BUILDING A BASIC RV32IM PIPELINE

PART 03

Group 03

E/18/058 - De Alwis K. K. M.
E/18/170 - Karunarathna W. K.
E/18/203 - Madhusanka K. G. A. S.

1. INSTRUCTION FETCH STAGE

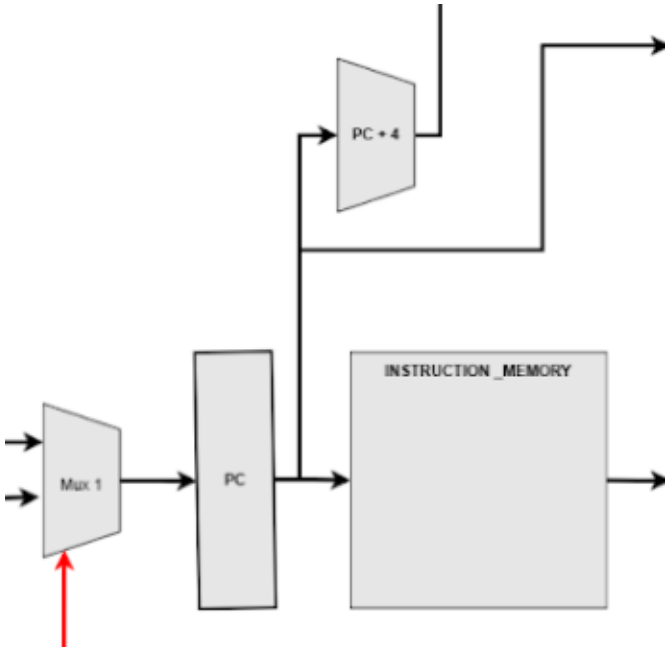

Figure 1: Instruction fetch stage

Modules in instruction fetch stage:

- Instruction fetch module
- Instruction cache module
- Instruction memory module

1.1 Instruction fetch module

The instruction fetch module is a combination of PC, PC+4 adder and PC mux (Mux1) module in the datapath.

I/O signals

Inputs:

CLK : 1 bit signal. Clock signal
RESET : 1 bit input. Reset signal. Used to Reset the PC
Instruction_mem_busywait : 1 bit input. Busywait signal from instruction memory
Data_mem_busywait : 1 bit input. Busywait signal from the cache memory
PC_mux_select : 1 bit input . Select whether to jump/branch or use PC value.
Jump_Branch_PC : 32 bit input - modified PC for branch or jump instructions

Outputs:

PC : 32 bit output. PC value for instruction cache.
PC+4 : 32 bit output. PC+4 value

Implementation and Timing

PC is held whenever data memory or instruction memory suffers a miss, therefore or operation is done between those Instruction_mem_busywait and data_memory_busywait.

When RESET, the PC value is set to -4 so that the CPU will restart from the next positive clock edge as PC = 0.

Every positive clock edge, PC is updated to PC+4 value from the adder when busywait is de-asserted. For the adder, we introduced #2 time unit delay. This is the same delay of the addition operation of the main ALU.

jump_branch_signal is used to select between modified PC from 3rd stage and typical PC+4 value. PC register write is incurred #2 delay.

1.2 Instruction cache module

The instruction cache module has a cache control module within it.

I/O signals

Inputs:

clk : clock signal
reset : 1 bit input. Reset signal.
imem_readblock : 127 bits of Read data from instruction memory
mem_BusyWait : busywait signal from instruction memory
icache_address : 10-bit input bus for instruction address

Outputs:

instruction : 32 bit instruction output from the cache
ichace_busywait : busywait signal of instruction cache
imem_read : read signal for instruction memory
imem_address : address of the required block to the cache

Implementation & Timing

Our Instruction cache can store 4 instructions per block. After 4 instructions cache holds the CPU and goes to the instruction memory and fetches the next four instructions from the instruction memory.

