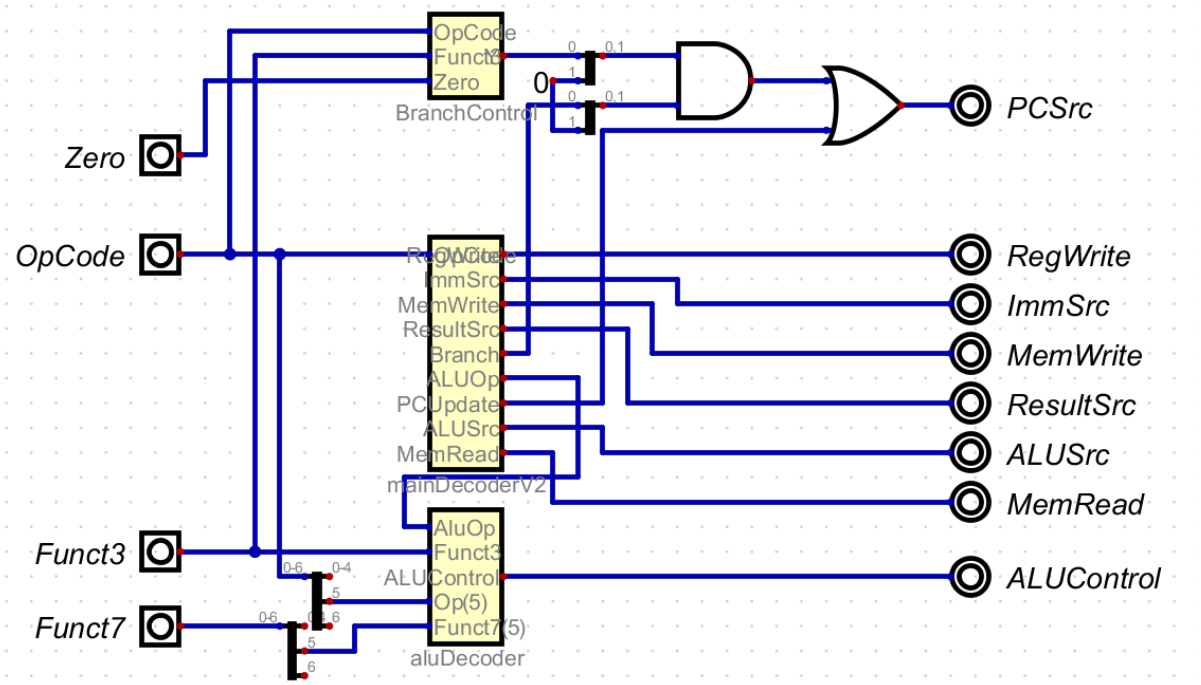
Relatório Arquitetura de Computadores
Gabriel Justus Ramos (GRR: 20232348)
Caio Francisco Stadler Sguario (GRR: 20232340)
Ricardo Prado Faria (GRR: 20235161)

Unidade de Controle:



Como podemos ver, a unidade de controle possui 3 subcircuitos, BranchControl, MainDecoder e AluDecoder.

BranchControl: Um pequeno circuito que usa o OpCode e Funct3 para determinar qual é o tipo de branch que o programa está tentando executar.

MainDecoder: Baseado no OpCode da instrução, ele envia os sinais apropriados para o resto do circuito, além disso, envia um sinal para o AluDecoder.

