CO502- Advanced computer architecture

Lab part 1- planning

E/20/024 – Ariyarathna D.B.S.

E/20/366 – Seneviratne A.P.B.P.

E/20/242- Malinga G.A.I.

Activity 1:

- Describe the instructions
- Clearly identify instruction encoding formats
- Clearly identify opcodes

Activity 2:

- Draw a pipeline diagram with datapath and control
- Clearly identify hardware units
- Clearly identify signals

Activity 3:

- List the tests you'll need to do to verify functionality of components

Instructions

Consider 32-bit words

Usage Template	Type	Description	Detailed Description
add rd,rs1,rs2	R	Add	rd rs1+rs2, pc pc+4
addi rd,rs1, imm	I	Add Immediate	rd rs1+immi, pc pc+4
and rd,rs1,rs2	R	And	rd rs1 rs2, pc pc+4
andi rd,rs1, imm	I	And Immediate	rd rs1 immi, pc pc+4
auipc rd, imm	U	Add Upper Immediate to PC	rd pc+immu, pc pc+4
beq rs1,rs2,pcrel13	B	Branch Equal	pc pc+((rs1==rs2)?immb:4)
bge rs1,rs2,pcrel13	B	Branch Greater or Equal	pc pc+((rs1>=rs2)?immb:4)

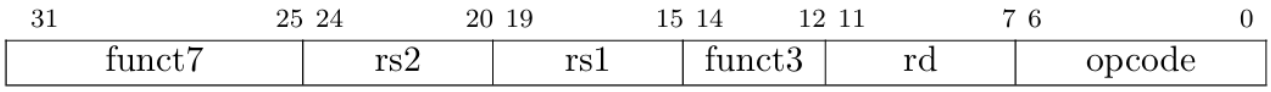
bgeu rs1,rs2,pcrel13	B	Branch Greater or Equal Unsigned	pc pc+((rs1>=rs2)?immb:4)
blt rs1,rs2,pcrel13	B	Branch Less Than	pc pc+((rs1<rs2)?immb:4)
Usage Template	Type	Description	Detailed Description
bltu rs1,rs2,pcrel13	B	Branch Less Than Unsigned	pc pc+((rs1<rs2)?immb:4)
bne rs1,rs2,pcrel13	B	Branch Not Equal	pc pc+((rs1!=rs2)?immb:4)
jal rd,pcrel21	J	Jump And Link	rd pc+4, pc pc+immj
jalr rd, imm(rs1)	I	Jump And Link Register	rd pc+4, pc (rs1+immi)& 1
lb rd, imm(rs1)	I	Load Byte	rd sx(m8(rs1+immi)), pc pc+4
lbu rd, imm(rs1)	I	Load Byte Unsigned	rd zx(m8(rs1+immi)), pc pc+4
lh rd, imm(rs1)	I	Load Halfword	rd sx(m16(rs1+immi)), pc pc+4
lhu rd, imm(rs1)	I	Load Halfword Unsigned	rd zx(m16(rs1+immi)), pc pc+4
lui rd, imm	U	Load Upper Immediate	rd immu, pc pc+4
lw rd, imm(rs1)	I	Load Word	rd sx(m32(rs1+immi)), pc pc+4
or rd,rs1,rs2	R	Or	rd rs1 rs2, pc pc+4
ori rd,rs1, imm	I	Or Immediate	rd rs1 immi, pc pc+4
sb rs2, imm(rs1)	S	Store Byte	m8(rs1+imms) rs2[7:0], pc pc+4
sh rs2, imm(rs1)	S	Store Halfword	m16(rs1+imms) rs2[15:0], pc pc+4
sll rd,rs1,rs2	R	Shift Left Logical	rd rs1<<(rs2%XLEN), pc pc+4
slli rd,rs1,shamt	I	Shift Left Logical Immediate	rd rs1<<shamti, pc pc+4
slt rd,rs1,rs2	R	Set Less Than	rd (rs1<rs2)?1:0, pc pc+4
slti rd,rs1, imm	I	Set Less Than Immediate	rd (rs1<immi)?1:0, pc pc+4
sltiu rd,rs1, imm	I	Set Less Than Immediate Unsigned	rd (rs1<immi)?1:0, pc pc+4
sltu rd,rs1,rs2	R	Set Less Than Unsigned	rd (rs1<rs2)?1:0, pc pc+4