Cache has a total of 8 blocks, each block is 128 bits and byte addressed. Since cache fetches 4 bytes per time least significant 2 bits of the PC are ignored. The offset is 2 bits wide so that we can

differentiate between 4 instructions total. The index is 3 bits wide so that a total of 8 blocks can be stored within the cache. Therefore TAG is 25 bits wide.

Whenever a new PC value is received the PC value is decoded according to index, offset, and TAG. Here we insert a check to disregard -4 value at the very beginning when the CPU resets. This decode costs #1 time unit.

Then the TAG matching and hit state resolving occur asynchronously. This costs #5 unit times. Overall #6 units delay for instruction cache.

While hit status resolving data extraction from the relevant block happens with #1 time unit delay. Hit status resolving and data extraction happen simultaneously.

Cache memory will flag a busywait as soon as a new PC comes, to signal the CPU that the cache is busy fetching the instruction.

If the current PC value is a hit, this Busywait signal will be deasserted in the following clock edge. The instruction is fetched before this clock edge. If it's not a hit cache will not deassert the busywait at this clock edge but a garbage instruction may already be fetched since it is asynchronous cache reading. This garbage instruction may be a register write or data memory write instruction. So this has a vulnerability that these garbage instructions can corrupt data in register file or data memory. This is where the "instrHit" signal is needed. When there's a miss the hit signal is deasserted. This is used as a signal in the register file and Data memory before doing any write operation to make sure the fetched instruction is not garbage.

When there's a miss, the cache controller module does the part to generate relevant control signals to get the required data from the instruction memory. After the required data is fetched from the memory that 127-bit block will be stored in the cache with TAG and setting VALID bit to 1. This register writing operation takes #2 time delay.

1.3 Instruction Memory Module

The instruction memory is a component that is responsible for storing and providing instructions to the instruction execution.

I/O signals

Inputs:

CLOCK_SIGNAL : Clock signal is used to synchronize the various operations that occur within the instruction memory, such as reading an instruction from he memory or writing an instruction to the memory.

ADDRESS : This input is a 32-bit binary value that represents the memory address of the instruction to be fetched. The address is typically generated by the program counter, which keeps track of the current instruction address and increments it after each instruction is fetched.

Outputs:

INSTRUCTION_OUT : This output is a 32-bit binary value that represents the instruction that was requested by the at the specified address.

Timing

We have added a #40 time unit delay for the instruction memory. Because normally, reading and writing to data memory can take a lot of time due to memory latency, data transfer rate limitations, data caching issues, and data dependencies. These factors can cause delays in the overall performance of the system and can impact the overall speed of the CPU.

Therefore, caches are used for instruction memory in order to improve the performance of the system by reducing the time it takes to access data and instructions.

