RISC-V Single-Cycle Processor
Milestone 2
Report

Omar Miniesy 900202087
Ziad Miniesy 900202283

Implementation
To support the RISC-V 40 instructions set, we added extra control signals to the control unit to have the following signals at the end: Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUsrc, RegWrite, auipc, jump, srcPC, pcload. A branching module was added to support different branching instructions; this also required introducing multiple flags in the ALU to check which branching instruction was to be executed.

The instruction types were tested by the assembly codes attached in the folder following this naming format: “*-type assembly.txt” and the corresponding hexadecimal notation which is loaded to the instruction memory “*type_test.mem”

R-type
For the r-type instructions, we modified the ALUop control signal to 3 bits instead of 2 bits to support more instructions. In the control unit, the opcode of the instruction is checked, and accordingly its control signals are determined including the ALUop which is then transferred to the ALUControl. The ALUcontrol decides which ALU operation to execute based on the ALUop as well as the funct3 and instruction[30]. We added another shifter module to support the shifting instructions in the r-type. This module takes the alu function to differentiate between different shifting instructions, it also takes the shifting amount and the value to be shifted.

I-type
For the I-type instructions we used the same ALUop signals to determine the ALU operation, the difference was the second input of the ALU which in that case was the output of the immediate generator.

The second part of the I-type instructions is the load instructions, it uses the MemRead control signal from the ALU and the funct3 of the instruction to check which load operation is to be executed. Then the value is loaded to the specified register.

Moreover, the jalr instruction used the jump control signal, the pcload and the srcPC to update the PC; it also stores this value in the register file.
Finally, the ECALL and FENCE instructions use the srcPC control signal to jump to address zero. The EBREAK instruction uses the pcload control signal which is set to zero in that case to prevent updating the PC.

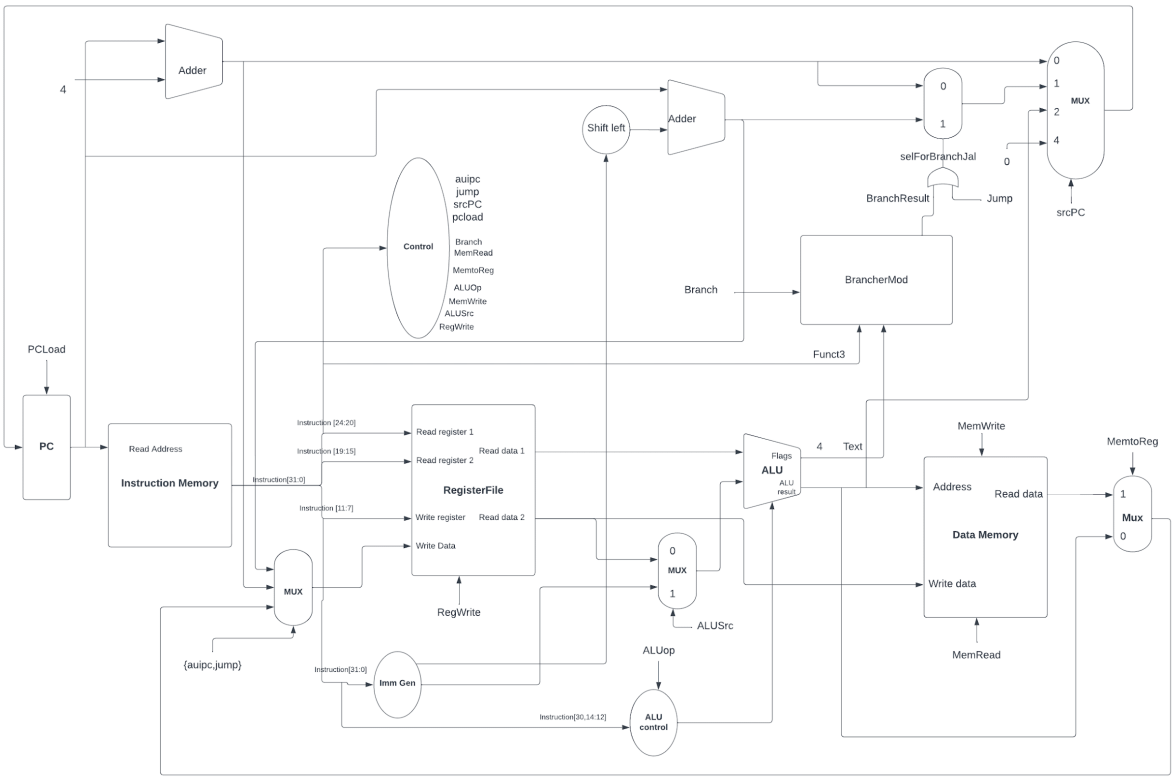
S-type
The store instructions used MemWrite control signal as well as the funct3 to determine which store instruction is to be executed. Then the specified value is stored in the chosen memory address in little endian format.

B-type
As mentioned above, those instructions required multiple flags: zf, vf, sf, cf. Those flags are computed in the ALU after checking if the 2 values are equal or one is greater than the other. This module also takes the branch control signal as well as the funct3 of the instruction. The output of this module is passed to an OR gate with the Jump control signal. Their output is then passed to a mux to determine the next value for the pc. If the instruction is either a jump instruction or branch instruction, the value of the pc is the current address + the shifted immediate value. Otherwise the next pc value is the current address + 4. There is then another mux to determine the final value for the pc, which is the selection of the previous mux result for jal instructions or the ALU result in case of jalr or the adder result which is the current address + 4 or zero in case of EBREAK.

U-type
The lui shifts the immediate 12 bits and stores the result in the specified register. The auipc instruction uses the auipc control signal to add the shifted immediate to the PC and store it in the specified register. At this moment we can write to a register using multiple instructions, either by loading from the data memory, or from U-type instructions, therefore, there should be a mux to determine which of those values will be written to the register file, this mux's select line is the concatenation of the jump and auipc control signal.

J-type
The jal instruction uses the jump control signal. It stores in the destination register the PC+4 and adds the immediate value to the PC. This value is then passed to the mux to determine the next PC value.

Schematic


Adder
4
Shift left
Adder
0
1
MUX
0
1
2
4
selForBranchJal
0
BranchResult
Jump
srcPC
auipc
jump
srcPC
pcload
Control
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Branch
BrancherMod
Funct3
PCLoad
PC
Read Address
Instruction Memory
Instruction [24:20]
Instruction [19:15]
Instruction[31:0]
Instruction [11:7]
Read register 1
Read register 2
Read data 1
RegisterFile
Write register
Read data 2
Write Data
RegWrite
MUX
{auipc,jump}
Instruction[31:0]
Imm Gen
Instruction[30,14:12]
ALU control
ALUop
0
MUX
1
ALUSrc
Flags
ALU
ALU result
4
Text
MemWrite
Address
Read data
Data Memory
Write data
MemRead
MemtoReg
1
Mux
0