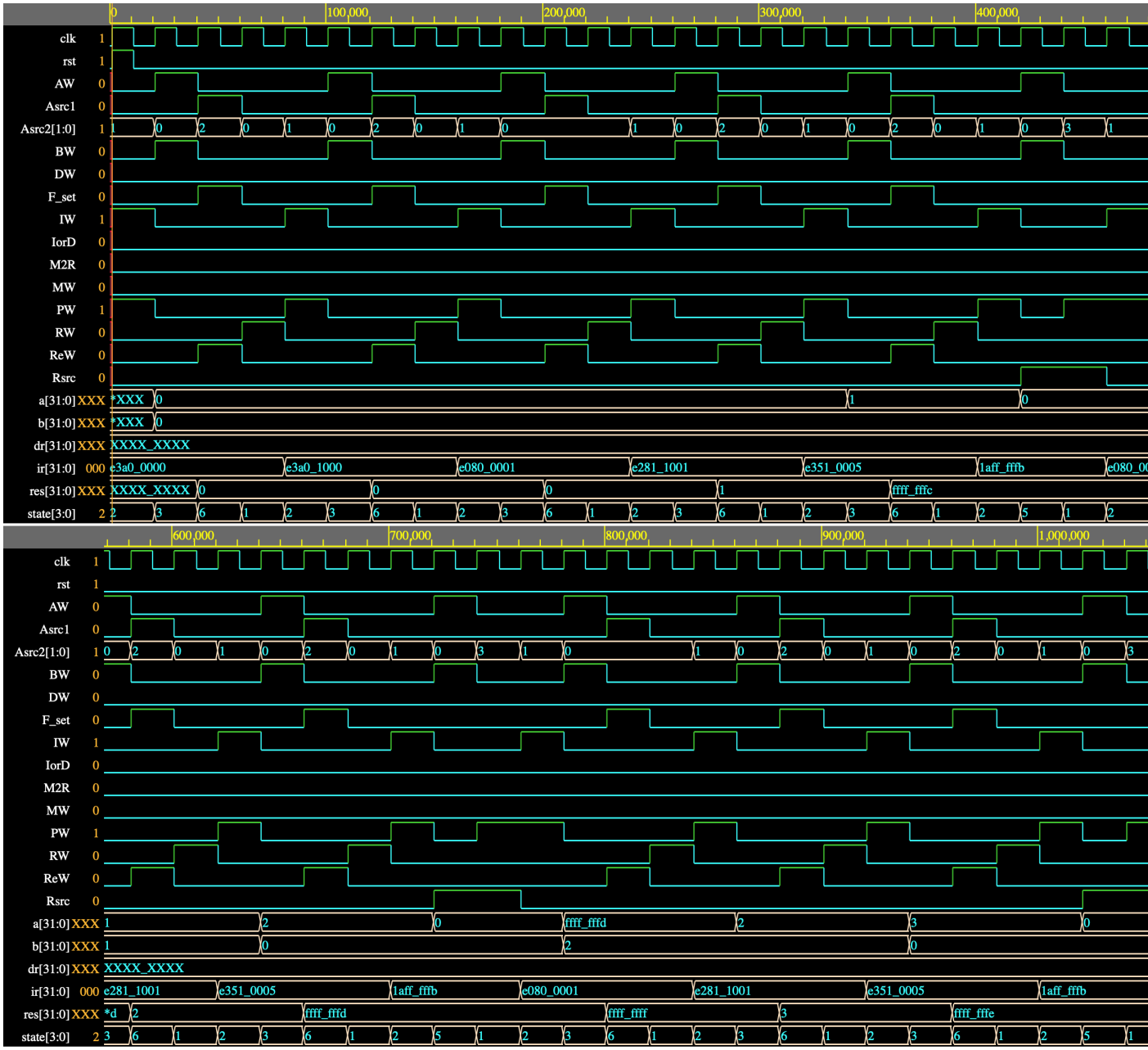
5 GlueLogic

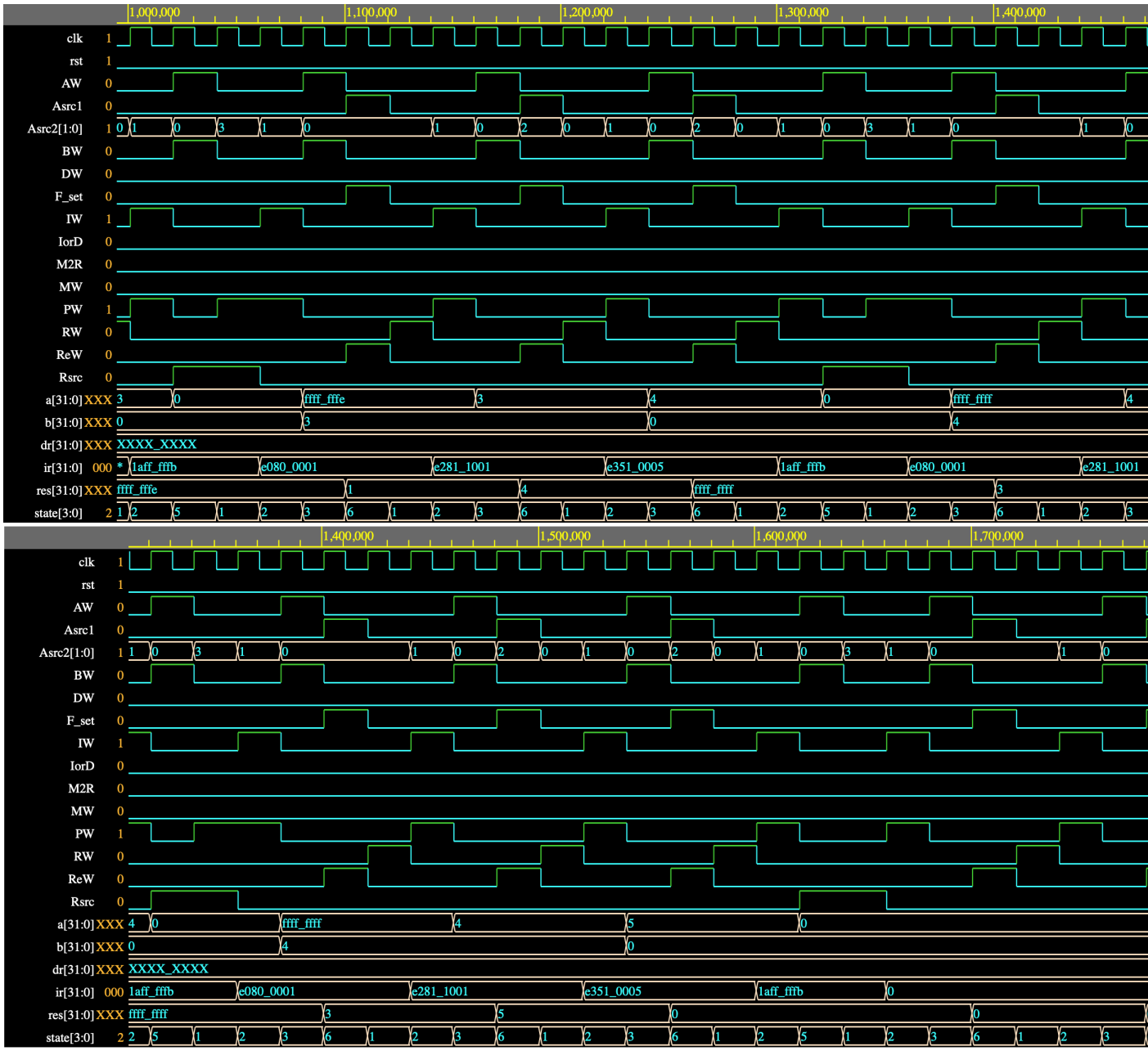
Here, according to each value of select signal and write enable signals, appropriate values are set for IR, DR, A, B, RES registers and other port signals.

```
rad1<=ir(19 downto 16);
wad<=ir(15 downto 12);
mem_wd<=b;
if(IW='1') then
    ir<= mem_rd;
    end if;
if(DW='1') then
    dr<=mem_rd;
    end if;
if(AW='1' ) then
    a<=rd1;
    end if;
if(BW='1') then
    b<=rd2;
    end if;
```

```
if(ReW='1') then
    res<=result;
    end if;
if(PW='1') then
    pc_in<=result(29 downto 0)&"00";
    end if;
if(MW='1') then
    mem_write_enable<="1111";
else mem_write_enable<="0000";
    end if;
if(M2R='1') then
    wd<=dr;
else wd<=res;
    end if;
if(Rsrc='0') then
    rad2<= ir(3 downto 0);
else rad2<=ir(15 downto 12);
    end if;
if(IorD='0') then
    mem_ad<=pc_out(7 downto 2);
else mem_ad<=res(5 downto 0);
    end if;
if(RW='1') then
    rf_write_enable<='1';
else rf_write_enable<='0';
