Indian Institute of Technology Bombay
Department of Electrical Engineering
EE-309: Microprocessors

IITB-RISC

Team ID: 24

Saarthak Krishan (22B3959) Aniket Patel (22B3981)
Sanat Agrawal (22B3919) Naresh Kumar (22B3947)

From our learnings from Microprocessors course and the current state technological landscape, it is quite evident that microprocessors are an indispensable part of human life. Hence, we as a team, tried to venture in the domain of microprocessors, by making a 16 bit very simple computer based on the Little Computer Architecture. It is a 16 bit computer system with 8 registers and it follows the standard 6 stage pipelines (Instruction fetch, instruction decode, register read, execute, memory access, and write back).The architecture includes hazard mitigation techniques by implementing forwarding mechanism.
This report offers comprehensive details of our project, wherein we executed the project of designing and implementing a 16-bit computer system utilizing VHDL:

The DATA – PATH:

DATA – PATH COMPONENTS:

- **Instruction Fetch:**
  - **PC:** stored as register 0 in register file, stores the PC value of current instruction.
  - **IM:** Instruction memory, takes PC input and gives the instruction as indicated by PC value.
  - **Adder_16**: takes 16 - bit input, used to update PC to PC+2.
- **Instruction Decode:**
  - Generates signal for Control signals for multiplexers in the pipeline stages.
  - Makes decision like:
    - Sign Extended value of the immediate data in the instruction if present.
    - Control Signal for SE which decides if to sign extend 6 bit or 9-bit immediate data.
    - The ALU control bits which tell the ALU of its operations.
    - Flag control bits which enable or disable the flag registers.
    - Clear bit for when JAL instruction arrives to clear the pipeline register (IF-ID).

- **Register Read:**
  - Register File: Contains 8 registers of 16 bits each, which stores data or memory location etc.
  - H_MUX_1 : This mux is used on correction of immediate dependencies for operand addresses defined by bits 8-6. Inputs are all data forwarding paths.
  - H_MUX_2: this mux is used on correction of immediate dependencies for operand addresses defined by bits 5-3. Inputs are all data forwarding paths.
  - LM_SM mux1: choses between new op-code and the opcode for LM-SM. This is done because the LM-SM state should be present for 8 cycles.
  - MUX_IMM: choses the input bits 8-0, the immediate bits for pipeline register 3. It has two inputs ,either usual Imm bits or shifted 8 bits .The control line for this mux are coming from Pipeline register 3 and implemented using if-else conditions in the code .
  - 9-bit_Zero _extender: selects immediate bits from pipeline register 3. This corrects the immediate dependencies in LLI instruction.
- **Execution:**
  - ALU1: performs ADD, NAND, comparator and Complement operation. It also updates the flags such as carry, zero and overflow flags
  - ALU_MUX_A : takes Ra and Rb values as immediate is fix to alu_mux_B input.
  - ALU_MUX_B: takes constants, Rb, sign extended Imm6 bits and sign extended Imm9 bits.
  - ALU_MUX_C: used for LM-SM instruction.

- 6bit_SE: extends the imm6 bits of I type instruction
- 9bit_SE: extends the Imm9 bits of J type instruction
- 1bit_shifter: used in LM-SM stage to get the bit which is set and corresponding register in register file is selected
- 1_subtractor: used in LM-SM state to decrease additional three bits which we have stored in pipeline register 4. These three bits indicate the rf_a3 value. These three bits will be 111 until instruction LM-SM is executed.
- MUX_SHIFT: selects if direct 8 bits or shifted 8 bits are to be transferred.

Memory Write/Read:

- Data memory: data memory from which the instruction reads data writes data.
- MUX_DI: Takes input as data of registers in register file or the output of ALU1 and sends it to mem_di port.
- MUX_WRITE : Takes usual input 1/0 or takes value of the shifted 8th bit from shifter.
- ALU5: used for LM-SM instruction, to update the memory address to consecutive memory addresses. This ALU takes input as ALU5's output.
- ALU5_B_mux: control bit is the 8th shifted bit from shifter.