Tabela verdade usada para montar o circuito do MainDecoder:

Instrução	RegWrite	ImmSrc	ALUSrc	MemWrite	MemRead	ResultSrc	Branch	ALUOp	PCUpdate
R - Type	1	XX	0	0	0	0	0	10	0
I - Type	1	0	1	0	0	0	0	10	0
B - Type	0	10	0	0	0	XX	1	1	0
LW	1	0	1	0	1	1	0	0	0
SW	0	1	1	1	0	XX	0	0	0
JAL	1	11	X	0	0	10	0	XX	1
JALR	1	0	X	0	0	10	0	XX	10

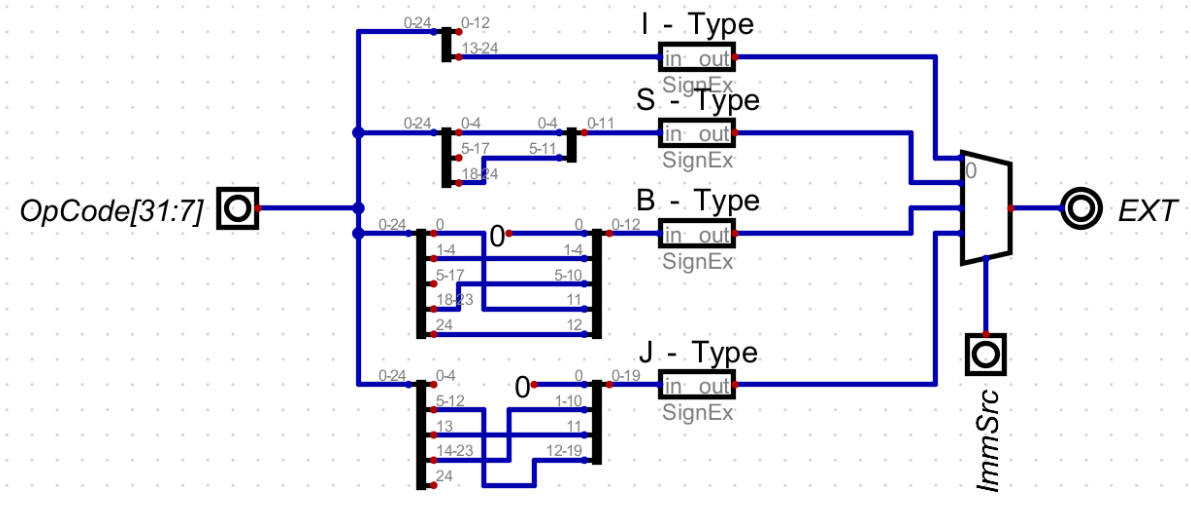
AluDecoder: Baseado no Funct3 e no quinto bit do OpCode e Funct7, ele determina qual será a operação realizada pela ALU.

Tabela verdade usada para montar o circuito do AluDecoder:

ALUOp	Funct3	Op5/Funct7	AluControl
0	XXX	XX	0
1	0/1	XX	110
1	100/101	XX	111
10	0	0/1/10	0
10	0	11	1
10	10	XX	111
10	110	XX	11
10	111	XX	10
10	1	XX	101
10	100	XX	100

Halt: Para fazer essa instrução ligamos o bit menos significativo do OpCode como enabler do PC(registrador), pois em todas as instruções esse bit é igual a 1.

