GROUP - 28

Kandula Revanth (21CS10035)
Pola Gnana Shekar (21CS10052)

Verilog Assignment 7:

Design and synthesize a 32-bit processor using Verilog

CONTENTS:

❖ Instruction Encoding:

1. 3 input instructions (R-type):

Opcode	rs	rt	rd	shamt	funct
6 bits (Arithmetic load/ store/ branch).	**5 bits** (Source register - operand A)	**5 bits** (Source register - operand B)	**5 bits** (Dest. register - result)	**5 bits** (Shift Amount)	**6 bits** (function code)

Examples:

- ★ **ADD R1, R2, R3:** R1 = R2 + R3
- ★ **SUB R4, R5, R6:** R4 = R5 - R6
- ★ **XOR R7, R8, R9:** R7 = R8 XOR R9

2. 2 input instructions (I-type):

Opcode	rs	rt	immediate value
6 bits (Operation code)	**5 bits** (Source register)	**5 bits** (Dest. register)	**16 bits** (Immediate value)

Examples:

- ★ **LD R10, 20(R11):** R10 = Memory[R11 + 20]
- ★ **ST R2, -4(R3):** Memory[R3 - 4] = R2

3. 1 input instructions (J-type):

Opcode	immediate value
6 bits (Operation code)	**26 bits** (immediate value)

Examples:

- ★ **NOT R12**: R12 = NOT R12
- ★ **SLAI R5, #2:** R5 = R5 << 2 (Shift left by 2 bits)
- ★ **BR #30:** PC = PC + 30 (Unconditional branch)

❖ Opcode for Different Instructions:

Operation	OPCODE	funct	ALUfunc
ADD	000 000	000000	000
SUB	000 001	001000	001
NOT	000 010	010000	010
AND	000 011	011000	011
XOR	000 100	100000	100
SLL	000 101	101000	101

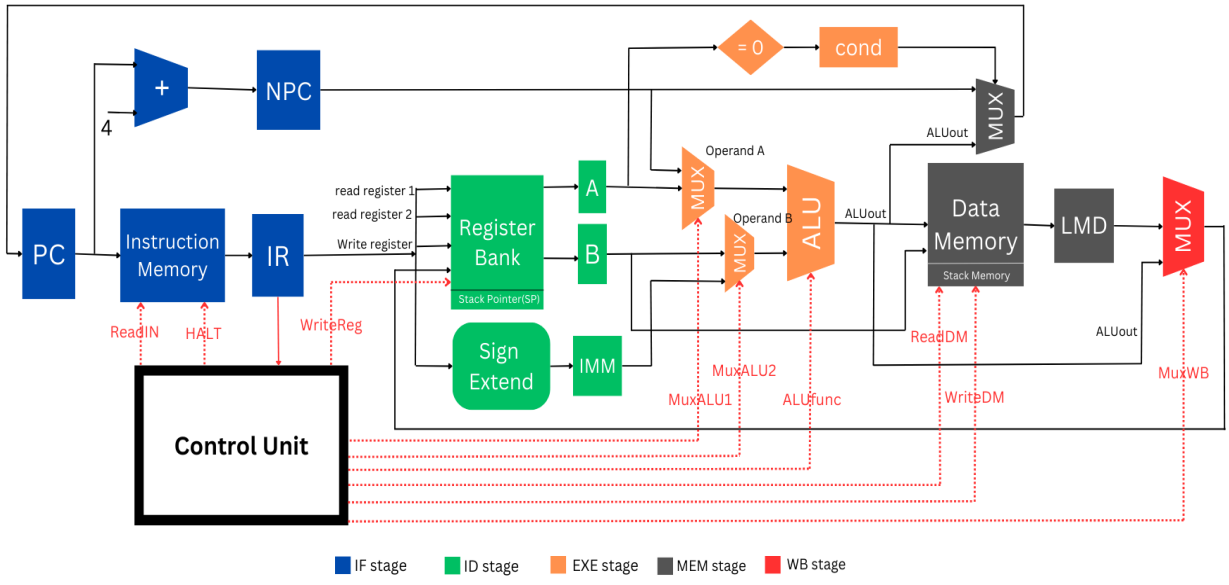
SRL	000 110	110000	110
SRA	000 111	111000	111
ADDI	001 000	N/A	000
SUBI	001 001	N/A	001
NOTI	001 010	N/A	010
ANDI	001 011	N/A	011
XORI	001 100	N/A	100
SLLI	001 101	N/A	101
SRLI	001 110	N/A	110
SRAL	001 111	N/A	111
LD	010 000	N/A	000
ST	010 001	N/A	000
LDSP	010 010	N/A	000
STSP	010 011	N/A	000
BR	011 000	N/A	000
BMI	011 001	N/A	000
BPL	011 010	N/A	000
BZ	011 011	N/A	000
PUSH	100 000	N/A	000
POP	100 001	N/A	000
CALL	100 010	N/A	000
RET	100 011	N/A	000
MOVE	101 000	N/A	000
NOP	110 000	N/A	N/A

HALT	111 000	N/A	N/A

❖ Control Signals:

Control Signal	Function
ReadIN	This signal controls the reading of instructions from memory. When active, it indicates that the processor should fetch the next instruction from memory.
MuxALU1	This signal controls a multiplexer (MUX) that selects the first input to the ALU. It specifies whether the first input should come from a register or an immediate value.
MuxALU2	This signal controls a MUX that selects the second input to the ALU, determining whether it should come from a register or an immediate value.
ALUfunc	This signal specifies the operation to be performed by the Arithmetic Logic Unit (ALU). It determines whether the ALU should perform addition, subtraction, bitwise operations, etc.
WriteDM	This signal enables writing data into the data memory. It is used for store instructions that store data into memory.
ReadDM	This signal enables reading data from the data memory (RAM) and loads it in LMD(Load Memory Data) value.
MuxWB	This signal controls a MUX that selects the destination for writing the result. It determines whether the result should be written back to a register, memory, or elsewhere
WriteReg	This signal controls the writing of data into a register file. It specifies when to write the result of an operation or instruction into a register.
HALT	The HALT control signal, when activated, halts the execution of instructions in the processor.

