CO502 - Advanced Computer Architecture
Lab 01 - Part 01
Group 04

Group Members

E/18/077 - Nipun Dharmarathne
E/18/397 - Shamod Wijerathne
E/18/402 - Chathura Wimalasiri

Activity 1

Instruction types and their encoding formats

There are 6 types of instructions in RISC-V architecture:

1. R-type (register-type) instructions: These instructions operate on registers and have three register operands. Examples include ADD, SUB, AND, OR, XOR, SLL, SRL, and SRA.

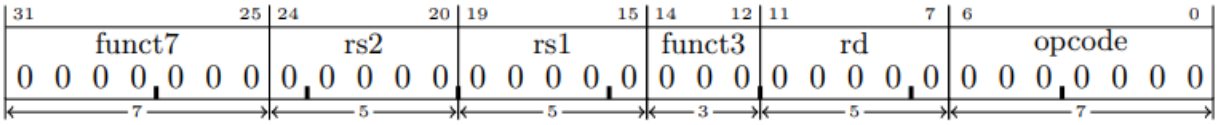

Figure 01: R-type instruction encoding format

Here, the funct7, funct3, and opcode fields identify the specific operation, while the rs1, rs2, and rd fields specify the source and destination registers.

The instruction performs the operation specified by funct3 on the values stored in the two source registers rs1 and rs2. The result of the operation is stored in the destination register rd. The specific operation performed by the instruction depends on the function code funct3.

2. I-type (immediate-type) instructions: These instructions have one immediate value and one register operand. Examples include ADDI, SLTI, ANDI, ORI, XORI, SLLI, SRLI, and SRAI.

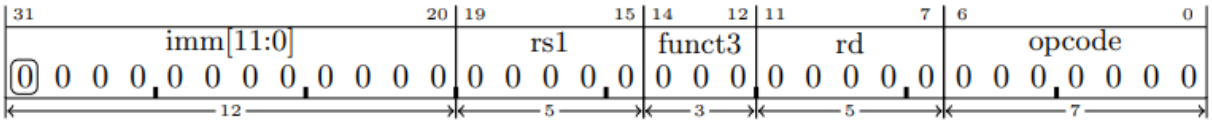

Figure 02: I-type instruction encoding format

Here, imm is a 12-bit immediate value, rs1 is the source register, funct3 is a function code that identifies the specific I-type instruction, rd is the destination register where the result is stored, and opcode is the operation code that identifies the type of instruction.

The instruction performs the operation specified by funct3 on the value stored in the source register rs1 and the immediate value imm. The result of the operation is stored in the destination register rd. The specific operation performed by the instruction depends on the function code funct3.

3. S-type (store-type) instructions: These instructions store a value from a register into memory with an offset. Examples include SB, SH, and SW.

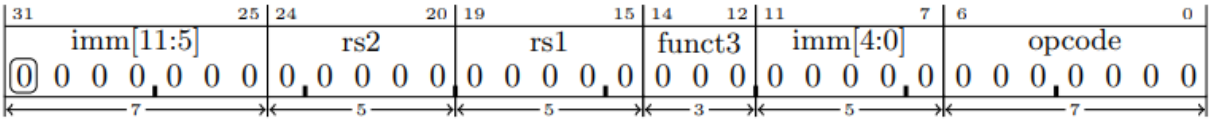

Figure 03: S-type instruction encoding format

Here, imm is a 12-bit immediate value used as an offset to access memory, rs1 is the base register used to calculate the memory address, rs2 is the source register whose value needs to be stored in memory, and funct3 identifies the specific store operation.

When executed, the S-type instruction calculates the memory address by adding the 12-bit immediate value imm to the value of register rs1. It then stores the value of register rs2 into the memory location specified by the calculated address. The specific store operation is determined by the funct3 field, which can be used to specify the byte size of the store operation, and can have values such as 000 for storing a byte, 001 for storing a halfword (2 bytes), or 010 for storing a word (4 bytes).

4. U-type (upper immediate) instructions: These instructions load a 20-bit immediate value into a register. Examples include LUI and AUIPC.

