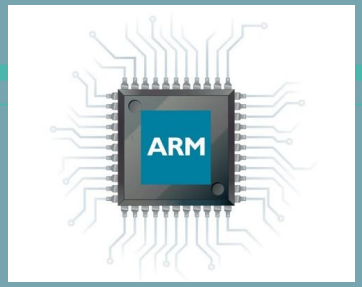
PROCESSOR DESIGN



Sripranav Mannepalli (CS18B036)

SRVS Maheswara Reddy (CS18B032)

Satya Sai N (CS18B024)

ARM (RISC) 32 bit general purpose processor:

- ARM – Advanced RISC (Reduced Instruction Set Computer) Machine
- Load/store architecture
- Single Cycle Processor
- Uniform and fixed-length(32 bit) instructions

Plan :

- Instruction Set Architecture
- Implementation of various Modules
  - Add, Sub, And, Or, Xor, Mux, SignExtend, Registers, BranchPC, JumpPC, ROM, ALU, ALU Control, CPU
- TestBenches
- Contributions

INSTRUCTION FORMAT

Arithmetic and Logical Type :

Load Store Type:

IMPLEMENTATION

➔ Add Module (Add.v)

Inputs : in1 , in2
Output : out
Psuedo Code : out = in1+in2

➔ Sub Module (Sub.v) ; And Module (And.v) ; Or Module (Or.v) ; Xor Module (Xor.v)

➔ Mux Module (Mux.v)

Explanation : A 2X1 Multiplexer.

➔ SignExtend Module (SignExtend .v)

Explanation : A Module to increase number of bits while preserving sign and value.

➔ Registers Module (Registers.v)

Inputs : readReg1 , readReg2 , writeReg , writeData , ControlRegWrite
Output : readData1 , readData2
Psuedo Code :

```
readData1 = memory[readReg1]
readData2 =memory[readReg2]
If ControlRegWrite == 1:
        memory[writeReg] = writeData
```

IMPLEMENTATION

➔ BranchPC Module (BranchPC.v)

Inputs : currentPC , imm
Output : branchPc
Psuedo Code : branchPc = currentPC + (imm<<2) (Our each is instruction is 4 Bytes)

➔ JumpPC Module (JumpPC.v)

Inputs : currentPC , inst
Output : jumpPc
Explanation : Jump instructions specify an absolute address which the PC will be set to whereas branch instructions offset the address in the program counter.

➔ RAM Module (RAM.v)

Inputs : address ,in , ControlMemRead , ControlMemWrite
Output : out
Psuedo Code :

```
If ControlMemRead == 1:
        memory[address ] = in
If ControlMemWrite == 1:
        out = memory[address ]
```

IMPLEMENTATION

➔ ROM Module (ROM.v)

Inputs : PC
Output : instruction
Psuedo Code :

```
instruction[7:0] = memory[PC+3]
instruction[15:8] = memory[PC+2]
instruction[23:16] = memory[PC+1]
instruction[31:24] = memory[PC]
```

➔ ALU Control Module (ALUControl.v)

Inputs : opcode
Output : ControlAluBits
Explanation : Based on the type of instruction and OpCode, it sets the ControlAluBits in such a way that ALU Module gets to know
What Arithmetic / Logical operation to perform.
Example : If opcode == 6'b001000 , then ControlAluBits is set to 6'b100000 (add instruction)
Similarly for other operations like "and" , "or" , "xor" , "beq" , "bne" , "bgtz" , "bltz" , "blez" , "bgez" etc.

IMPLEMENTATION

➔ ALU Module (ALU.v)

Inputs	:	in1 , in2 , ControlAluBits (obtained from ALUControl Module)
Output	:	out , zero
Explanation	:	Based on the ControlAluBits , we get to know what Arithmetic/Logical operation to perform on the in1 and in2 inputs.
Example	:	If ControlAluBits == 6'b001000 , then we do : out = in1+in2 (add instruction) Similarly, we do for other Arithmetic operations like "or" , "and" , "xor" , "sub" etc If ControlAluBits == 6'b110000, then we do : zero = (in1==in2) (beq instruction) Similarly, we do for other Arithmetic operations like "bne" , "bgtz", "bltz", "blez", "bgez" etc.

