Pipelined IITB-RISC-22

Koustubh Rao
Moiz Shakruwala
Pinkesh Raghuvanshi
Sarthak Mittal

May 10, 2022

Contents

1 Abstract

Project submission for the course CS 230: Digital Logic Design and Computer Architecture by the collaborative efforts of Sarthak Mittal, Pinkesh Raghuvanshi, Moiz Shakruwala and Koustubh Rao of CSE 2024 under the guidance of Professor Virendra Singh.

2 Intoduction

IITB-RISC is a 16-bit very simple computer developed for the teaching that is based on the Little Computer Architecture. The IITB-RISC is an 8-register, 16-bit computer system. It has 8 general-purpose registers (R0 to R7). Register R7 is always stores Program Counter. All addresses are short word addresses (i.e., address 0 corresponds to the first two bytes of main memory, address 1 corresponds to the second two bytes of main memory, etc.). This architecture uses condition code register which has two flags Carry flag (C) and Zero flag (Z). The IITB-RISC is very simple, but it is general enough to solve complex problems. The architecture allows predicated instruction execution and multiple load and store execution.There are three machine-code instruction formats (R, I, and J type) and a total of 17 instructions.

The processor is a five stage pipelined processor. It follows the standard 5 stage pipeline (Instruction fetch, instruction decode register read, execute, memory access, and write back). The architecture is optimized for performance by including hazard mitigation techniques like forwarding and branch prediction.

3 Data Path Components

3.1 Instruction Fetch

- **PC** : Stores "Program Counter" for current instruction
- **IM** : "Instruction Memory" - memory from which we get the instructions.
- **PC++** : Increments the Program Counter by 1 to fetch the next possible instruction.
- **Brach Prediction Table Block** : Store the already encountered "BEQ" instruction PCs, the branch address and the recent history bit (whether it was taken or not)
- **MUX** : Selects one of the two: i) address given by the branch prediction table and ii) address given by PC++ . The control signal is also generated by the table block itself

3.2 Instruction Decoder & Register Read

- **SE** : Sign extends the 6 bit or 9 bit immediate to 16 bit. It also observes the extra LLI bit in which case it only packs the immediate data by zeros.
- **SE+PC** : Adds the PC value of the current instruction to SE value.
- **Decoder**: Outputs the control signals based upon the current instruction.
- **MUX0** : Chooses between PC+1 and (SE+PC) value. Selects (SE+PC) when JAL instruction.
- **LS** : Left shifts the sign extended value.
- **RF** : Register file: the registers (R0-R7).
- **MUX1** : Directs inputs between (SE+PC) and LS. SE+PC when JAL instruction and LS when LHI instruction.
- **MUX2** : Directs the inputs between AR2 from IDRR register and the output from LMSM block. Mux before AR2.
- **MUX3** : Directs the inputs between DO1 and the ALU output which gives the DO+1 value during lmsm instruction. Mux after DO1.
- **Hazard Mux** : Mux controlled by hzdLogic.
- **Hazard Logic** : The Hazard Block present in RR stage. Sends Input to the PC register according to the instruction.
  1. SE during LLI and R7 is the destination.
  2. SE_LS during LHI and R7 is the destination.
  3. DO1 during the JLR instruction.
- **LMSM mux** : Choses the input for the LMSM block between the data after decoder or from the IDRR pipeline.
- **LMSM block** : It operates the LM and SM operation.

3.3 Execution

- **ALU** : Executes different logical operation.
- **Flags** : Contains the carry, zeros and overflow flag bits.
- **Forwarding Blocks** : Checks if there is dependency present between RREX pipeline and the pipeline registers in the further stages. It accordingly controls the forward logic muxes.
- **MUX1** : The second mux before ALU second input. It directs inputs according to the arithmetic operation or address calculation.
- **MUX2** : Mux just before the second input of ALU. Controlled by the LMSM block. Sends 1 during the LM or SM operation to increment the address.
- **Hazard Mux** : Mux controlled by the hazard logic block in execution stage.
- **Staller** : Stalls the pipeline registers if there is an immediate dependency after the load instruction.
- **Hazard_EX** : Controls the hazard when during arithmetic instruction the destination is R7. It loads the PC with the required value controlling the hazard mux. Flushes the pipeline.