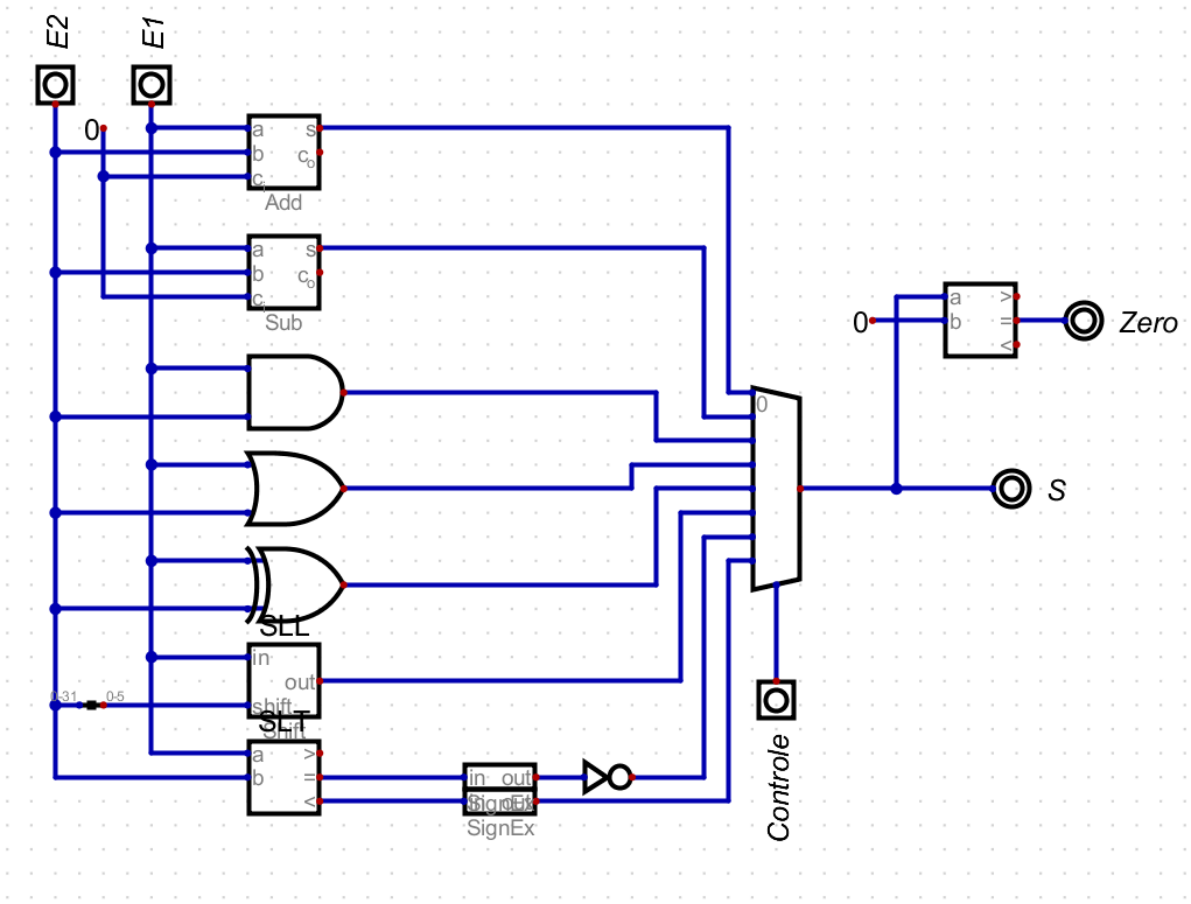
Extensor de Sinal:



O extensor de sinal foi feito com base na seguinte tabela:

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

ALU:

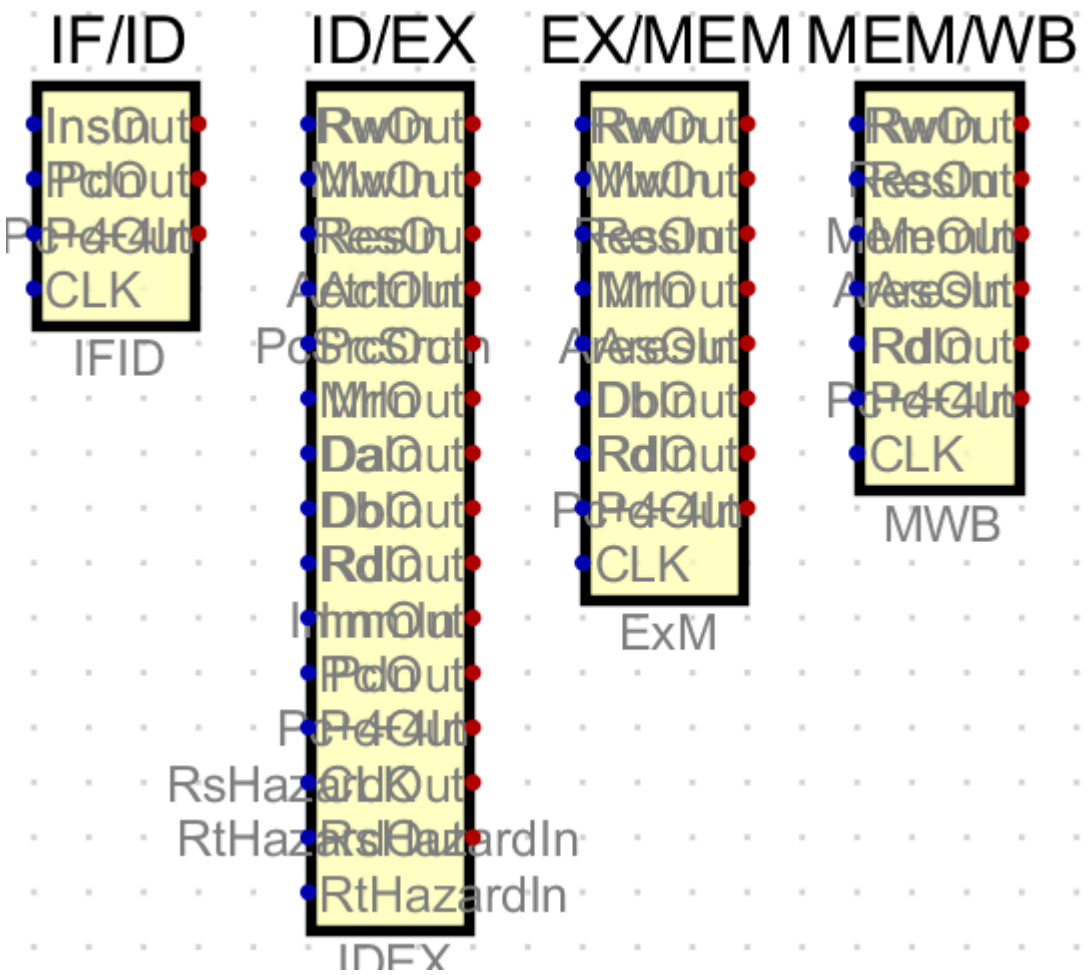


Adicionamos a operação para comparar se dois números são iguais para facilitar no BEQ e BNE.

Existem 3 adders individuais pois um é PC = PC + 4, outro é PC = PC + IMM para o JAL e o último é PC = RS1 + IMM para o JALR