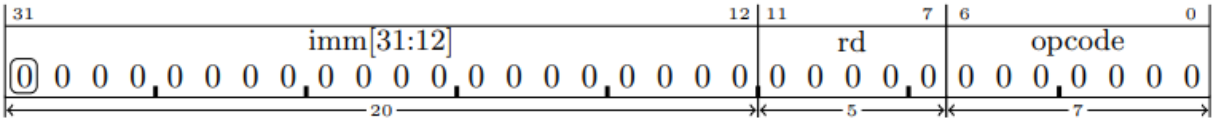

Figure 04: U-type instruction encoding format

Here, imm is a 20-bit immediate value, rd is the destination register where the immediate value is loaded, and opcode is the operation code that identifies the specific U-type instruction.

When executed, the U-type instruction loads the 20-bit immediate value imm into register rd. The immediate value is zero-extended to 32 bits before being stored in the register. Since the immediate value is 20 bits, it cannot represent the entire 32-bit address space. To load a full 32-bit address, the U-type instruction is typically combined with a lower 12-bit immediate value specified in an I-type instruction, using a special encoding called the UJ-type (unconditional jump-type) instruction.

5. B-type (branch-type) instructions: These instructions perform conditional branches based on the comparison of two register values. Examples include BEQ, BNE, BLT, BGE, BLTU, and BGEU.

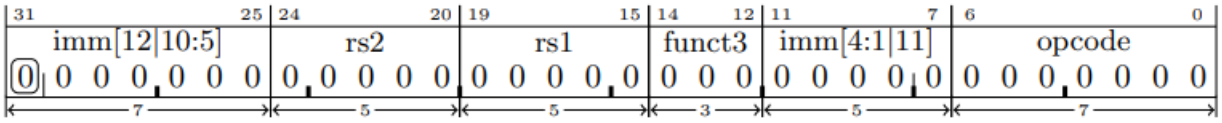

Figure 05: B-type instruction encoding format

Here, rs1 and rs2 are the two source registers, funct3 is a function code that identifies the specific B-type instruction, opcode is the operation code that identifies the type of instruction, and imm is a 13-bit immediate value that encodes the branch offset.

The immediate value is constructed from three fields: the 1-bit imm[12] field, which is used to construct the top bit of the 13-bit immediate value; the 6-bit imm[10:5] field, which is used to construct bits 4-9 of the 13-bit immediate value; and the 4-bit imm[4:1|11] field, which is used to construct bits 0-3 and 12 of the 13-bit immediate value.
The rs1 and rs2 registers are compared based on the specific function code funct3. If the comparison is true, then the program counter is updated to the target address computed from

the relative branch offset. If the comparison is false, the program counter is incremented to the next sequential instruction.

6. J-type (jump-type) instructions: These instructions perform unconditional jumps and jump with a link (i.e., save the address of the next instruction to be executed). Examples include JAL and JALR.

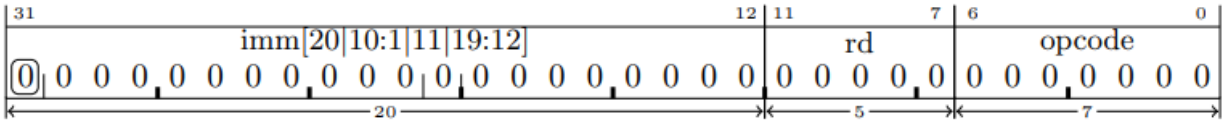

Figure 06: J-type instruction encoding format

Here, rd is the destination register, which is used to store the return address of the jump instruction. The immediate value is added to the current program counter (PC) value to compute the target address. The immediate value is then shifted left by 1 bit to adjust for the fact that RISC-V instructions are 32 bits long and the target address must be aligned to a 32-bit boundary.

After the jump instruction is executed, the PC is set to the target address computed from the immediate value, with the least-significant bit set to zero to ensure that the target address is aligned to a 32-bit boundary. The rd register is used to store the return address of the jump instruction, which is the address of the instruction following the jump instruction.