3.4 Memory Access

- **MEM** : Memory from which the data is read or written into. Data Memory
- **MUX1** : Mux before address of data memory. Steers inputs between ALU output (address calculation) or DO1 (during LM or SM operation).
- **Hazard MUX** : Mux being controlled by the hazard block in MW stage.
- **Hazard MM** : Checks the hazard during load instruction when the destination is R7. It suitably loads the PC value by controlling the hazard mux.

3.5 Write Back

- **Flags_user** : Flags that are visible to user.
- **WB_mux** : Decides the input to be written in the register file. The inputs according to the instructions are
  1. ALU output : During arithmetic instructions.
  2. LS_PC : LHI instruction.
  3. SE : LLI instruction.
  4. PC+1 : JAL, JLR instruction.
  5. Mem_out : LW, LM instruction.
- **R7_mux** : It decides the input to the R7 in the register file. The inputs according to the instructions are
  1. DO1 : During JLR instruction.
  2. PC+1 : Normal Program Flow.
  3. LS_PC : For BEQ instruction when the branch is taken.
- **Conditional and Hazard Control**: Takes care of the following cases
  1. Conditional arithmetic instruction is not taken and dependency present in previous pipeline registers.

2. Conditional arithmetic instruction when the destination is R7.
3. JLR instruction when the destination is R7.
4. JAL instruction when the destination is R7.
5. Check for BEQ instruction if the branch is taken or not.
6. Flushes the pipeline if any of the above condition is true.

- **Hazard_mux** : Controlled by the above logic block. Decides the input to PC register.
  1. (SE+PC) when branch is taken in BEQ.
  2. PC+1 when JLR and JAL hazard is seen.
  3. Otherwise the default value coming at 0 input.

4 Control Signals Flow

4.1 Instruction Fetch

- PC incremented and sent to PC register
- Instruction is fetched from Memory
- Branch Prediction Table (BPT) used if applicable

4.2 Instruction Decode & Register Read

- Creation of Signals:
  - Control Signals for muxes in the other stages
  - Sign Extended value of the Immediate data in the instruction for I and J type of instructions.
  - Sum of the sign extended value and the current PC value
  - Input for the mux which selects if PC+1 or (SE+PC) value has to be sent to the PC register. (SE+PC) value is sent when JAL instruction is encountered.
  - Clear bit for when JAL instruction arrives to clear the pipeline register (IF-ID)
  - Control bit for Sign Extend which decides if to sign extend the 6 bit or 9 bit immediate data
  - LLI bit for SE which just packs the immediate data with zeros to make the data 16bit
  - Address of the register file (AR1, AR2, AR3)
  - The LM or SM data to be given to the LMSM block. It holds the data of the registers whose data has to be stored or loaded
  - The ALU control bits which controls operations done by the ALU.
  - Flag control bits which toggles flag registers
  - The condition bits that specify the presence or absence of conditional arithmetic instruction.
  - Register write and Memory write signals.
  - Control Signal for the BPT indicating to it if the instruction is BEQ type.
  - Branch Prediction Table produces the index of the instruction from the table and also information on whether the branch was taken or not.
- Consumption of Signals:
  - The LM or SM data is consumed by the LMSM block and also LM or SM bit which activates the LMSM block
  - AR1 and the AR2 data for the register file
  - The control signal for the mux which steers inputs between (SE+PC) and the left shifted value of the sign extended immediate data
  - LMSM block creates the control signal for the mux before AR2 which steers the input between the original AR2 and the AR2 created by the LMSM block which sets data bits to zero
  - LMSM block also creates the signal for the mux which decides the data to be sent in the DO1 register. The original DO1 value or the auto incremented value of DO1 by the LMSM block
  - RR_PC created by the hazard block in RR stage which controls the mux which decides the input to PC should be PC+1 or SE value (when R7 is AR3 in LLI) or left shifted SE (when R7 is AR3 in LHI) or DO1 (when the instruction is JLR)
- Creation of Signals: DO1 and DO2 data created from the register file

4.3 Execution

- Consumption of Signals:
  - Two control signals for the forwarding mux created by the forwarding logic block
  - The control signal for the mux which steers input between DO2 or the sign extended value DO2 is used for arithmetic operations whereas sign extended value is used for address calculation
  - The control signal for the mux which is created by the LMSM block to select 1 such that the required address is always incremented by 1 during the LM or SM instruction
  - The control signal for the hazard mux
  - The control signal for the mux created by LMSM block to store the created address in the register file for storing data during LM from the memory
- Creation of Signals:
  - The flag register values after the ALU operation
  - The ALU output data after ALU operation