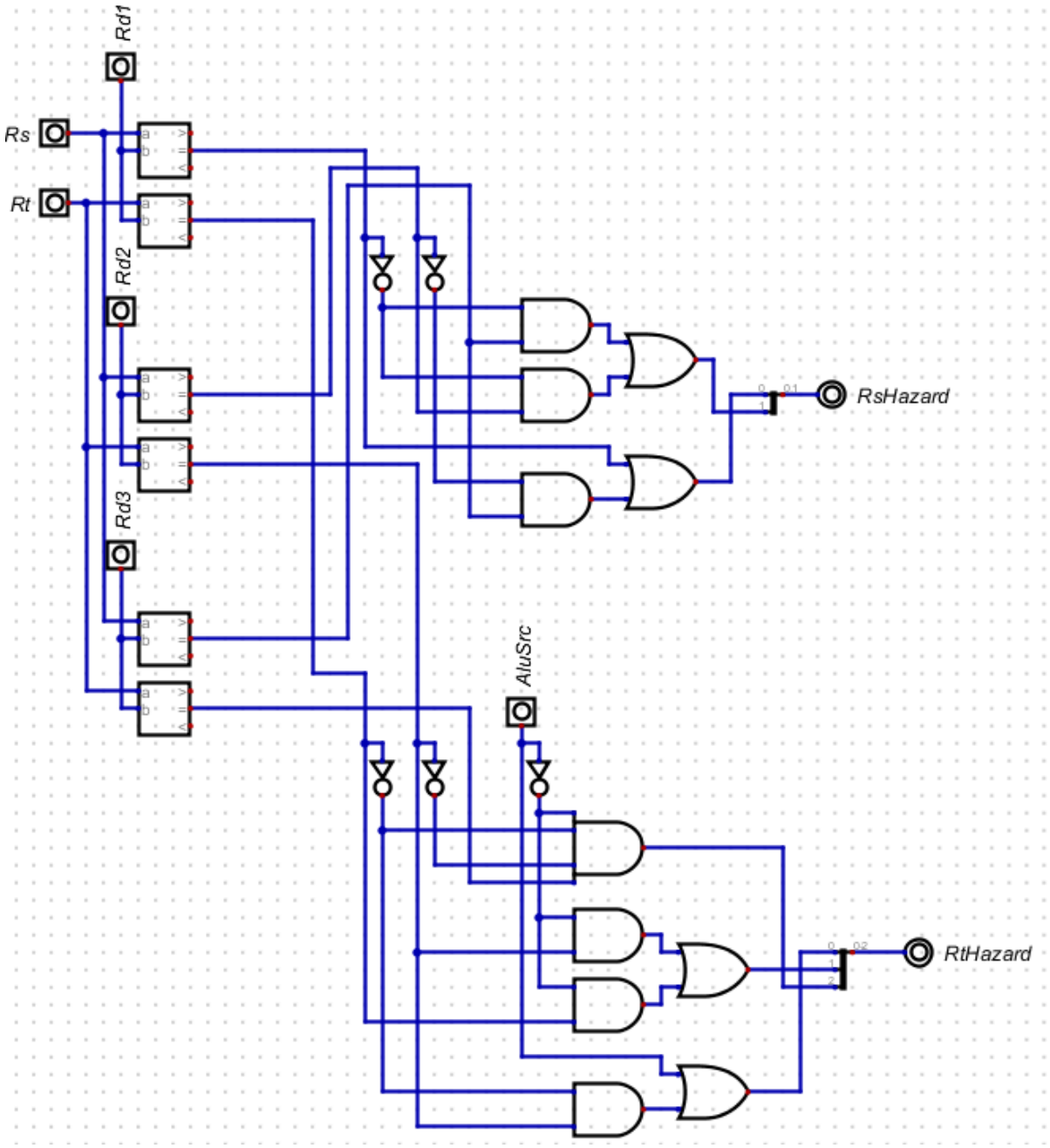
Existem 3 mux no circuito, o primeiro determina o resultado de qual dos adders acima vai para o PC, o segundo determina se a ALU vai usar um imediato ou registrador, e o último determina qual informação será guardada no banco de registradores, sendo que as opções são: PC + 4, resultado da ALU ou informação que estava contida na memória de dados

Para implementar o pipeline, inicialmente fizemos a divisão dos estágios em: IF, ID, EX , MEM e WB, para isso, utilizamos um conjunto de registradores entre cada estágio, esses conjuntos são: IF/ID, ID/EX, EX/MEM, MEM/WB, sendo que, esses conjuntos passam para frente todos os sinais de controle e informações necessárias para a execução da instrução nos diversos estágios do processador.



Quanto ao tratamento de hazards, optamos por tratar dos hazards de dados, para isso, fizemos uma pseudo-hazardUnit, que baseado, nos registradores de destino presentes nos estados EX, MEM e WB e no Rs e Rt da instrução atual, determina se há algum hazard presente no código e envia para os mux presentes nas entradas da ULA o sinal apropriado para tratar o hazard em questão.

Hazard Detection Unit:



Porém, como não era possível fazer a escrita em borda de descida, optamos por adicionar um registrador extra no estágio WB para guardar o resultado final por 1 ciclo para que ele possa ser uma entrada válida para a ULA.