    end if;
if(Asrc1='0') then
    operand1<="00" & pc_out(31 downto 2);
else operand1<=a;
end if;
if(Asrc2="00") then
    operand2<=b;
elsif(Asrc2="01") then
    operand2<=X"00000001";
elsif(Asrc2="10") then
    operand2<="00000000000000000000" & ir(11 downto 0);
else operand2<="00000000"& branch_offset; --word offset;
end if;
if(Asrc2="11") then carryin<='1';
else carryin<=C;
end if;
```

6 Testcases


clk
rst
AW
Asrc1
Asrc2[1:0]
BW
DW
F_set
IW
IorD
M2R
MW
PW
RW
ReW
Rsrc
a[31:0]
b[31:0]
dr[31:0]
ir[31:0]
res[31:0]
state[3:0]
e3a0_0000
e3a0_1000
e080_0001
e281_1001
e351_0005
1aff_fffb
ffff_fffc
XXXX_XXXX
ffff_fffd
ffff_ffff
ffff_fffe


1,000,000
1,100,000
1,200,000
1,300,000
1,400,000
clk 1
rst 1
AW 0
Asrc1 0
Asrc2[1:0] 1
BW 0
DW 0
F_set 0
IW 1
IorD 0
M2R 0
MW 0
PW 1
RW 0
ReW 0
Rsrc 0
a[31:0] XXX
b[31:0] XXX
dr[31:0] XXX
ir[31:0] 000
res[31:0] XXX
state[3:0] 2
ffff_fffe
e080_0001
e281_1001
e351_0005
1aff_fffb
ffff_ffff
XXXX_XXXX
1,400,000
1,500,000
1,600,000
1,700,000