➔ CPU Core Module (CPU.v)

Inputs	:	Clk , instruction , regData1 , regData2 , ramOutput,
Output	:	ControlRegWrite , ControlMemRead , ControlMemWrite , ControlBranch readReg1, readReg2 , writeReg , aluOutput , writeRegData , PC
Explanation	:	Our CPU Module is the main module that integrates all the other modules. Any instruction is recieved by this module. Then, this module decodes the instruction and sends to the corresponding modules for execution.
Example	:	Let us say that that there is an "R" type instruction

TESTBENCHES

We wrote TestBenches for our modules. A test bench or testing workbench is an environment used to verify the correctness or soundness of a design or model.

Example :

Module

```
module Adder(
        input [31:0] in1,
        input [31:0] in2,
        output [31:0] out);

        assign out = in1+in2;
endmodule
```

TestBench

```
module Adder_tb;
        reg [31:0] i1;
        reg [31:0] i2;
        wire [31:0] o;
        Adder test(i1, i2, o);
        initial begin
                i1 = 32'00000000;
                i2 = 32'00000001;
                #100
                i1 = 32'fffffffe;
                i2 = 32'00000001;
                #100
                $finish;
        end
endmodule
```

CONTRIBUTION : Sripranav Mannepalli (CS18B036)

IDEA GENERATION:

- Did research on existing Processors based on RISC architectures.
- Collaborated with my team members to plan about our Instruction Set Architecture.
- Collaborated with my team members to design our ALU and CPU.
- Collaborated with my team members to plan about the modules that we need to implement.
- Studied various repositories in github so that we can get a good head start before starting our implementation.

LINES OF CODE:

- **30** LOC in Add.v for Adder Module (Module with Testbench) .
- **45** LOC in Mux.v for Mux Module (Module with Testbench).
- **75** LOC in RAM.v for RAM Module (Module with Testbench).

CONTRIBUTION : Sripranav Mannepalli (CS18B036)

- **70** LOC in Register.v for Register Module (Module with Testbench).
- **100** LOC in ALUControl.v for ALUControl Module
- **150** LOC in ALU.v for ALU Module.
- **150** LOC in CPU.v for CPU Module.

WORKING STATUS:

- Addition, Subtraction, Multiplexer, RAM, Register Modules are all implemented.
- Wrote TestBenches for Add Module, Sub Module, Mux Module, RAM Module and Register Modules and all these modules are working as expected.
- Tested CPU and ALU modules for few examples and they are working.

OTHERS:

- Contributed in preparing the Final Presentation.

CONTRIBUTION SRVS Maheswara Reddy (CS18B032)

IDEA GENERATION:

- Collaborated with team members on the design of ALU.
- Collaborated with team members to plan about the modules we implemented.
- Studied some existing works on the processor design.
- Part of the discussion on ISA design.

LINES OF CODE:

- 25 LOC in And.v for And Module.
- 25 LOC in OR.v for Or Module.
- 30 LOC in SignExtend.v for SignExtend Module.
- 80 LOC in ROM.v for ROM Module.

CONTRIBUTION SRVS Maheswara Reddy (CS18B032)

- 20 LOC in jumpPc.v for jump module
- 80 LOC in ALU.v for ALU Module.

WORKING STATUS:

- And, Or, SignExtender, JumpPc, ROM are all implemented.
- Wrote test bench for the ROM module and register modules. All these are working as expected.

OTHERS:

- Contributed in preparing the Final Presentation.

CONTRIBUTION Satya Sai N (CS18B024)

IDEA GENERATION:

- Collaborated with my team members to plan about our Instruction Set Architecture.
- Collaborated with my team members to design our ALU and CPU.
- Collaborated with my team members to plan about the modules that we need to implement.
- Researched about existing work on processor design

LINES OF CODE:

- **25** LOC in Sub.v for Subtraction Module
- **25** LOC in Xor.v for Xor Module
- **100** LOC in ALUControl.v for ALUControl Module
- **100** LOC in ALU.v for ALU Module

CONTRIBUTION Satya Sai N (CS18B024)

- **50** LOC in RAM for RAM Module
- **50** LOC in BranchPC.v for BranchPC Module

WORKING STATUS:

- Sub, Xor, BranchPc, ALUControl, RAM module were implemented
- Wrote TestBenches for Sub, Xor, BranchPC and all these modules are working as expected.
- Tested ALU module for few examples and they are working fine as expected.

OTHERS:

- Contributed in preparing the Final Presentation.

THANK YOU