sra rd,rs1,rs2	R	Shift Right Arithmetic	rd rs1>>(rs2%XLEN), pc pc+4
srai rd,rs1,shamt	I	Shift Right Arithmetic Immediate	rd rs1>>shamti, pc pc+4
srl rd,rs1,rs2	R	Shift Right Logical	rd rs1>>(rs2%XLEN), pc pc+4
srli rd,rs1,shamt	I	Shift Right Logical Immediate	rd rs1>>shamti, pc pc+4
sub rd,rs1,rs2	R	Subtract	rd rs1-rs2, pc pc+4
sw rs2, imm(rs1)	S	Store Word	m32(rs1+imms) rs2[31:0], pc pc+4
xor rd,rs1,rs2	R	Exclusive Or	rd rs1 rs2, pc pc+4
xori rd,rs1, imm	I	Exclusive Or Immediate	rd rs1 immi, pc pc+4

Instruction encoding formats

Mainly there are four base instruction formats in RISC-V architecture. They are R-type, I-type, S-type and U type. Furthermore, there are two variants of the instruction formats named B –type and J- type.

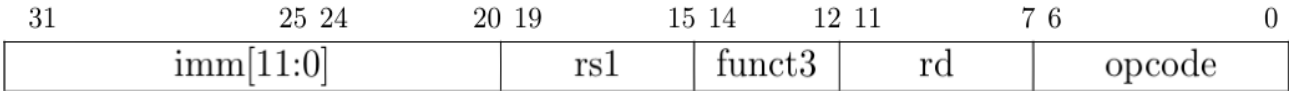
1. R-Type (Register Type)
   Used for arithmetic, logic, and shift instructions that operate on registers.
   - opcode (7 bits): Specifies the operation.
   - rd (5 bits): Destination register.
   - funct3 (3 bits): Function modifier for the operation.
   - rs1 (5 bits): First source register.
   - rs2 (5 bits): Second source register.
   - funct7 (7 bits): Additional function modifier
   - Ex- ADD, SUB, AND, OR, XOR, SLL, SRL, and SRA.



Instruction	Description	Opcode	func3	func7
ADD	ADD the value in rs1 and rs2, and put it into rd.	0110011	000	0000000
SUB	SUBTRACT the value in rs1 and rs2, and put it into rd.	0110011	000	0100000
SLL	Shift Left Logical (rs1 value by rs2 amount) and put it into rd.	0110011	001	0000000
SLT	Set Less Than (if rs1 < rs2, put 1 into rd; else 0).	0110011	010	0000000
SLTU	Set Less Than Unsigned (if rs1 < rs2, put 1 into rd; else 0).	0110011	011	0000000
XOR	XOR the value in rs1 and rs2, and put it into rd.	0110011	100	0000000
SRL	Shift Right Logical (rs1 value by rs2 amount) and put it into rd.	0110011	101	0000000
SRA	Shift Right Arithmetic (rs1 value by rs2 amount) and put it into rd.	0110011	101	0100000
OR	OR the value in rs1 and rs2, and put it into rd.	0110011	110	0000000
AND	AND the value in rs1 and rs2, and put it into rd.	0110011	111	0000000
MUL	Multiplication of rs1 and rs2, and put the result into rd.	0110011	000	0111011
MULH	Returns upper 32-bits of signed x signed (rs1 x rs2) into rd.	0110011	001	0111011