2. INSTRUCTION DECODE / REGISTER FILE READ STAGE

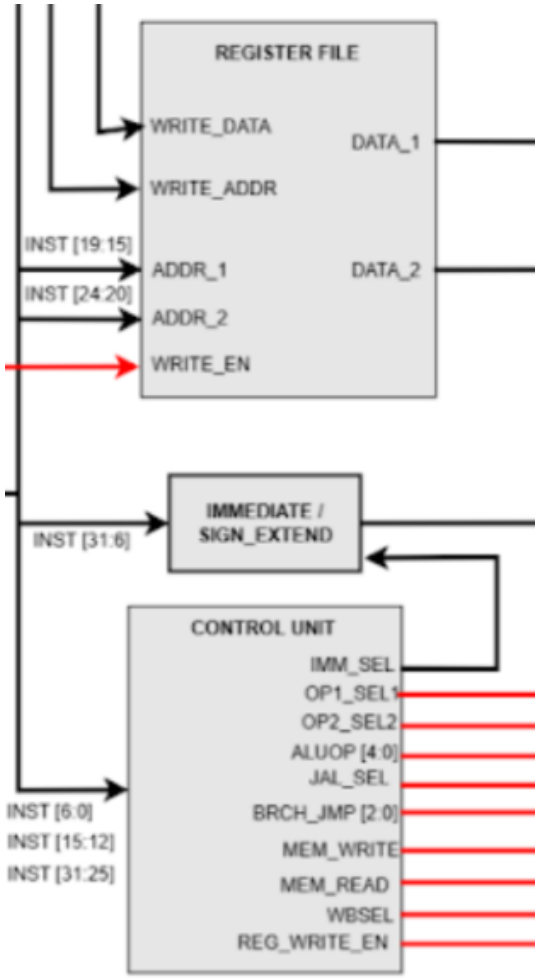

Figure 2: Instruction decode/ Register file read

2.1 Control Unit

The control unit generates the control signals needed to execute each instruction in the datapath. The control unit is responsible for coordinating the operation of all the components in the datapath, including the instruction fetch, decode, execute, memory access, and write-back stages.

The control unit determines the type of instruction being executed using the instruction opcode and creates the required control signals to carry out the instruction. Control signals are used to select the right inputs for each component in the datapath, as well as to enable or disable each component as needed.

I/O signals

Inputs:

Inst[6:0] : The opcode field in the instruction

Inst [31:25] : The funct7 field - here since we only consider the base instruction set and standard M extension, we only needed two bits from the funct7 field. Though we inserted the whole funct7 field as an input.

Inst [14:12] : The funct3 field - this field is needed to generate instruction-specific control signals for instructions in the same type.

Ex: to generate control signals into ADDI and SLTI. Both instructions have the same Opcode. The difference between these two instructions is in the funct3 field.

Outputs:

IMM_SEL : 3 bit signal. Mux 2 is the multiplexer that selects between immediates according to the instruction. IMM_SEL signal is used for that. There are 5 such immediates so we needed 3 bits to distinguish between them.
U → 011, J → 100, B → 000, L → 001, I → 010, S → 101

JAL_SEL : 1 bit signal. This signal goes to the multiplexer MUX4 which have 2 inputs namely ALU output and current pc + 4 value.
ALU output → 0, pc+4 → 1

OP1SEL : 1 bit signal. A select signal to MUX2, which selects between reg file output and PC value before giving them to the ALU for its operations.
PC → 0, DATA1 → 1

OP2SEL : 1 bit select signal for MUX3. Multiplexer MUX3 is used to select between reg file output (DATA_2) and immediate value which direct into the ALU module.
IMM → 0, DATA2 → 1

ALUOP : 5 bit signal for ALU. This signal tells the ALU which operation it should perform.

MEM_WRITE and MEM_READ : 1 bit signals. These signals are checked when accessing data memory to distinguish whether it's a write or read.

REG_WRITE_EN : 1 bit signal. This signal is checked whenever a write occurs to refile at the last stage of the pipeline. When this signal is set to one, the write occurs in the reg file.

WB_SEL : 1 bit signal. This is the select signal for multiplexer mux_5 which selects between ALU output and data memory output. After this, relevant data is direct to the register file.

BRANCH_JUMP : 2 bit signal. Two separate signals to distinguish whether the instruction is a jump or a branch. This signal is an input to the BRANCH_LOGIC unit. **JUMP → 01, BRANCH → 10**

Implementation and Timing

Opcode, funct3, and funct7 field 0th and 7th bits are taken as inputs to the control unit. According to the opcodes, common control signals were generated to each instruction type. Here , #1 time unit has been introduced for generating common control signals to represent the combination logic delays here.

To identify each type, an internal 4 bit control signal named instr_type was generated. Then an instruction-specific opcode was created by concatenating funct7 two bits, funct3, and instr_type. Using this instruction-specific opcode, instruction-specific control signals were generated.

Design Decisions

Generating basic control signals:

IMM_SEL, JAL_SEL, OP1SEL, OP2SEL, ALUOP, MEM_WRITE, MEM_READ, REG_WRITE_EN, WB_SEL, BRANCH_JUMP are identified as basic control signals These signals are unique to a particular OPCODE. Therefore these control signals are generated just by comparing opcodes.