4.4 Data Memory Write/Read

- Consumption of Signals:
  - Memory write control for the data memory during instructions like SW or SM
  - The control signal for the mux created by the LMSM block which steers inputs between the ALU output or the address calculated by the LMSM block
  - The control signal for the hazard mux in the Memory Write stage
- Creation of Signals:
  - The data after reading from the data memory

4.5 Write Back

- Consumption of Signals:
  - Flag values to be written in the user flag registers
  - The control signals for the mux which decides the data to be written in the register file according to the instruction (ALU output, left shifted sign extended value, (SE+PC), SE, PC+1, Memory out data)
  - Flag control bits, Condition bits, Control Signals is sent to the hazard logic block in WB stage
  - The control signal for the mux which decides the input to R7 in register file This created by the hazard block

The data after WB mux, AR3 and some of the control signals (valid, the control signals for WB mux) is sent to a temporary register This register holds data for the forwarding logic.

5 LM and SM

The LMSM block is responsible generating the control signals to run the Load and Store multiple instructions. Inherently, the instruction is a multi-cycle instruction and thus has to be performed by halting the pipeline. The LMSM block has the following inputs and outputs:

5.1 Inputs

- 8-bit data (from the instruction to be fed to the priority encoder)
  - Note that there is a mux connected to it, since the inputs for LM and SM (which are the same) will be in different clock cycles, the details of which are explained later.
- LM bit and SM bit
  - The LM and SM bits specifically tell when the instruction is LM or SM and are used to activate the block, so that priority encoder can start giving address
- clk and reset

5.2 Outputs

- Register Address: AR2 in case of SM, AR3 in case of LM
- Clear and disable for the pipeline registers
- RF_DO1_mux: Controls the mux connected to the input of DO1 register in RREX
- ALU2_mux: Controls the mux connected to the second input of ALU, decides when +1 should be the input to the ALU
- AR3_mux: Used in case of LM, this mux controls the data that is stored in AR3 in EXMM
- AR2_mux: Used in case of SM, this mux controls the input to the register file for AR2
- mem_in_mux: Controls the input to the address of the memory which is usually the output of the ALU except in LM or SM where it is DO1

NOTE – The LMSM block starts outputting the address one cycle after it is activated

The block has been built using an FSM which goes into different states, depending on whether the instruction is LM or SM.

5.3 FSM Logic for LM

- The block gets activated when the current instruction is in the RF stage. Here it is in the S1 state. The bits that control memory input, AR3, and ALU are '1' and are put through the pipeline register.
- It then moves to the S2 state, where the mux connected to input of DO1 now starts accepting the output of ALU (DO1 + 1), and the block starts outputting AR3 addresses, which will be written in the write back stage into the register file. The disable signal is high, since the registers are now disabled (RREX, IDRR, IFID and PC are disabled).
  - Note that there is a special enable signal to DO1 in RREX since that has to be enabled during the LMSM process.
- The whole cycle continues till the last bit in the input of PE goes to zero, when the valid signal goes low, and one instruction currently in the RREX register has to be disabled. The disable signal goes low as well, meaning the pipeline flow has started again.

5.4 FSM Logic for SM

- The block gets activated when the current instruction is in the decode stage.
- In the next clock cycle, the LMSM block starts outputting the AR2 address and thus, SM begins. The control bits follow the same pattern as LM, except the mux controlling the input to DO1 starts accepting the ALU output one cycle later. When the valid signal goes low, the last set of data is in the RREX register and so, in SM, no clear signal is required. The disable signal goes low and pipeline resumes.

6 Pipeline Register Contents

Pipelined Register	Components	Length (bits)
IF-ID	PC	16
	Instruction	16
	PC++	16
IDRR-RREX	PC	16
	SEPC	16
	LSPC	16
	SE	16
	CL (Control Lines)	14
	LS_PC BEQ LM LW SE_DO2 WB_mux valid LM_SM Control	1 1 1 1 1 3 3 3
	Opcode	4
	ALU Control	2
	Flag Control	3
	Condition bits	2
	Condition	2
	Write bits	2
	Write	2
	BLUT	4
	DO1	16
	DO2	16
	AR1	3
	AR2	3
	AR3	3
	PC++	16
	LM Input	8
EXMM	LSPC	16
	SE	16
	CL (Control Lines)	8
	BEQ WB_mux Valid LM_SM control	1 3 3 1
	Flag Control	3
	Condition	2
	Write	2
	AR1	3
	AR2	3
	AR3	3
	Flags	3
	DO1	16
	DO2	16
	ALU output	16