Write Back:

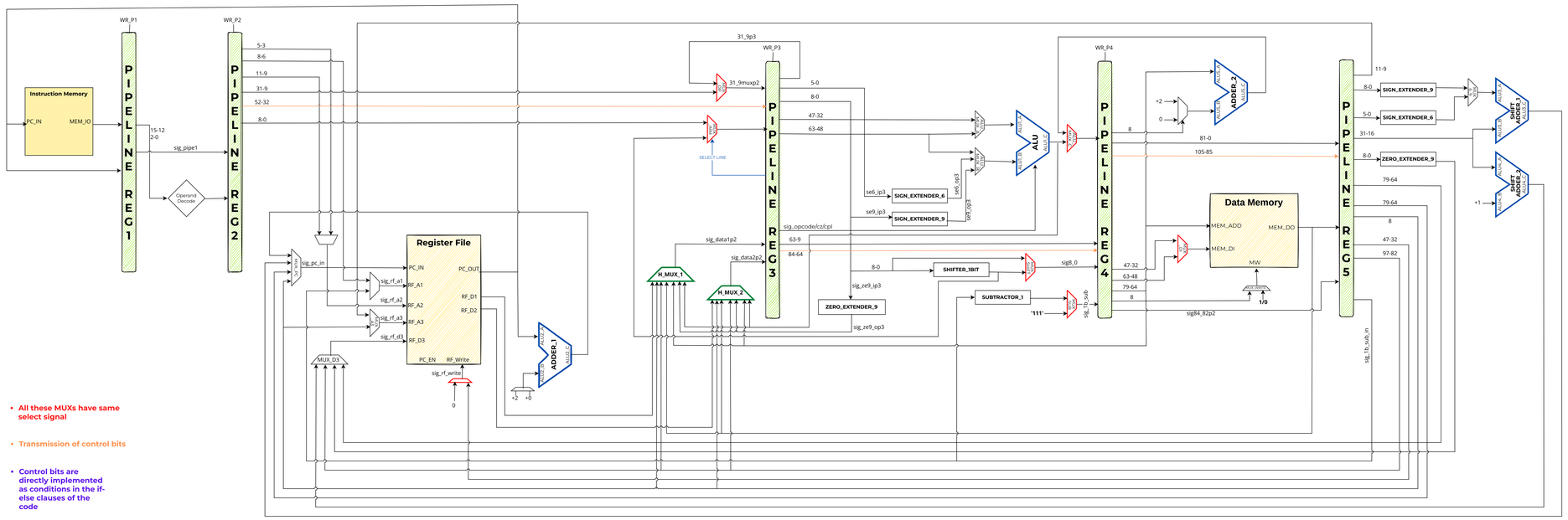
- ALU3: Calculates PC+2*Imm for instructions like BEQ, BLU, BLT and JAL
- MUX6_9 : Selects 6Imm bits or 9Imm bits corresponding
- ALU4: Updates PC, PC+2.
- 9_Zero extender: Zero extends 9Imm bits for J type instructions.

- 6_Sign Extender: Sign Extends 6Imm bits for I type instructions
- 9_Sign Extender: Sign Extends 9Imm bits for J type instructions.
- MUX_A3: have a separate input for LM-SM states.
- MUX_D3: select values to be entered in register.
- MUX_PC: 4 select line 2 are PC+2 and two from the write back stage.
- Rf_write_mux: have separate select line for LM-SM state.

PIPELINE REGISTERS:

	Pipeline Register	Components	Lengths(bits)
	PIPE1	Instruction	16(15 – 0)
		PC	16(31 - 16)
	PIPE2	Instruction	16(15 - 0)
		PC	16(PC)
	PIPE3	Instruction	16(15 – 0)
		PC	16(31 – 16)
		DATA A1	16(47 – 32)
		DATA A2	16(63 – 48)
		Instruction	16(15 – 0)

	PIPE4	PC	16(31 – 16)
		DATA A1	16(47 – 32)
		DATA A2	16(63 – 48)
		ALU OUT	16(79 – 64)
		CZ ALU OUT	2(81 – 80)
		RFA3 Value	3(84 – 82)
PIPE5		Instruction	16(15 – 0)
		PC	16(31 – 16)
		DATA A1	16(47 – 32)
		DATA A2	16(63 – 48)
		ALU OUT	16(79 – 64)
		CZ ALU OUT	2(81 – 80)
		MEM DOUT	16(97 – 82)
		Sig_1b_sub_in	3(100 – 98)