Generating instruction specific control signals:

To generate instruction specific ALUop, first control unit resolve which type of instruction it is (to identify type we used instr_type signal within the control unit), and then depending on the funct3 and funct7 2 bits it generates 5-bit ALUop.

Load and store instructions:

The control unit generates the same control signals for all Load instructions and the same control signals for all Store instructions. Whether the instruction is a full word, half word, byte is considered in the memory access stage using the funct3 field. The following table consists of all the control unit signals for each specific instruction.

TABLE 1 : INSTRUCTIONS AND RELATED CONTROL SIGNALS

INSTRUCTION	PC_SEL	IMM_SEL	JAL_SEL	OP1SEL	OP2SEL	ALUOP	MEM_WRITE	MEM_READ	REG_WRITE_EN	WB_SEL	BRANCH_JUMP
LUI	PC + 4	011	0	*	IMM	FRWD	0	0	1	0	00
AUPIC	PC + 4	011	0	PC	IMM	ADD	0	0	1	0	00
JAL	ALU	100	1	PC	IMM	ADD	0	0	1	0	01
JALR	ALU	100	1	PC	IMM	ADD	0	0	1	0	01
BEQ	PC + 4 / ALU	000	0	PC	IMM	SUB	0	0	0	*	10
BNE	PC + 4 / ALU	000	0	PC	IMM	SUB	0	0	0	*	10
BLT	PC + 4 / ALU	000	0	PC	IMM	SUB	0	0	0	*	10
BGE	PC + 4 / ALU	000	0	PC	IMM	SUB	0	0	0	*	10
BLTU	PC + 4 / ALU	000	0	PC	IMM	SUB	0	0	0	*	10
BGEU	PC + 4 / ALU	000	0	PC	IMM	SUB	0	0	0	*	10
LB	PC + 4	001	0	DATA1	IMM	ADD	0	1	1	1	00
LH	PC + 4	001	0	DATA1	IMM	ADD	0	1	1	1	00
LW	PC + 4	001	0	DATA1	IMM	ADD	0	1	1	1	00
LBU	PC + 4	001	0	DATA1	IMM	ADD	0	1	1	1	00
LHU	PC + 4	001	0	DATA1	IMM	ADD	0	1	1	1	00
SB	PC + 4	101	0	DATA1	IMM	ADD	1	0	0	*	00
SH	PC + 4	101	0	DATA1	IMM	ADD	1	0	0	*	00
SW	PC + 4	101	0	DATA1	IMM	ADD	1	0	0	*	00
ADDI	PC + 4	010	0	DATA1	IMM	ADD	0	0	1	0	00
SLTI	PC + 4	010	0	DATA1	IMM	SLT	0	0	1	0	00
SLTIU	PC + 4	010	0	DATA1	IMM	SLTU	0	0	1	0	00