Instructions and their opcodes

R-Type instructions

Instruction	Description	Opcode	func3	func7
ADD	ADD the value in rs1 and rs2 value and put it into rd	0110011	000	0000000
SUB	SUBSTRACT the value in rs1 and rs2 value and put it into rd	0110011	000	0100000
SLL	Shiflt Left Logical (rs1 value by rs2 amount and put it to rd)	0110011	001	0000000
SLT	Set less than (if rs1 value less than rs2 value then put 1 to the rd register else 0 to the rd register)	0110011	010	0000000
SLTU	Set less than Unsigned (if rs1 value less than rs2 value then put 1 to the rd register else 0 to the rd register)	0110011	011	0000000
XOR	XOR the value in rs1 and rs2 value and put it into rd	0110011	100	0000000
SRL	Shift Right Logical (rs1 value by rs2 amount and put it to rd)	0110011	101	0000000
SRA	Shift Right Arithmatic (rs1 value by rs2 amount and put it to rd)	0110011	101	0100000
OR	OR the value in rs1 and rs2 value and put it into rd	0110011	110	0000000
AND	AND the value in rs1 and rs2 value and put it into rd	0110011	111	0000000
MUL	Multiplication	0110011	000	0111011
MULH	Returns upper 32-bits of signed x signed (use rs1 value and the rs2 value and put the answer to the rd register)	0110011	001	0111011
MULHSU	Returns upper 32-bits of signed x unsigned (use rs1 value and the rs2 value and put the answer to the rd register)	0110011	010	0111011
MULHU	Returns upper 32-bits of unsigned x unsigned (use rs1 value and the rs2 value and put the answer to the rd register)	0110011	011	0111011

DIV	Signed Interger division (use rs1 value and the rs2 value and put the answer to the rd register)	0110011	100	0111011
REM	Signed remainder of integer division (use rs1 value and the rs2 value and put the answer to the rd register)	0110011	101	0111011
REMU	Unsigned remainder of interger division (use rs1 value and the rs2 value and put the answer to the rd register)	0110011	111	0111011

I-Type instructions

Instruction	Description	Opcode	func3	func7
LB	Load Byte to rd register (signed extended)	0000011	000	
LH	Load 2 Bytes to rd register (signed extended)	0000011	001	
LW	Load Word to rd register (signed extended)	0000011	010	
LBU	Load Byte to rd register (zero extended)	0000011	100	
LHU	Load 2 Bytes to rd register (zero extended)	0000011	101	
ADDI	ADD immediate value and value of rs1 and put result to rd register	0010011	000	
SLLI	Shift Left Logical with Immediate (shift rs1 value by immediate amount)	0010011	001	
SLTI	if immediate value is less than value of rs1 put 1 to rd register otherwise put 0	0010011	010	
SLTIU	if immediate value is less than value of rs1 put 1 to rd register otherwise put 0 (unsigned)	0010011	011	
XORI	XOR immediate value and value of rs1 and put result to rd register	0010011	100	
SRLI	Shift Right Logical with Immediate (shift rs1 value by immediate amount)	0010011	101	
SRAI	Shift Right Arithmatic Immediate (shift rs1 value by immediate amount)	0010011	101	
ORI	OR immediate value and value of rs1 and put result to rd register	0010011	110	
ANDI	AND immediate value and value of rs1 and put result to rd register	0010011	111	
JALR	Jump and Link Register	1100111		

S-Type instructions

Instruction	Description	Opcode	func3	func7
SB	Store Byte rs2 reg value, base is in rs1 address and the offset taken from the immediate value	0100011	000	
SH	rs2 reg value, base is in rs1 address and the offset taken from the immediate value	0100011	001	
SW	rs2 reg value, base is in rs1 address and the offset taken from the immediate value	0100011	010	
SBU	Store unsigned Byte	0100011	100	
SHU	Store unsigned half word	0100011	101	

U-Type instructions

Instruction	Description	Opcode	func3	func7
AUIPC	Add upper immediate to PC and put to rd register	0010011		
LUI	Load Upper Immediate (puts the immediate value with 12 zeros at the end and put it into rd register)	0110111		