❖ Schematic Diagram:



❖ Design Description:

- **Opcode Description:**

The first three bits of the opcode field in our instruction format are used to categorize instructions into different operation types. These operation types are as follows:

000 (Arithmetic): This category includes arithmetic and logical operations. The next three bits in the opcode are used as ALUfunc to specify the exact operation, such as ADD, SUB, AND, etc.

001 (Immediate Value): Instructions in this category also utilize the next three bits as ALUfunc, specifying operations involving immediate values.

010 (Load/Store): Load and store instructions, for memory access, use ALUfunc as 000, signifying that addition of an offset is required to compute the memory address.

011 (Branch): Branching instructions use ALUfunc as 000 for conditional branch calculations, signifying that addition of an offset is required

100 (Push/Pop/Ret/Call): These instructions utilize ALUfunc as 000.
101 (Move Operation): Move instructions use ALUfunc as 000.
110 (NOP): The ALUfunc is not applicable (N/A) for NOP instructions.
111 (HALT): The ALUfunc is not applicable (N/A) for HALT instructions.

- **Data Path Description:**

Instruction Fetch: The first instruction is fetched from memory at the address stored in the Program Counter (PC). The instruction is loaded into the Instruction Register (IR).

Control Unit: The Control Unit interprets the opcode to set various control signals based on the type of instruction. For example, if the instruction is HALT, the HALT control signal is activated, causing the processor to halt. The ReadIM control signal is set to fetch the next instruction from memory, and the PC is incremented by 4 to prepare the address for the next instruction, stored in the Next Program Counter (NPC).

Register Access: Values stored in the registers (read register 1 and read register 2) are passed to the A and B operands within the data path. The destination register for the result is determined based on the instruction (write register).

Immediate Values: The immediate value is sign-extended (if needed) and placed in the IMM (Immediate) for subsequent operations.

ALU Operations: The selection of operands for the ALU is determined by the MuxALU1 and MuxALU2 control signals, allowing for flexibility in data sources, including registers, immediate values, and memory addresses. The specific operation to be carried out is governed by the ALUfunc control signal, serving as a code that instructs the ALU on whether to perform arithmetic operations (e.g., addition, subtraction) or logical operations (e.g., bitwise AND, OR, XOR). Additionally, the ALU can handle bit-shifting operations (e.g., left shift, right shift), with the shift amount typically defined by the instruction. The outcome of the operation, referred to as ALUout, represents the computed result, which is subsequently employed in various stages of the data path as needed by the instruction's requirements.

Memory Access: In load, store, push, pop, and similar instructions that involve memory addresses, the ALUout value is used as the address for data memory access. The WriteDM control signal enables writing to memory, and the ReadDM control signal enables reading from memory. For load instructions, the data retrieved is passed to the Load Memory Data (LMD) register.

Write Back: ALUout is passed through the Write Back Mux (MUX WB), which selects either ALUout or the data loaded from memory as the value to be written back to the registers. The WriteReg control signal is used to write the result into the destination register specified by the instruction.

Branching: For branching instructions, the ALUout value is used to calculate the branch target address. Based on the branch condition (e.g., equal to zero), the processor selects either the branched target or the next instruction's address using a MUX, which is then passed to the PC, updating the program counter for the next instruction.

Move Operation: In a move operation, the A operand is set to the source register whose value is to be transferred. The B operand is set to the zero register since no arithmetic or logical operation is required. The ALU performs an ADD operation on these operands, resulting in the same value as operand A. The destination register (specified by the write register field) is then updated with the value of ALUout. This move operation effectively copies the value from one register to another without any modification

Stack Pointer Functionality: The stack pointer, stored as a dedicated register within the register bank, points to an address within the stack memory. The stack pointer's value can be updated through operations that modify its content, and these modifications are managed through the write back stage, ensuring that the stack pointer's value is always in sync with stack memory.

Push/Pop Operation: In a push operation, the processor follows this sequence, read register 1 is set to the zero register (a register containing the value 0), and read register 2 is set to the stack pointer. The ALU then performs an ADD operation, utilizing an immediate value (the data to be pushed). The result of this ADD operation is identical to the immediate value and is designated as ALUout. This ALUout value represents the data to be pushed onto the stack, and it is directed to the data memory, which corresponds to the stack memory. The address to which the data is to be pushed is specified by the stack pointer, and this address is used as operand B in the data memory. The updated stack

pointer value is returned to the stack pointer register via the write back stage, ensuring that it points to the next available location in the stack memory.

In this manner, the data path of the processor effectively manages the execution of a wide range of instructions by controlling the flow of data and operations in response to instruction type and control signals.