XORI	PC + 4	010	0	DATA1	IMM	XOR	0	0	1	0	00
ORI	PC + 4	010	0	DATA1	IMM	OR	0	0	1	0	00
ANDI	PC + 4	010	0	DATA1	IMM	AND	0	0	1	0	00
SLLI	PC + 4	010	0	DATA1	IMM	SLL	0	0	1	0	00
SRLI	PC + 4	010	0	DATA1	IMM	SRL	0	0	1	0	00
SRAI	PC + 4	010	0	DATA1	IMM	SRA	0	0	1	0	00
ADD	PC + 4	*	0	DATA1	DATA2	ADD	0	0	1	0	00
SUB	PC + 4	*	0	DATA1	DATA2	SUB	0	0	1	0	00
SLL	PC + 4	*	0	DATA1	DATA2	SLL	0	0	1	0	00
SLT	PC + 4	*	0	DATA1	DATA2	SLT	0	0	1	0	00
SLTU	PC + 4	*	0	DATA1	DATA2	SLTU	0	0	1	0	00
XOR	PC + 4	*	0	DATA1	DATA2	XOR	0	0	1	0	00
SRL	PC + 4	*	0	DATA1	DATA2	SRL	0	0	1	0	00
SRA	PC + 4	*	0	DATA1	DATA2	SRA	0	0	1	0	00
OR	PC + 4	*	0	DATA1	DATA2	OR	0	0	1	0	00
AND	PC + 4	*	0	DATA1	DATA2	AND	0	0	1	0	00
MUL	PC + 4	*	0	DATA1	DATA2	MUL	0	0	1	0	00
MULH	PC + 4	*	0	DATA1	DATA2	MULH	0	0	1	0	00
MULHSU	PC + 4	*	0	DATA1	DATA2	MULHSU	0	0	1	0	00
MULHU	PC + 4	*	0	DATA1	DATA2	MULHU	0	0	1	0	00
DIV	PC + 4	*	0	DATA1	DATA2	DIV	0	0	1	0	00
DIVU	PC + 4	*	0	DATA1	DATA2	DIVU	0	0	1	0	00
REM	PC + 4	*	0	DATA1	DATA2	REM	0	0	1	0	00
REMU	PC + 4	*	0	DATA1	DATA2	REMU	0	0	1	0	00

2.2 Register File

The register file is a hardware component in a CPU that stores a small amount of data called registers. In RISC-V architecture, the register file contains 32 32-bit registers. These registers can be used to store data, addresses, pointers, and other information that are used in instructions.

It is accessed at the pipeline's execute stage, when the instruction's operands are fetched from the register file and used in the ALU or other execution units.Since there are 32 registers, they are identified by a 5 bit binary number.

I/O signals

Inputs:

WRITE_EN : Write Enable Signal is used to enable the write operation. Only If this has activated data will be written to the registers.has one bit.

WRITE_DATA : This input is a 32-bit binary value that represents the data that needs to be written to the register file.

WRITE_ADDR : This input is a 5-bit binary value that represents the register number that needs to be written to.

ADDR_1 : 5-bit binary value that represents the register number of the first operand that needs to be read from the register file.

ADDR_2 : 5-bit binary value that represents the register number of the second operand that needs to be read from the register file.

Outputs:

DATA_1 : This output is a 32-bit binary value that represents the contents of the register specified by the Read ADDR_1 input.

DATA_2 : This output is also a 32-bit binary value that represents the contents of register specified by the Read Register 2 input.

Implementation and Timing

The register file has two read ports and one write port. The two read ports allow two instructions in the same cycle to read two different registers simultaneously, while the write port allows one instruction to write a result back to a register.

The register file provides quick and low-latency access, making it an appropriate location to store frequently needed data. The register file, on the other hand, has a limited capacity and cannot hold enormous amounts of data. Data must be saved in memory in such instances, which is accessed at a slower pace than the register file.

Register file takes a total of 4 time units.