MULHSU	Returns upper 32-bits of signed x unsigned (rs1 x rs2) into rd.	0110011	010	0111011
MULHU	Returns upper 32-bits of unsigned x unsigned (rs1 x rs2) into rd.	0110011	011	0111011
DIV	Signed integer division (rs1 ÷ rs2) and put into rd.	0110011	100	0111011
REM	Signed remainder of integer division (rs1 ÷ rs2) into rd.	0110011	101	0111011
REMU	Unsigned remainder of integer division (rs1 ÷ rs2) into rd.	0110011	111	0111011

2. I – Type (Immediate Type)
   Used for instructions that involve an immediate value, such as arithmetic with constants, memory loads, and system calls.
   - `opcode` (7 bits): Specifies the operation.
   - `rd` (5 bits): Destination register.
   - `funct3` (3 bits): Function modifier.
   - `rs1` (5 bits): Source register.
   - `imm` (12 bits): Immediate value (sign-extended).
   - Ex- ADDI, SLTI, ANDI, ORI, XORI, SLLI, SRLI, and SRAI.



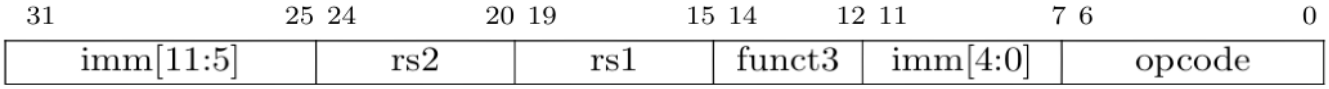
Instruction	Description	Opcode	Func3
LB	Load Byte to rd register (signed extended)	0000011	000
LH	Load 2 Bytes to rd register (signed extended)	0000011	001
LW	Load Word to rd register (signed extended)	0000011	010
LBU	Load Byte to rd register (zero extended)	0000011	100
LHU	Load 2 Bytes to rd register (zero extended)	0000011	101
ADDI	ADD immediate value and value of rs1, and put result to rd	0010011	000
SLLI	Shift Left Logical with Immediate (shift rs1 value by immediate amount)	0010011	001
SLTI	If immediate value < rs1, put 1 to rd; otherwise put 0	0010011	010
SLTIU	If immediate value < rs1 (unsigned), put 1 to rd; otherwise put 0	0010011	011
XORI	XOR immediate value and value of rs1, and put result to rd	0010011	100
SRLI	Shift Right Logical with Immediate (shift rs1 value by immediate amount)	0010011	101

SRAI	Shift Right Arithmetic Immediate (shift rs1 value by immediate amount)	0010011	101
ORI	OR immediate value and value of rs1, and put result to rd	0010011	110
ANDI	AND immediate value and value of rs1, and put result to rd	0010011	111
JALR	Jump and Link Register	1100111	

3. S- Type (Store Type)

   Used for memory store instructions.

   - `opcode` (7 bits): Specifies the operation.
   - `imm[11:5]` (7 bits): Upper bits of the immediate value.
   - `funct3` (3 bits): Function modifier.
   - `rs1` (5 bits): Base register.
   - `rs2` (5 bits): Source register (data to store).
   - `imm[4:0]` (5 bits): Lower bits of the immediate value.

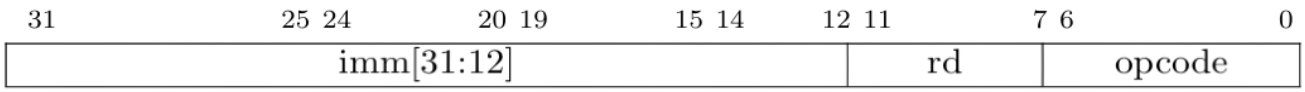


Instruction	Description	Opcode	Func3
SB	Store Byte (rs2 reg value, base is in rs1 address, and the offset is taken from the immediate value)	0100011	000
SH	Store Halfword (rs2 reg value, base is in rs1 address, and the offset is taken from the immediate value)	0100011	001
SW	Store Word (rs2 reg value, base is in rs1 address, and the offset is taken from the immediate value)	0100011	010
SBU	Store Unsigned Byte	0100011	100
SHU	Store Unsigned Halfword	0100011	101

4. U- Type (Upper – Immediate Type)

   Used for instructions that load a 20-bit immediate value into the upper 20 bits of a register.
   The immediate value has only 20 bits, so before loading it to rd , it should be zero extended to 32 bits.

