Microprocessors Project 2 - IITB-RISC Pipeline

Meet Udeshi - 14D070007
Arka Sadhu - 140070011
Shruti Hiray - 14D0700XX
Ravi Sharma - 14D0700XX

November 2016

List of Control Signals

NOP bit

NOP bit will be active low, so for each instruction, NOP is high. When NOP is low then that instruction will not be executed.

NOP bit controls the output of individual control decoders of each stage. If NOP bit is set, all control signals are disabled for that stage.

RR Control

Control Signal	Description
JLR	Controls stall for JLR instruction. Goes to PC load mux and sets NOP of previous stages.
A1c/A2c	Controls mux for A1/A2 input of Register File.
RDc	Controls mux for RD(destination register) address.
LM/SM	Controls ALUI2 mux for passing Zero-Padded-PE output to ALU.
Dmem	Controls mux for choosing Dmem and RM for store operations.

Exec Control

Control Signal	Description
BEQ	Controls stall for BEQ instruction if ALU gives zero. Goes to PC load mux and sets NOP of previous stages.
ALU_c	Controls operation to be done by ALU (ADD/NAND/XOR).
Flag	Enables flag forwarding block.
Imm	Controls mux for ALUI2 to choose Immediate data.
PC1	Controls mux for ALUI1 to choose constant "+1". Controls mux for ALUI2 to choose PC.
LHI	Controls mux for ADmem to send ALUO or Immediate data(for LHI).
C/Z	Controls flag muxes to pass flags from ALU if instruction sets them.

Mem Control

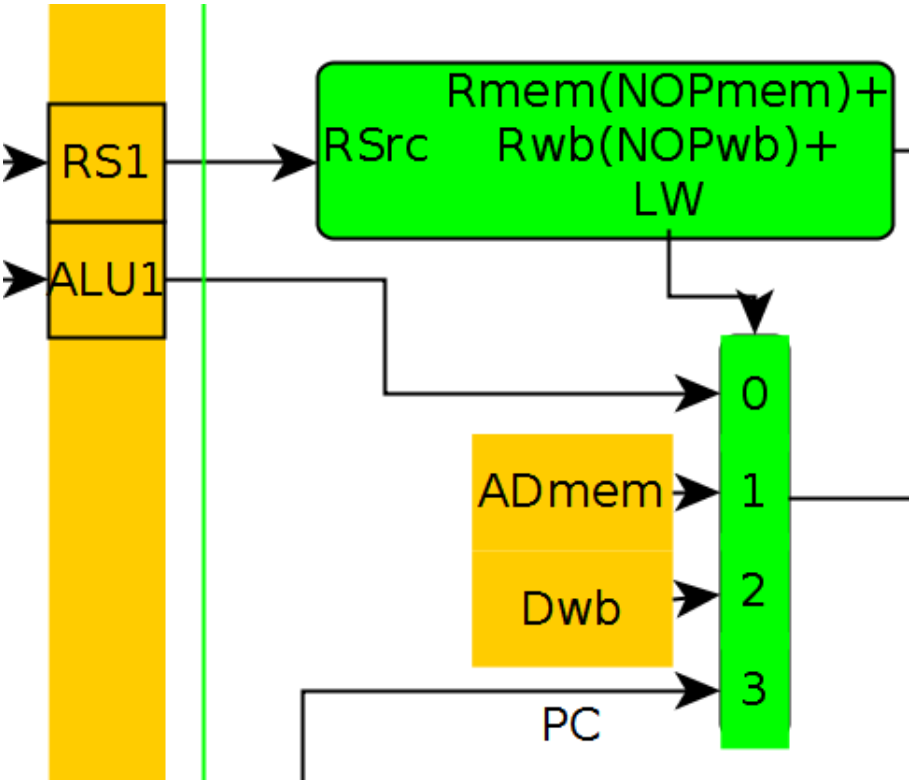
Control Signal	Description
LW	Goes to Register Forwarding blocks in Exec stage to determine LW stall.
MW/MR	Controls Read/Write enable of RAM.
Out_c	Controls mux to select RAM data output or ALU output.
Zc	Controls whether Z flag is updated by data read.

WB Control

Control Signal	Description
Wen	Enable data write for Register File
Cen/Zen	Enable write for C/Z registers.

Combinational Blocks

Forwarding Block



This block will compare **Rsrc** with **Rmem** and **Rwb** and output a control signal for mux to select the corresponding data. If **Rsrc** is equal to **R7**, we need to select the current PC.

Also, if **LW** is set for Mem stage and **Rsrc** matches, we need to stall.