2.3 Immediate Sign Extend Unit

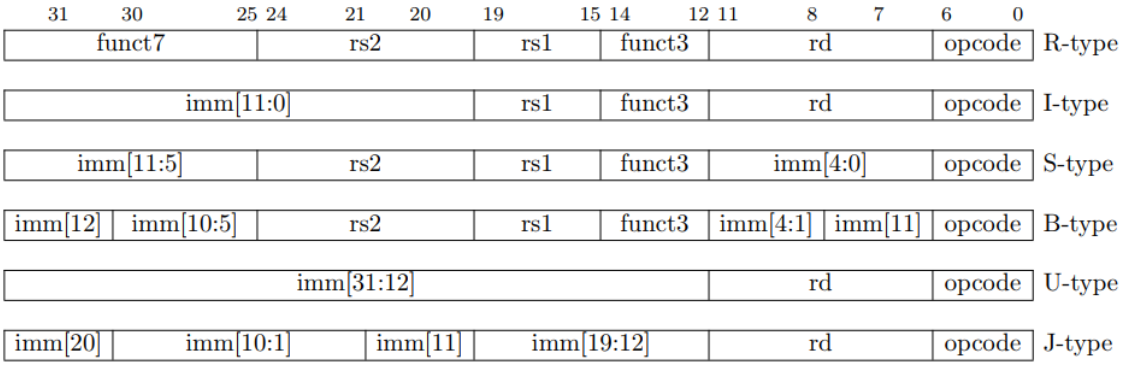

Figure 3: RISC-V Base Instruction Formats

According to the above figure we can see that in RISC-V each instruction format has a different bit encoding pattern. Therefore when we create the 32-bit immediate value we have to rearrange the bits according to the instruction format. Then, to make it 32-bits long we need to do the corresponding sign bit/ zero bit extension according to the RISC-V mentioned.

Immediate sign extend unit does the above explained process. It takes the 32-bit long instruction as the input and produces the 32-bit U_IMM, J_IMM, I_IMM, S_IMM, B_IMM which are respectively U type immediate, J type immediate, I type immediate, S type immediate and B type immediate values. After that, the output of this unit passed to the MUX3 in datapath.

Due to the propagation delays of the hardware components, #1 time unit artificial delay was added to the Immediate Sign extend unit.

3. EXECUTE STAGE

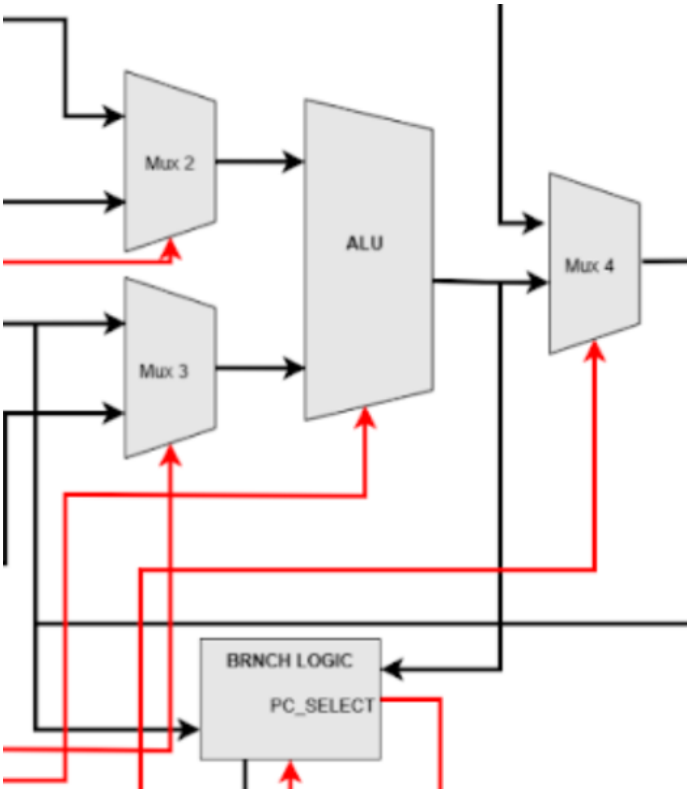

Figure 3: Execute

3.1 Arithmetic And Logic Unit

The Arithmetic and Logic Unit (ALU) is a key component of a CPU's datapath, responsible for performing arithmetic and logical operations on data. In a RISC-V implementation, the ALU is typically a combinational logic circuit that operates on two input operands and produces a single output result based on a control signal provided by the control unit.

The operands are first passed through a multiplexer that selects which of the operands to use as the input. The output of the ALU is then passed to the next stage of the datapath, typically the memory stage or the write-back stage.