   - `opcode` (7 bits): Specifies the operation.
   - `rd` (5 bits): Destination register.
   - `imm[31:12]` (20 bits): Immediate value.
   - Ex- LUI and AUIPC

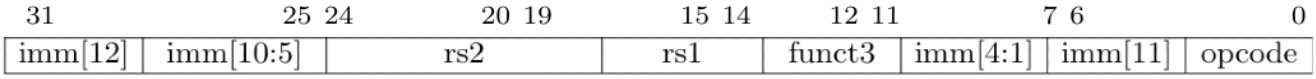


Instruction	Description	Opcode
AUIPC	Add upper immediate to PC and put result in the register	0010011
LUI	Load Upper Immediate (puts the immediate value with 12 zeros at the end into the register)	0110111

5. B- Type (Branch Type)

   The only difference between the S and B formats is that the 12-bit immediate field is used to encode branch offsets in multiples of 2 in the B format. So here it can assume that the immediate value is 13 bits (LSB of immediate value is always 0)

   - `opcode` (7 bits): Specifies the operation.
   - `imm[12]` (1 bit): Sign bit.
   - `imm[10:5]` (6 bits): Upper bits of the offset.
   - `funct3` (3 bits): Branch condition.
   - `rs1` (5 bits): First register for comparison.
   - `rs2` (5 bits): Second register for comparison.
   - `imm[4:1]` (4 bits): Lower bits of the offset.
   - `imm[11]` (1 bit): MSB of the offset.
   - Ex- BEQ, BNE, BLT, BGE, BLTU, and BGEU



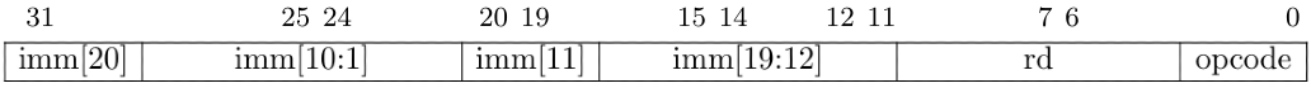
Instruction	Description	Opcode	Func3
BEQ	Branch if equal (ifvaluesinrs1andrs2areequal jump to the offset)	1100011	000
BNE	Branch if not equal (ifvaluesinrs1andrs2arenotequal jump to the offset)	1100011	001
BLT	Branch if lower than (if values in rs1<rs2 jump to the offset)	1100011	100
BGE	Branch if greater than (if values in rs1>rs2 jump to the offset)	1100011	101
BLTU	Branch if lower than, unsigned (if values in rs1	1100011	110
BGEU	Branch greater than or equal, unsigned (ifvaluesinrs1>rs2 jump to the offset)	1100011	111

6. J- Type (Jump Type)

   Similarly, the only difference between the U and J formats is that the 20-bit immediate is shifted left by 12 bits to form U immediates and by 1 bit to form J immediates. Here rd saves the return address (PC +4) and the 20-bit immediate value in the **J-Type** instruction format acts as an offset to the current Program Counter (PC).

   Here also there is jump offset is multiple of two like B Type, therefore the LSB is zero.