- **All these MUXs have same select signal**
- **Transmission of control bits**
- **Control bits are directly implemented as conditions in the if-else clauses of the code**

HAZARD DETECTION AND MITIGATION:

- **R0 HAZARD:**
  - As per our understanding of the problem statement, we are considering that the programmer is highly skilled and given specifications of our design, he/she won't run instructions which are 'R0 hazard' specific. i.e. none of the instructions will have the destination address as "000", which is the address for PC.
  - The above mentioned Hazard is very frequent for such a design in which the PC is in the register file itself.
- **Dependencies without R0:**
  - If the operand register for new instructions is the same as that of the destination register of previous instruction, let it be in the execution stage or memory stage or writeback stage, there will be errors and we can use forwarding directly from those stages. One thing to be noted is that in case of conditional execution like ADC, ADZ, NDC, NDZ if the respective flags are not set then there is no need for forwarding even if the destination register of previous instructions are the same.
    - For all these forwarding we are using two muxes and the output is to be given to pipeline3 reg ports - data47_32 and data63_48.
    - For ADD, ADC, ADZ, NDU, NDC, NDZ ,ADI and other R-type instructions we are forwarding from ALU_1 and thus correcting IMMEDIATE dependencies.

- For ADD, ADC, ADZ, NDU, NDC, NDZ ,ADI and other R-type instructions we are also forwarding from 79-64 bits of pipeline register 4 and thus correcting LEVEL 2 dependencies.
- For LW we are forwarding from the mem_d0 LEVEL 2 dependencies.
- For JAL and JLR instructions , forwarding is the same as other R type instructions.

- We can't correct LW immediate dependencies and so we have to stall the pipeline for 1 cycle so that load reaches memory stage and add or nand is in register-read stage and then we do the forwarding from mem_do.

- **LM-SM hazards:**

In case of LM-SM instruction we have to stall pipeline register1, Pipeline register 2 and stop updating the PC .

- LM:
  - Subtractor is turned ON when LM is in Pipe4
  - Wrp1, wrp2 (write enable for pipeline registers) takes input 0 when LM in pipe3.
  - PC updation stops when LM is in Pipe3.
  - Simultaneously transfer Shifter output into pipe4 when LM is in pipe3
  - When 8 cycles complete:
    - We are checking the above condition by checking sig_pipe4[84-82] != "000".
  - After 8 cycles PC updation starts

- Subtractor is turned off, and bits 84-82 of pipeline register 3 are selected as “111”.
- wrp<=1, wrp2<= 1
- All muxes related to LM-SM, start passing normal signals for other instructions.

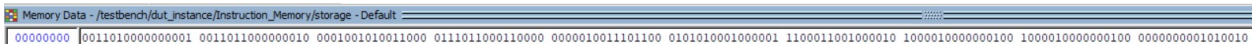
- SM:
  - Subtractor is turned ON when SM is in pipe2.
  - Wrp1 , wrp2(write enable for pipeline registers) are set 0 when SM is in pipe 3
  - PC updation stops when SM is in pipe3
  - Simultaneously transfer Shifter output into pipe3 when SM is in pipe2
  - When 8 cycle completes:
    - We are checking the above condition by checking, sig_pipe5[100-98] != “000”
  - After 8 cycles PC update starts.
  - Subtractor is turned OFF
  - wrp<= 1, wrp2<= 1(enable write enable for pipeline registers)
  - All muxes related to LM-SM start passing Normal signals for other instructions.

RTL SIMULATION RESULTS and TRANSCRIPT:

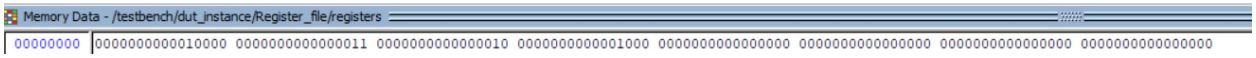
Program we used for checking -

```
storage(0) <= "0011010000000001";   --load 1 in r2
storage(1) <= "0011011000000010";   -- load 2 inr3
storage(2) <= "0001001010011000";   -- ADA R1 = R2 + R3 =3
storage(3) <= "0111011000110000";  -- sm memory(2)
storage(4) <= "0000010011101100"; -- ADI R2 = R3 + 24
storage(5) <= "0101010001000001"; -- SW R2 = 24 , mem_add = r1(0) + imm6(1), now storage(1) = 4;
storage(6) <= "1100011001000010"; -- jal pc = pc + 2*imm9
storage(7) <= "1000010000000100"; -- beq
storage(8) <= "1000010000000100"; -- beq
storage(9) <="0000000001010010"; -- Adi
```