I/O Signals

Inputs:

MUX2_OUT	:	Output of the multiplexer 2 on datapath. It selects either the PC value or the DATA1 coming from the register file and the selected output passed to the ALU as an input.
MUX3_OUT	:	Output of the multiplexer 3 on datapath. It selects either the Immediate/Sign Extend value or the DATA2 coming from the register file and the selected output passed to the ALU as an input.
ALUOP	:	5 bit control signal coming from the control unit to select which ALU operation result to select as the ALU output.

Outputs:

ALU_OUT : 32-bit ALU result as the output for the selected operation.

All the ALU operations are done asynchronously. There are 19 different ALU operations. Because of that we have used a 5 bit ALU opcode to select the ALU output. For the ADD and SUB operation we used 2 time unit delay. For the Multiplication, Division and Remainder operations we use 3 time unit delay because of the complexity of their operation compared to ADD and SUB. For the other operations we introduced 1 time unit delay.

Below table shows all the ALU operations, their corresponding opcodes and the respective delays assigned to it.

TABLE 2 : ALU OPERATIONS AND RELATED OPCODES

ALU Operation	Assigned OPCODE	Delay (# of time units)
ADD	00000	2
SUB	00001	2
AND	00010	1
OR	00011	1
XOR	00100	1
SLL	00101	1
SRL	00110	1
SRA	00111	1
MUL	01000	3
MULH	01001	3
MULHU	01010	3
MULHSU	01011	3
DIV	01100	3
DIVU	01101	3
REM	01110	3
REMU	01111	3
SLT	10000	1
SLTU	10001	1
FORWARD	10010	1

3.2 MUX 2

Determines whether the PC or DATA1 value should be sent to the ALU

I/O Signals

Inputs:

DATA1, PC

OP1_SEL - 1 bit signal to determine the output of the mux

- 0-> PC

- 1-> DATA1

Outputs:

MUX2_OUT - selected value from PC and DATA1

3.3 MUX 3

Determines whether the IMM(immediate) or DATA2 value should be sent to the ALU.

I/O Signals

Inputs:

DATA2, IMM

OP2_SEL - 1 bit signal to determine the output of the mux

- 0-> IMM

- 1-> DATA2

Outputs:

MUX3_OUT - selected value from IMM and DATA2

3.4 MUX 4

Selects the output value from ALUOUT and PC_4.

I/O Signals

Inputs:

ALUOUT - Output from the ALU

PC_4 - Incremented PC value

JAL_SEL - 1 bit signal to determine the output of the mux

- 0-> ALUOUT

- 1-> PC_4

Outputs:

MUX4_OUT - selected value from ALUOUT and PC_4

3.5 Brach and Logic Unit

I/O Signals

Inputs:

PC
RESET
Branch_imm
Alu_Jump_imm
func3
zero_signal
sign_bit_signal
sltu_bit_signal

Outputs:

branch_jump_mux_signal
Branch_jump_PC_OUT

4. MEMORY ACCESS STAGE

Figure 4: Execute

4.1 Data Memory

The data memory is a component that is responsible for storing and retrieving data. It is a block of memory that can be accessed using memory addresses. Data memory is accessed using memory access instructions.These instructions specify the memory address that the data should be read from or written to. The data memory then retrieves or stores the data at the specified memory address.

I/O Signals

Inputs:

WRITE_DATA : This input is used to provide the data that needs to be written into the data memory.

ADDRESS : This input provides the memory address where the data needs to be Written.

WRITE_EN : The write enable input is also typically a binary value that controls whether a write operation should be performed on the data memory. When the write enable input is set to high, the data memory will allow write operations to occur.

READ_EN : The read enable input is typically a binary value that controls whether a read operation should be performed on the data memory. When the read

enable input is set to high, the data memory will allow read operations to occur.

Outputs:

READ_DATA : This output provides the data that is read from the data memory.