- opcode (7 bits): Specifies the operation.
- rd (5 bits): Destination register.
- imm[20] (1 bit): Sign bit.
- imm[10:1] (10 bits): Bits of the offset.
- imm[11] (1 bit): Additional offset bit.
- imm[19:12] (8 bits): More offset bits.
- Imm[0] is always zero
- Ex- JAL and JALR



Instruction	Description	Opcode	Func3
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

OPCODES

Load/store Instructions

Instruction Name	Opcode
LW	0000011
LH	0000011
LHU	0000011
LB	0000011
LBU	0000011
SW	0100011
SH	0100011

ALU Instructions

Instruction Name	Opcode
AND	0110011
OR	0110011
XOR	0110011
ANDI	0010011
ORI	0010011
XORI	0010011
SLL	0110011
SRL	0110011
SRA	0110011
SLLI	0010011
SRLI	0010011
SRAI	0010011
ADD	0110011
SUB	0110011
ADDI	0010011
MUL	0110011
DIV	0110011
DIVU	0110011

REM	0110011
REMU	0110011

Control flow Instructions

Instruction Name	Opcode
BEQ	1100011
BNE	1100011
BLT	1100011
BLTU	1100011
BGE	1100011
BGEU	1100011
JAL	1101111
JALR	1100111

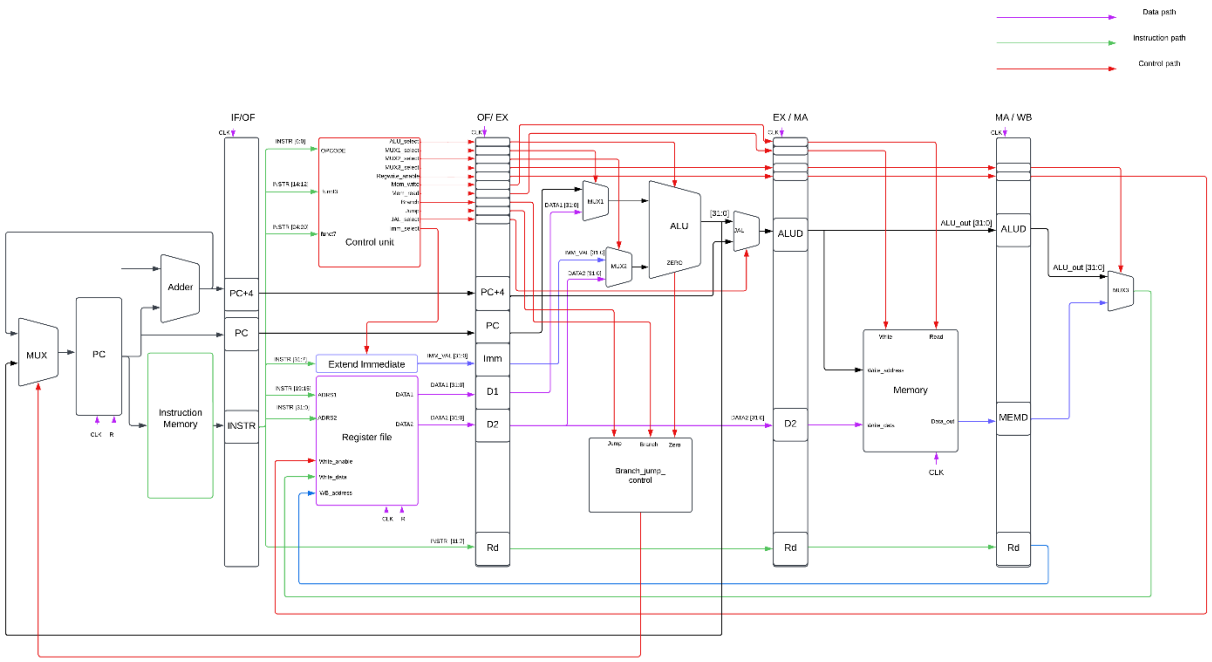


Figure 01: pipeline diagram

Main Hardware Units

1. Instruction Memory

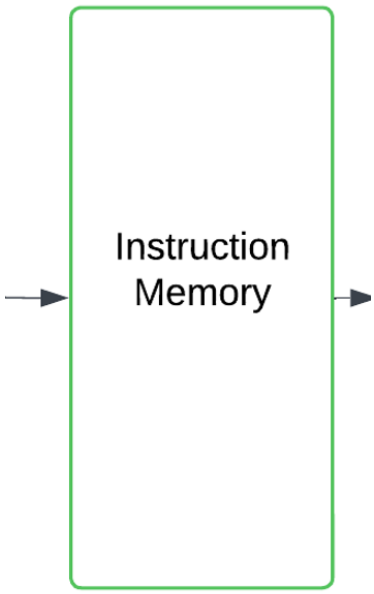


Figure 02: Instruction Memory

Input:

- A 32-bit input representing the address of the instruction to be fetched from memory.