B-Type instructions

Instruction	Description	Opcode	func3	func7
BEQ	Branch if equal (if values in rs1 and rs2 are equal jump to the offset)	1100011	000	
BNE	Branch if not equal (if values in rs1 and rs2 are not equal jump to the offset)	1100011	001	
BLT	Branch if lower than (if values in rs1 < rs2 jump to the offset)	1100011	100	
BGE	Branch if grater than (if values in rs1 > rs2 jump to the offset)	1100011	101	
BLTU	Branch if lower than, unsigned (if values in rs1 < rs2 jump to the offset)	1100011	110	
BGEU	Branch greater than or equla, unsigned (if values in rs1 > rs2 jump to the offset)	1100011	111	

J-Type instructions and other instructions

Instruction	Description	Opcode	func3	func7
JAL	Jump and Link (Jumps to the address in rs1 and put the current PC to the rd register)	1101111		
FENCE	Fence - This to ensure all the operation before FENCE observed before operation after the Fence	0001111	000	
FENCE.I	Fence Instruction	0001111	001	

Activity 2

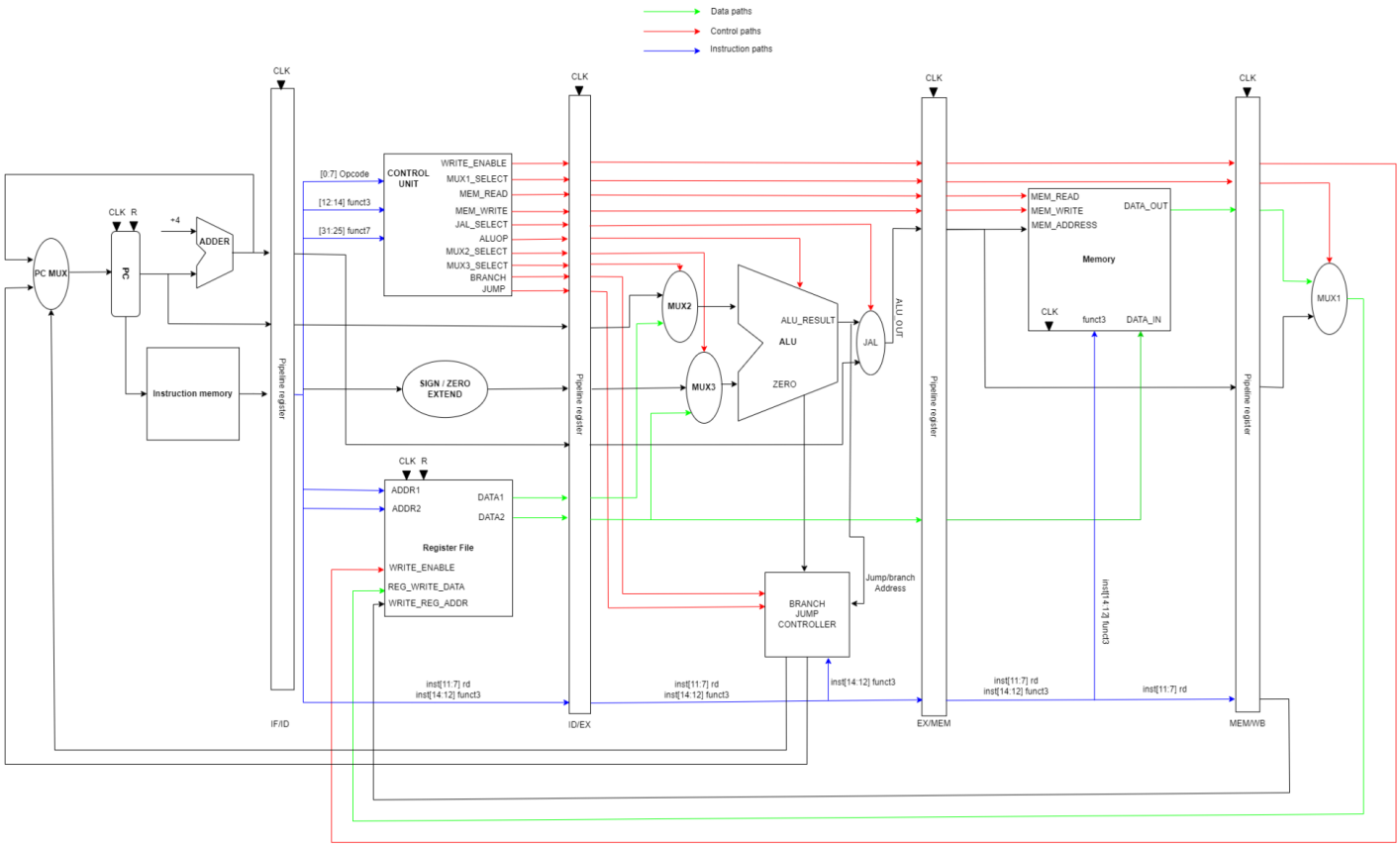

Figure 07: Pipeline diagram with datapath and control path

In case of not clear enough to observe above FIGURE 07 - [Pipeline diagram with datapath and control path]

Main units of the processor

1. Instruction Memory

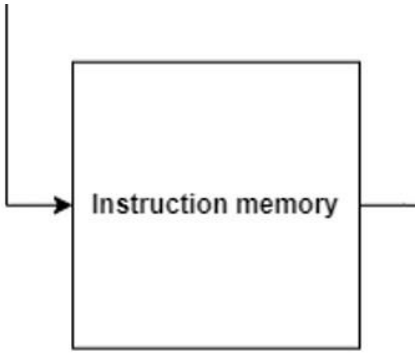

Figure 08: Instruction Memory

Inputs

1. PC - the memory address of the next instruction to be executed in a computer program.

Outputs

1. 32-bit Instruction

2. Control Unit

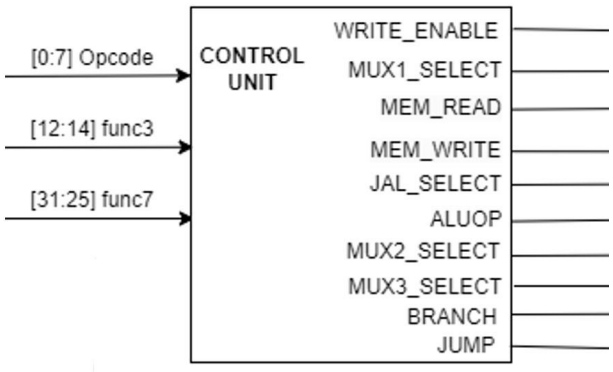

Figure 09: Control Unit

Inputs

1. opcode

   An opcode, short for "operation code," is a binary code that specifies the operation to be performed by a processor or microcontroller in order to execute a particular instruction.

2. func3

   func3 (short for function 3) is a field in the instruction format that specifies a subset of operations that can be performed by an instruction.

3. func7

   func7 (short for function 7) is a field in the instruction format that provides additional information about the operation to be performed by an instruction.

Outputs - Control signals

1. WRITE_ENABLE:

   1 bit signal. This is the control signal that determines whether data can be written into the register. When WRITE_ENABLE is high it enables the write operation to the register file.

2. MUX1_SELECT:

   1 bit signal. This is the select signal for MUX1 which is used to select between ALU_OUT and data memory output (DATA_OUT). Then the selected data is directed to the register file.

3. MEM_READ:

   1 bit signal. This is the control signal that enables the output data from the memory unit. When the MEM_READ signal is high, the memory unit is instructed to output data (DATA_OUT) from the specified address, which is provided on the address lines (MEM_ADDRESS).

4. MEM_WRITE:

   1 bit signal. This is the control signal that enables the input data to be written into the memory unit. When the MEM_WRITE signal is high, the memory unit is instructed to write the data provided on the data lines (DATA_IN) to the specified address on the address lines (MEM_ADDRESS).