Implementation and Timing

The Data Memory is byte addressable and it contains 256 registers of 127 bit wide. Since data memory and cache communicate in blocks each read or write request from cache involves a 127-bit block. The block address is 27 bit wide. Data memory fetches data byte by byte until the whole block is fetched. Each byte fetch costs #40-time units which means for the whole block to be fetched there will be a total of 40 x 16 time units delay. Also when writing to the data memory each byte write costs #40 time units. Therefore writing to data memory also takes 40x16 time unit delay.

5. WRITE BACK STAGE

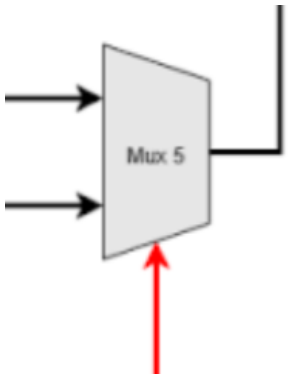

Figure 5: Write back

5.1 MUX 5

I/O Signals

Inputs:

ALUOUT, DATA_MEM

WB_SEL - 1 bit signal to determine the output of the mux

- 0-> ALUOUT

- 1-> DATA_MEM

Outputs:

MUX5_OUT - selected value from IMM and DATA2

6. PIPELINE REGISTERS

All together there are 4 pipeline registers. When the reset signal is high then all the 4 pipeline registers will be cleared. In every positive clock edge if the reset and busywait signals are low then the pipeline register output ports will be updated with the pipeline input ports and this process will take #1 time unit delay. If the busywait is high then the pipeline registers will be stalled until the busywait is released.

Pipeline Datapath

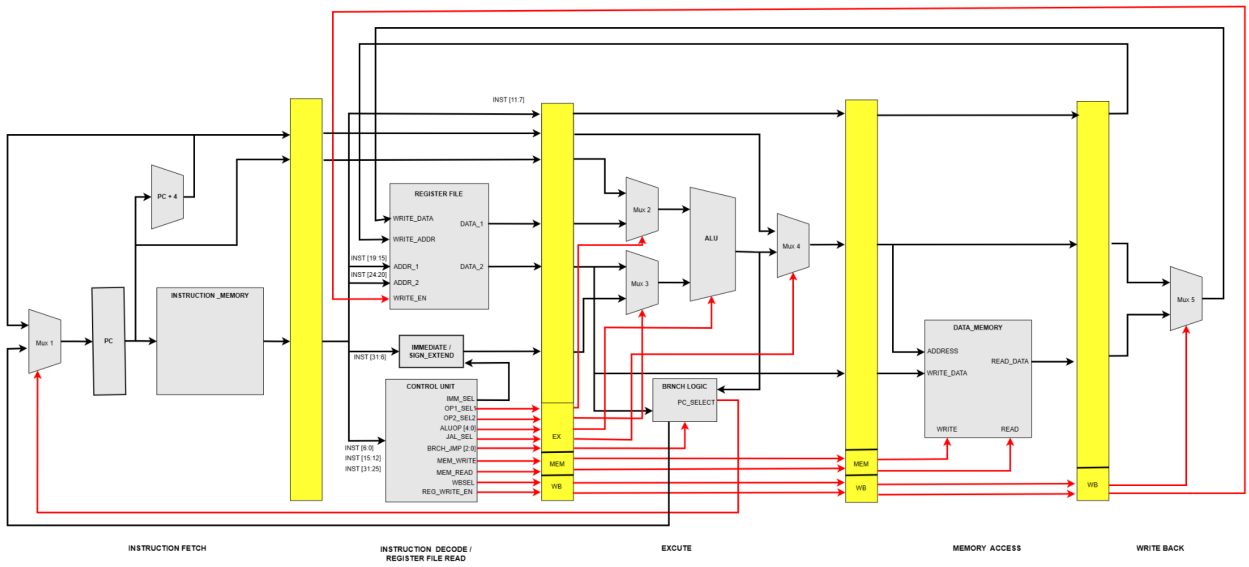

Figure 4 : Datapath Diagram

Full image of the datapath can be seen here. Datapath.png