- This address is generated by the program counter (PC) and is aligned to 4 bytes (word-aligned) since RV32IM instructions are 4 bytes long.

Output:

- A 32-bit binary value fetched from the instruction memory.
- This value represents the machine code of the instruction to be executed in the pipeline.

2. Control Unit

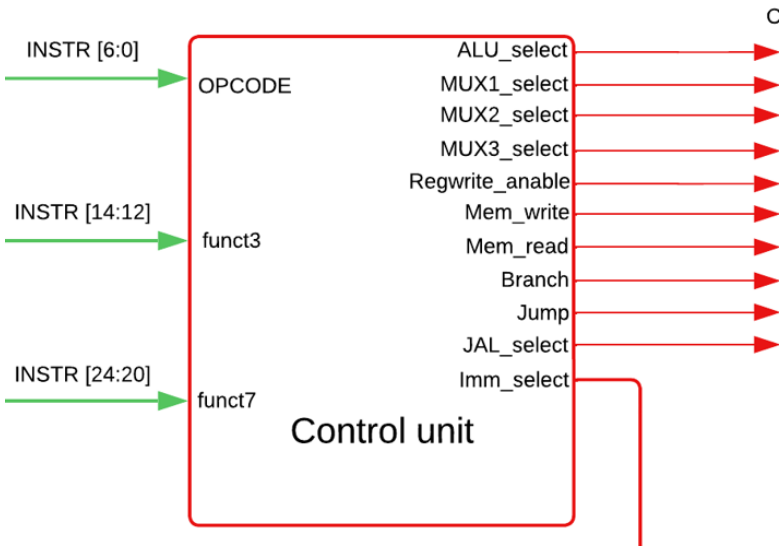


Figure 03: Control Unit

Input:

- Opcode
  - Use the [7:0] bit field in the instruction.
  - Specifies the operation to be performed by the processor.
  - Used to determine the type of instruction (e.g., arithmetic, memory access, branch)
- func3
  - Use the [14:12] bit field in the instruction.
  - Provides additional information about the operation, typically used to differentiate between variants of the same opcode.
- func7
  - Use the [31:25] bit field in the instruction.
  - Providing further operation-specific details, often used with func3 for ALU operations.

Output:

- WRITE_ENABLE:
  - 1-bit signal for Register file which controls whether data is written into the register file.
    - High: Enables write operations to the register file.

        - Low: Disables write operations.