5. JAL_SELECT:

   1 bit signal. This is the select signal for the JAL multiplexer which is used to select between ALU_RESULT and current PC+4 value.

6. ALUOP:

   5-bit signal for ALU. This signal specifies the arithmetic or logical operation to be performed by the ALU. It selects one of several operations that the ALU can perform, such as addition, subtraction, multiplication, bitwise AND, OR, XOR, and shift operations.

7. MUX2_SELECT:

   1 bit signal. This is the select signal for MUX2, which is used to select between register file output (DATA1) and PC value. Then the selected data is directed to the ALU for its operations.

8. MUX3_SELECT:

   1 bit signal. This is the select signal for MUX3, which is used to select between immediate value and register file output (DATA2). Then the selected data is directed to the ALU for its operations.

9. BRANCH and JUMP:

   1-bit signals. These signals are used in the BRANCH JUMP CONTROLLER in order to distinguish whether the instruction is a branch or a jump. The BRANCH signal is used to conditionally execute a set of instructions based on a particular condition. The JUMP signal is used to unconditionally jump to a specific instruction address.

3. Register FIle

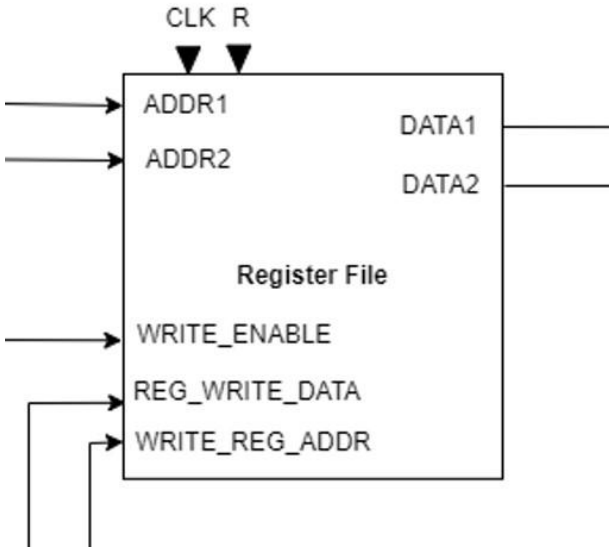

Figure 10: Register File

Inputs

1. CLK

   In a register file, the clock signal is used to trigger the read or write operations to the registers.

2. R

   The R signal is a read control signal that is used to enable a read operation from the register file.

3. ADDR1 and ADDR2

   These are address signals that are used to specify the register addresses for a read or write operation.

4. WRITE_ENABLE

   The write enable signal is used to enable a write operation to the register file.

5. REG_WRITE_DATA

   The register write data signal is used to specify the data value to be written to a register in the register file during a write operation.

6. WRITE_REG_ADDR

   The write register address signal is used to specify the address of the register to be written during a write operation.

Outputs

1. DATA1 and DATA2

   DATA1 and DATA2 are typically used in the context of an arithmetic or logic operation in a computer system. They represent the two data operands that are being used in the operation.

4. ALU

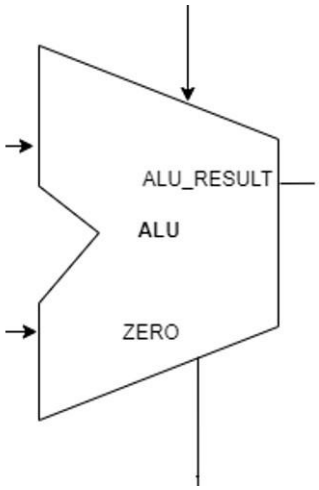

Figure 11: ALU

Inputs

1. MUX2 output

   The output of multiplexer 2 is one of the inputs to the ALU.

2. MUX3 output

   The output of multiplexer 3 is one of the inputs to the ALU.

3. ALUOP

   The ALU operation code is a control signal that specifies the operation to be performed by the ALU.

Outputs

1. ALU_RESULT

   The ALU result is the output of the ALU operation.