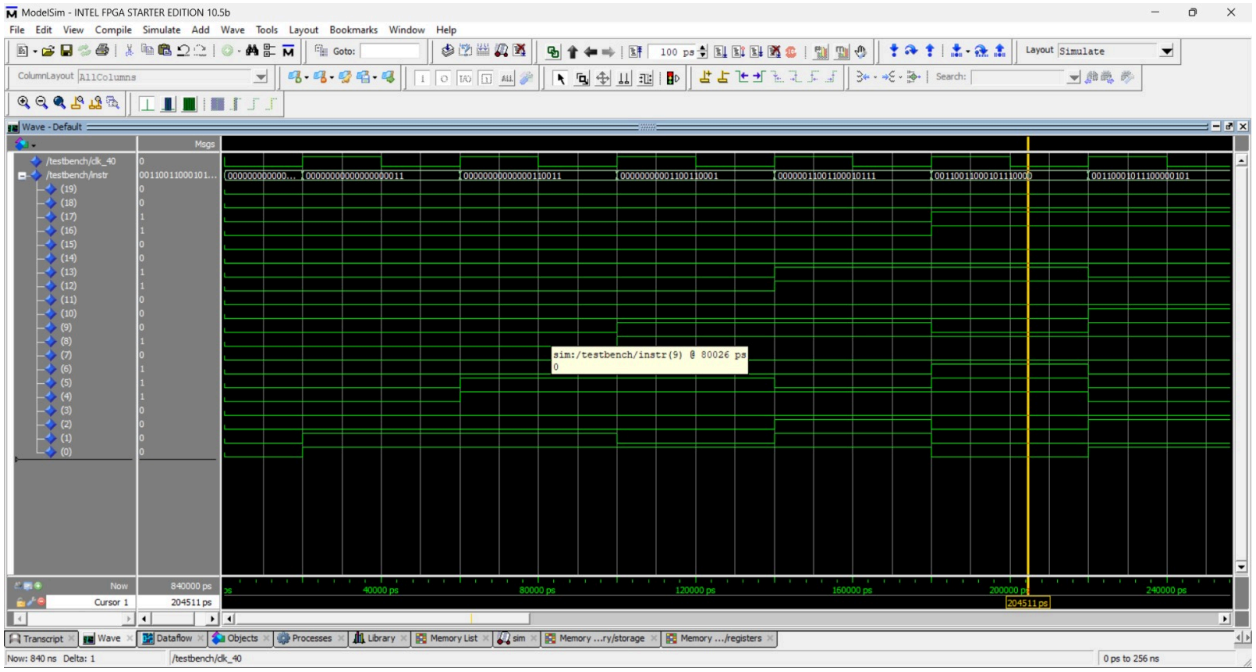
Instruction Memory



Register File after execution -



RTL Simulation showing the opcodes in all pipeline registers -



Work Distribution:

In this section, we outline the roles and responsibilities undertaken by each team member in the execution of the project. The allocation of tasks was done to expedite the progress of the project. The following provides a transparent overview of the work distribution:

Task allocation:

Along the names of the team members are the components/areas on which they have partially/ fully worked on.

- Saarthak Krishan: ALU, Pipeline Registers, Instruction and Data Memory, Final Pipeline Integration
- Aniket Patel: Register File ,Datapath, CZ Register, Pipeline Registers, Final Pipeline Integration
- Sanat Kumar Agrawal: Register File, SE_6, SE_9, Instruction and Data Memory, Final Pipeline Integration
- Naresh Kumar: Adder, 16-bit register, Documentation and report making, Final Pipeline Integration

Apart from making components individually, each team member has contributed in debugging and state minimization of the FSM and overall code, as well as pen-paper design of datapath and components.