- MUX1_SELECT:
    - 1-bit signal for MUX1, which is used to select between register file output (DATA1) or PC value.
        - 0: Register file output (DATA1).
        - 1: Program counter value (PC).
- MUX2_SELECT:
    - 1-bit signal for MUX2, which is used to select between register file output (DATA) or PC value.
        - 0: Register file output (DATA2).
        - 1: Program counter value (PC).
- MUX3_SELECT:
    - 1-bit signal for MUX1 which selects the input to the register file from
        - 0: ALU result.
        - 1: Data memory output.
- MEM_READ:
    - 1-bit signal for Memory Unit which enables read operations from memory.
        - High: Outputs data from the specified memory address.
        - Low: Disables data output.
- MEM_WRITE:
    - 1-bit signal for Memory Unit which enables write operations to memory
        - High: Writes data to the specified memory address.
        - Low: Disables memory writes.
- JAL_SELECT:
    - 1-bit signal for JAL multiplexer which is used to select between ALU_RESULT or current PC+4 value.
        - 0: Uses ALU result.
        - 1: Uses PC + 4.
- ALUOP:
    - 5 bits signal ALU which specifies the arithmetic or logical operation to be performed.
    - Select one of several ALU operations, such as addition, subtraction, bitwise AND/OR/XOR, or shifts.
- BRANCH:
    - 1-bit signal for BRANCH JUMP CONTROLLER to distinguish whether the instruction is a branch.
- JUMP:
    - 1-bit signal for BRANCH JUMP CONTROLLER to distinguish whether the instruction is a jump.

3. Register File

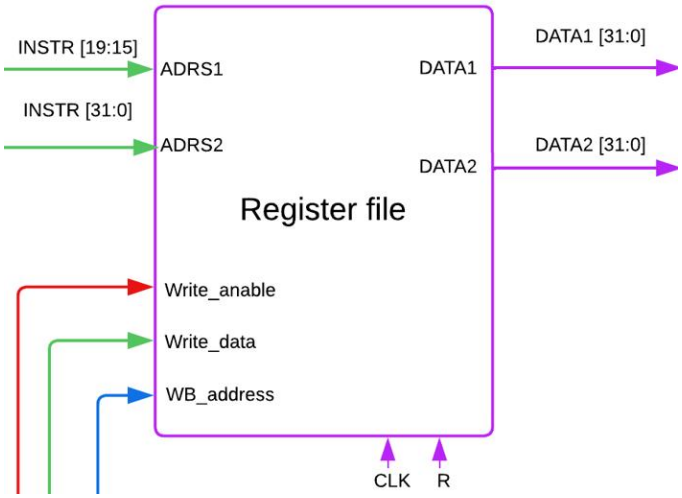


Figure 04: Register File

Input:

- CLK (Clock Signal):
  - Synchronizes operations in the register file.
  - Triggers read/write operations on the rising or falling edge of the clock.
- R:
  - Enables read operations from the register file.
    - High: Allows reading from the registers specified by ADDR1 and ADDR2.
    - Low: Disables reading.
- ADDR1 and ADDR2:
  - Specify the addresses of the registers to be read.
  - These signals determine which registers' contents are output on DATA1 and DATA2.
- WRITE_ENABLE:
  - Enables write operations to the register file.
    - High: Allows data to be written to the register specified by WRITE_REG_ADDR.
    - Low: Disables writing.
- REG_WRITE_DATA
  - Specifies the data to be written into a register during a write operation.

Output:

- DATA1 and DATA2
  - Provide the data read from the registers specified by ADDR1 and ADDR2.

- These outputs are typically operands for arithmetic or logical operations performed in the ALU.

4. ALU

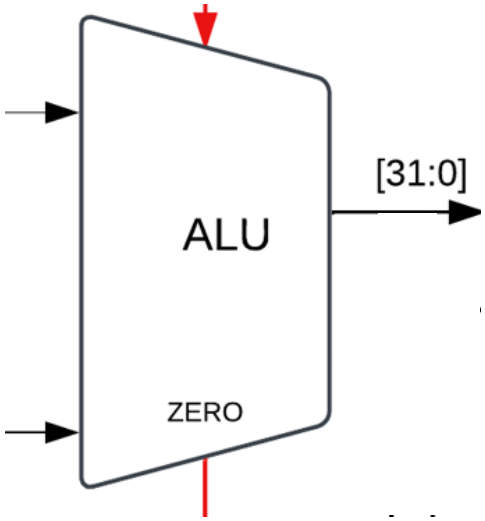


Figure 05: ALU

Input:

- MUX1 Output:
  - Provides one of the ALU's operands.
  - This data is selected by MUX1, which typically chooses between the output of the register file (DATA1) or the program counter (PC)
- MUX2 Output:
  - Provides one of the ALU's operands.
  - This data is selected by MUX2, which typically chooses between the output of the register file (DATA1) or an immediate value.
- ALUOP
  - Specifies the operation to be performed by the ALU, such as addition, subtraction, bitwise AND/OR, shifts, or comparisons.