2. ZERO

   The ZERO flag is an output signal that indicates whether the result of the ALU operation is zero or not.

5. Branch-Jump Controller

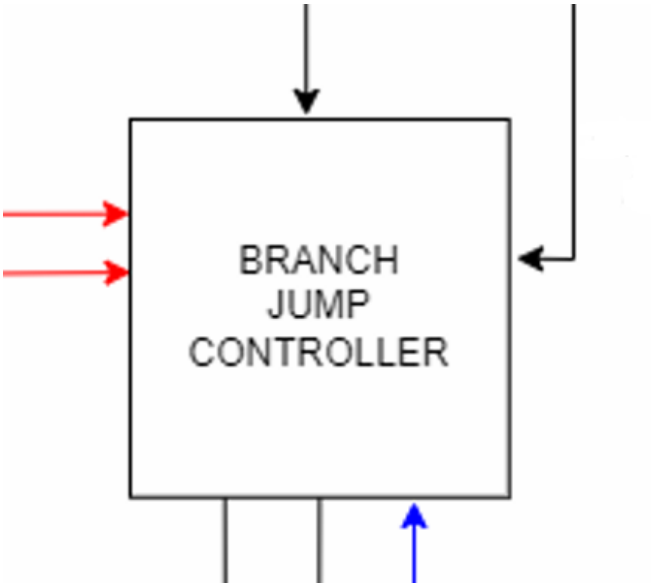

Figure 12: Branch Jump Controller

Inputs

1. BRANCH

   The BRANCH input is a signal that indicates whether a branch instruction has been executed.

2. JUMP

   The JUMP input is a signal that indicates whether a jump instruction has been executed.

3. ZERO

   The ZERO input is a signal that indicates whether the ALU has produced a zero output.

4. funct3

   Used to distinguish the condition of the branch instruction (Ex: Branch Less Than, Branch Greater Than or Equal, Branch Less Than Unsigned, etc)

5. Jump/Branch Address

   This contains the target address of the Branch/ Jump instruction that has been executed.

Outputs

1. Branch/ Jump signal

   This is used in PC MUX as the select bit to choose the next PC value

2. Branch/ Jump PC address

   This contains the target address of the Branch/ Jump instruction that has been executed.

6. Memory

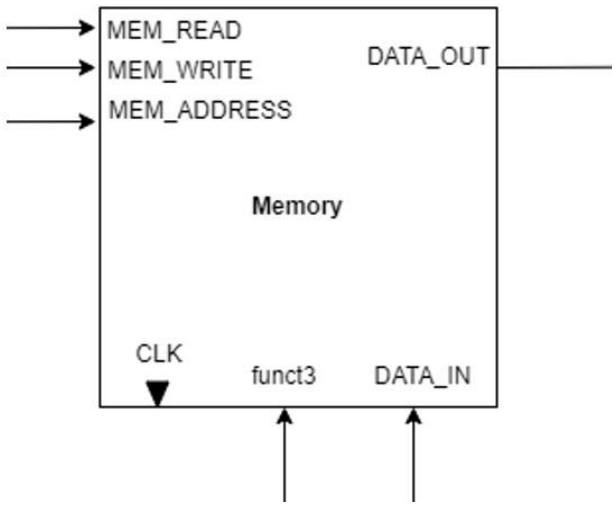

Figure 13: Memory

Inputs

1. CLK

   The clock signal is an input that synchronizes the operation of the memory with other components in the system.

2. MEM_READ

   The MEM_READ signal is an input that specifies whether the memory should read data from the specified address.

3. MEM_WRITE

   The MEM_WRITE signal is an input that specifies whether the memory should write data to the specified address.

4. MEM_ADDRESS

   The MEM_ADDRESS signal is an input that specifies the memory address to read from or write to.

5. funct3

   funct3 is an input that is used to decode the instruction and determine the operation to be performed on the memory.

6. DATA_IN

   The DATA_IN signal is an input that provides data to be written to the specified memory address.

Outputs

1. DATA_OUT

   The DATA_OUT signal is an output that provides the data read from the specified memory address.