Logic:

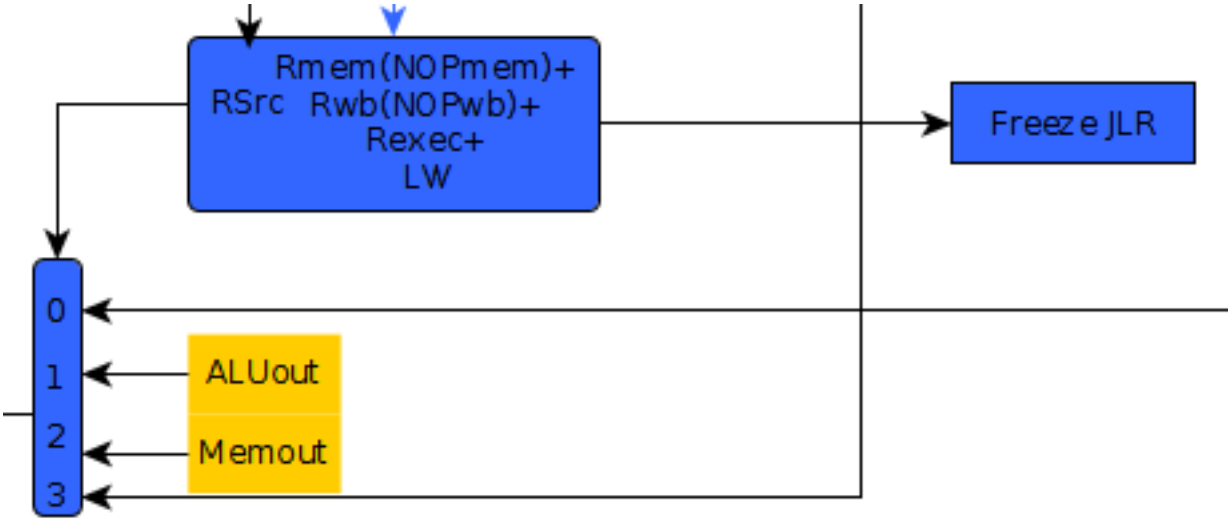
```
if(Rsrc == R7){
    Out = PC;
} else {
    if(NOPmem == 0 && Rsrc == Rmem && Wen_mem == 1){
        if (LW == 1) {
            Stall;
        } else {
            Out = ADmem;
        }
    } else if(NOPwb == 0 && Rsrc == Rwb && Wen_wb == 1){
        Out = Dwb;
    }
}
```

LW Stall Block

It takes input from the 3 forwarding blocks and if we want to stall, it disables write of all previous pipeline registers(and PC) so they stay in same state, and sets **NOP** bit of **E/M** pipeline register to send **NOP** instruction (i.e. stall) to Mem stage.

Current instruction in Exec stage will be recomputed using the forwarding values obtained from WB stage(which now has **LW** instruction).

Forwarding Block for jlr and Freeze Logic



This works the same as the normal forwarding block except that this is now in the register read stage. It compares **Rsrc** with **Rexec**, **Rmem** and **Rwb** and output a control signal for the mux to select the corresponding data. Similarly if **Rsrc** is **R7** we need to select current PC. Also we will have to output to pipeline freeze logic. We may need to freeze the pipeline sometimes, because we might not have the required value of the register until the end of the execute stage. This is done by **freeze_jlr** which will disable the write signals for all the Pipeline registers before RegRead, ie DRR and FD will be frozen.
Logic for Data Forwarding if there is JLR.

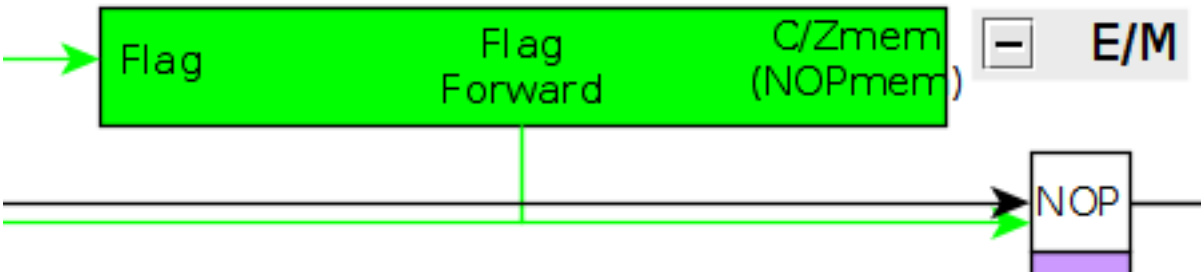
```
Out = D1;
if (Rsrc == R7)
{
  Out = PC;
}
else
{
  if (NOPexec == 0 && Rsrc == Rexec && Wen_exec==1)
  {
    if(LWexec == 1)
    {
      freeze_jlr;
    }
    else
    {
      out = ALUout;
    }
  }
  elsif (NOPmem == 0 && Rsrc == Rmem && Wen_mem == 1)
  {
    if (LWmem == 1)
    {
      out = Memout;
    }
  }
}
```