Output:

- ALU_RESULT
  - The primary output of the ALU after performing the specified operation on its inputs.
- ZERO
  - Indicates whether the result of the operation is zero.
    - High (1): Result is zero.
    - Low (0): Result is non-zero.

5. Branch-Jump Controller

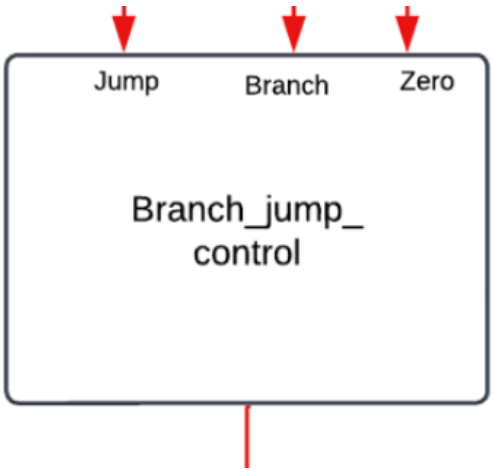


Figure 06: Branch-Jump Controller

Inputs:

- BRANCH
  - The BRANCH input is a signal that indicates whether a branch instruction has been executed.
- JUMP
  - The JUMP input is a signal that indicates whether a jump instruction has been executed.
- ZERO
  - The ZERO input is a signal that indicates whether the ALU has produced a zero output.

Output:

- Branch/ Jump signal
  - This is used in PC MUX as the select bit to choose the next PC value

6. Memory

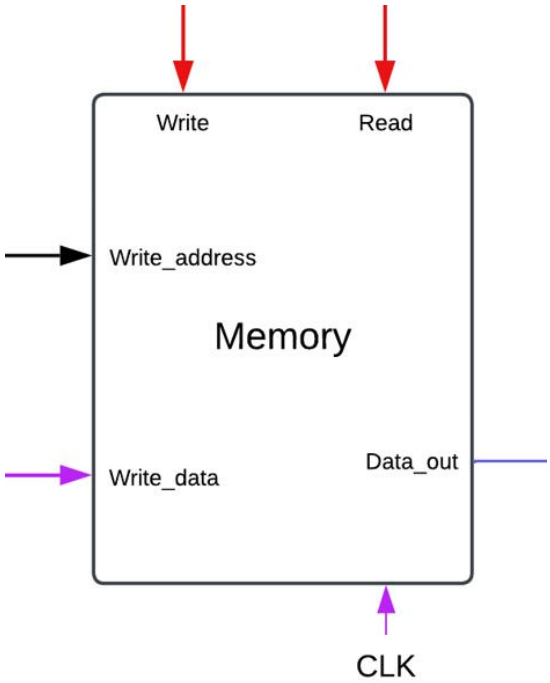


Figure 07: Memory

Inputs

- CLK
  - The clock signal is an input that synchronizes the operation of the memory with other components in the system.
- Read
  - The MEM_READsignal is an input that specifies whether the memory should read data from the specified address.
- Write
  - The MEM_WRITE signal is an input that specifies whether the memory should write data to the specified address.
- Write_address
  - The MEM_ADDRESS signal is an input that specifies the memory address to read from or write to.
- Write_data
  - The DATA_IN signal is an input that provides data to be written to the specified memory address.

Outputs:

- Data_out

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