MMWB	LSPC	16
	SE	16
	CL (Control Lines)	7
	BEQ WB_mux Valid	1 3 3
	Flag Control	3
	Condition	2
	Write	2
	AR1	3
	AR2	3
	AR3	3
	ALU output	16
	Memory output	16
	DO1	16

7 Hazard Detection And Mitigation

7.1 Data Forwarding

Data is forwarded through two multiplexers. Each mux is controlled by separate forwarding unit which cheeks if any of the operand in execution stage depends on output of any instruction in further stages or current value of PC and provides corresponding data to ALU input. Each pipeline register holds valid bits corresponding to operand and destination register address of corresponding instruction. Forwarding block considers these bits to determine dependency among instruction

1. Forwarding Priority: R7 > [EX-MM] > [MM-WB] > [WBT]
2. Pseudocode

```
if ([RR_EX](ARi) is valid) then
    if([RR_EX](ARi) = ''111'') then -- R7
        Di <= [RR_EX](PC);
    elsif (([EX_MM](AR3) is valid) and ([EX_MM](AR3) = [RR_EX](ARi))) then
        Di <= [EX_MM]{Write_Data};
    elsif (([MM_WB](AR3) is valid) and ([MM_WB](AR3) = [RR_EX](ARi))) then
        Di <= [MM_WB]{Write_Data};
    elsif (([WBT](AR3) is valid) and ([WBT](AR3) = [RR_EX](ARi))) then
        Di <= [WBT](D3);
    end if;
end if;
```

7.2 Instruction-Wise Hazards and Mitigation

7.2.1 Arithmetic Instructions

- Arithmetic instructions not having R7 as destination
  - If instruction is unconditional only data forwarding is sufficient.
  - If instruction is conditional, assume instruction to be taken and check instruction in writeback stage. If instruction is false and preceding instructions depend on the current instruction, then flush the pipeline. Dependency can be found by AR1/2 and Valid bits in pipeline registers.
- Arithmetic instructions having R7 as destination
  - If instruction is unconditional then update PC in execution state and R7 in writeback stage. Flush [IFID] and [ID-RR] registers.
  - If instruction is conditional assume it to be taken i.e. update PC in execution stage. If condition becomes false (checked in Writeback stage) then flush the pipeline.

7.2.2 Load Instructions

- **LW**: Stall only if instruction in [RR-EX] is LW and instruction in [ID-RR] depends on its output.
- If instruction in MM stage is load type (both LM and LW) and destination register is R7 then flush previous registers and update PC.

7.2.3 Branch and Jump Instructions

- **JLR**: Update PC in RR stage and clear [IF-ID] and [ID-RR] register.
- **JAL**: Update PC in ID stage and clear [IF-ID] register.

- **BEQ**: BEQ instructions are assumed to be taken/not-taken based on the branch prediction table.
  - If not found in the table, they are assumed to be not-taken as the address to branch to is not known. Condition is checked in WB stage.
  - If condition becomes opposite to what was predicted, the table is accordingly edited, the PC updated and the pipeline is flushed.
  - New entries to the table are made in the decode stage

7.3 Hazard Mitigation Blocks

7.3.1 Stalling Block

- If instruction in [ID-RR] is LW and the instruction in [IF-ID] depends on earlier output, wait for one cycle.
- Clear enable pin of PC, [IF-ID] and [ID-RR]. This creates two copies of LW instruction, one in [ID-RR] and one in [RR-EX]. So, pass the same signal through register to clear [ID-RR] in next signal.
- If instruction in [IF-ID] is SM then delay SM start bit through register and MUX.

7.3.2 EX Stage Block

- This increments PC in case of ALU Instructions with R7 as destination.
- It sends output to PC and clears the registers [IF-ID] and [ID-RR].

7.3.3 MM Stage Block

- For load type instructions having R7 as destination increment PC in MM stage.
- Clear the registers [IF-ID], [ID-RR] and [RR-EX].

7.3.4 Condition Control Block

Condition	Action
Conditional Arithmetic Instruction not having R7 as output destination becomes false and preceding instructions depends on it	Clear register write signal and flush pipeline
Conditional Arithmetic Instruction not having R7 as output destination becomes false	Clear register write signal and flush pipeline. Write PC and R7 with PC+1
(JLR R7 R7)	Flush pipeline and write PC and R7 with PC+1
BEQ becomes true	Flush pipeline and write PC and R7 with (PC+SE)