Forwarding Block Inside the Regfile

We note that there might be cases in which the value of the register in the regfile is updated by the writeback pipeline register, and at the same time the register is being read by A1 and A2.The value in the register will be updated at the end of the cycle. Therefore in case A1(or A2) is the same as A3, we need to do dataforwarding. Extra logic inside the Register File:

```
if (A1 == A3)
{
  D1 = D3;
}
if (A2 == A3)
{
  D2 = D3;
}
```

Flag Forwarding Block



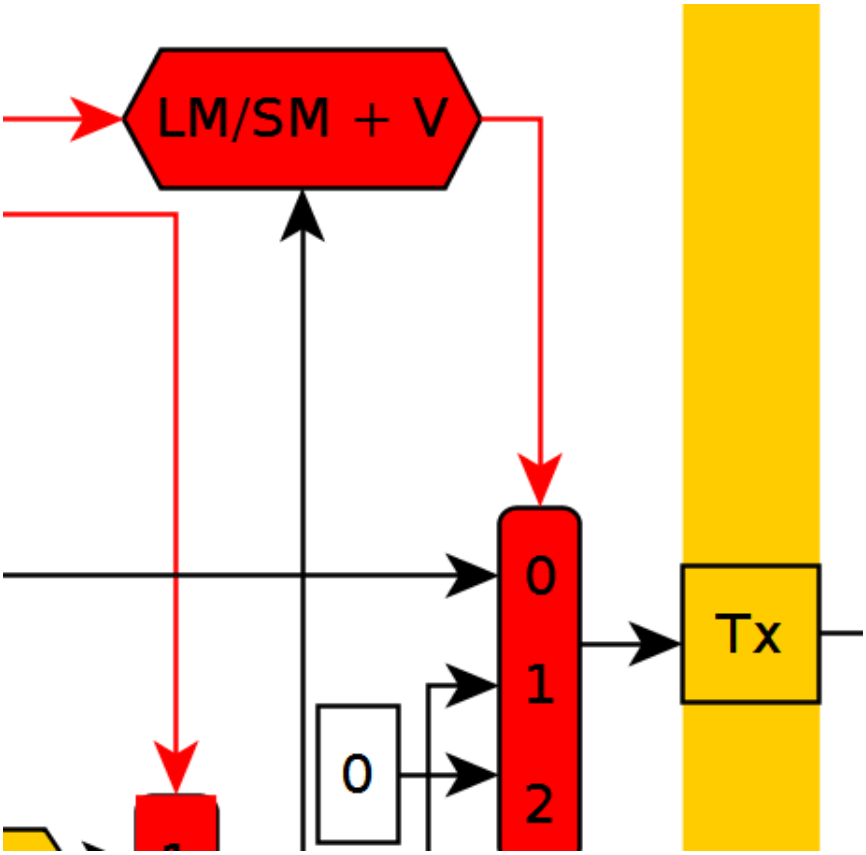
Used by **ADC/ADZ/NDC/NDZ** instructions to determine whether to execute or change to **NOP**.

Logic:

```
if(NOPmem == 1){
    if(Flag == C && Cmem == 0){
        Dont execute;
    } else if(Flag == Z){
      if (LW == 1)
      {
        Stall;
      }
      else
      {
        if (Zmem = 0)
        {
          Dont execute;
        }
      }
    }
}
```

If we decide to not execute, the Exec state's control decoder is disabled, and **NOP** bit is set in **E/M** pipeline register to send **NOP** instruction. This means the instruction will not execute further.

LM/SM Initialisation and Freeze logic

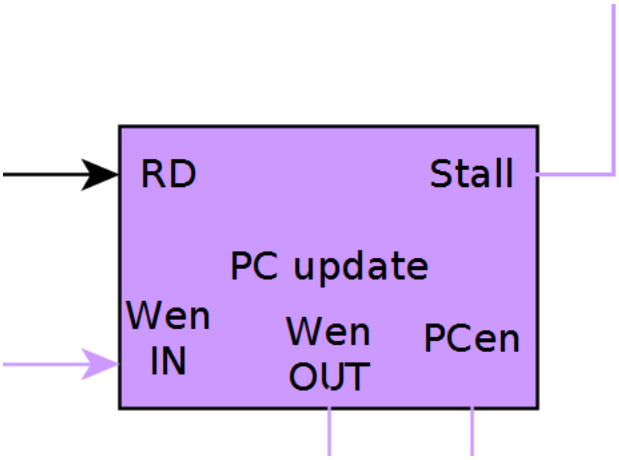


Initialisation logic:

```
if(LM/SM == 0 && V == 0){
    Tx = 0;
} else if(LM/SM == 1 && V == 0){
    Tx = Immediate;
} else if(V == 1){
    Tx = Txn;
}
```

When V = 1, then **Wdis** is on for previous two pipeline registers and PC so write is disabled and hence the next two instructions loaded are frozen in fetch and decode stage.

PC update block

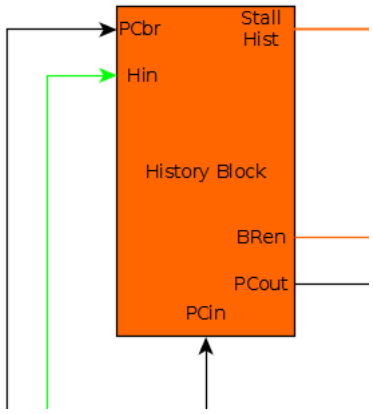


Stall will make all stages **NOP** so that no instructions are executed, and PC sets to branched value. **WenOUT** will enable the PC branch mux to take from **Dwb**. In normal case, **R7upd** enables writing of **PC** to **R7**.

Logic:

```
if (NOP == 1){
    if (WenIN == 1 && RD == R7){
        R7upd = 0;
        WenOUT = 1;
        Stall = 1;
    } else {
        R7upd = 1;
    }
}
```

History Block



PCin is PC of the current fetched instruction.
BRen is output control signal to choose whether to update PC from history table or not.
PCout is the next PC for current PC in history table.

PCbr is the PC of the **BEQ** instruction which may/may not branch.
Hin is the action that has been taken by that **BEQ** instruction.

StallHist is the control signal to flush and stall the pipeline if history bit mismatched.

Internally, the **History Block** stores a history table with **PC_curr**,**PC_next** and **H_curr**.

Logic for **BRen**,**PCout** and **StallHistory**:

```
for(PC_curr,PC_next,H_curr in HistoryTable){
    if(PCbr == PC_curr){
        if(Hin != H_curr){
            H_curr = Hin;
            StallHistory = 1;

            // PC+1 if Hin = 0, else BEQ will update directly
            BRen = not(Hin);
            PCout = PCbr + 1;

            break;
        }

    } else if(PCin == PC_curr){
        if(H_curr == 1){
            BRen = 1;
            PCout = PC_next;
        } else {
            BRen = 0;
        }
    }
}
```

Control signals for each instruction

ADD/NDU

Stage	**Signals**
Decode	–
Reg Read	A1c = ra A2c = rb RDc = rc
Exec	ALU_c = add/nand Z = 1 C = 1(for add)
Mem	–
Write Back	Wen = 1 Cen = 1(for add) Zen = 1

ADC/ADZ/NDC/NDZ

Stage	Signals
Decode	–
Reg Read	A1c = ra A2c = rb RDc = rc
Exec	ALU_c = add/nand Z = 1 C = 1(for add) Flag = C/Z
Mem	–
Write Back	Wen = 1 Cen = 1(for add) Zen = 1

ADI

Stage	Signals
Decode	–
Reg Read	A1c = ra RDc = rb
Exec	ALU_c = add Z = 1 C = 1 Imm = 1
Mem	–
Write Back	Wen = 1 Cen = 1(for add) Zen = 1

LHI

Stage	Signals
Decode	LHI = 1
Reg Read	RDc = ra
Exec	LHI = 1
Mem	–
Write Back	Wen = 1

LW

Stage	Signals
Decode	–
Reg Read	A1c = rb RDc = ra
Exec	ALU_c = add Imm = 1
Mem	LW = 1 MR = 1 Out_c = 1 Zc = 1
Write Back	Wen = 1 Zen = 1

SW

Stage	Signals
Decode	–
Reg Read	A1c = rb A2c = ra Dmem = D2 RM = A2
Exec	ALU_c = add Imm = 1
Mem	MW = 1
Write Back	–

BEQ

Stage	Signals
Decode	–
Reg Read	A1c = ra A2c = rb
Exec	BEQ = 1 ALU_c = xor
Mem	–
Write Back	–

JAL

Stage	Signals
Decode	JAL = 1
Reg Read	RDc = ra
Exec	ALU_c = add PC1 = 1
Mem	–
Write Back	Wen = 1

JLR

Stage	Signals
Decode	–
Reg Read	RDc = ra A1c = rb JLR = 1
Exec	ALU_c = add PC1 = 1
Mem	–
Write Back	Wen = 1

LM

Stage	Signals
Decode	LM/SM = 1
Reg Read	LM/SM = 1 RDc = pe A1c = ra
Exec	ALU_c = add
Mem	MR = 1 Out_c = 1
Write Back	Wen = 1

SM

Stage	Signals
Decode	LM/SM = 1
Reg Read	LM/SM = 1 A2c = pe A1c = ra
Exec	ALU_c = add
Mem	MW = 1
Write Back	–