Relatório ISA 12 bits

Por se tratar de uma arquitetura de **12 bits**, a organização das minhas **instruções** foi feita priorizando operações maiores com o registrador, tornando a codificação mais detalhada, porém, com uma maior capacidade.

Optei por usar 4 bits para o opcode, visando implementar até **16 instruções** diferentes. Embora não tenha conseguido implementar todas, o número de instruções desenvolvidas ainda assim ultrapassou as 8 instruções máximas que seriam possíveis caso tivesse optado por 3 bits para o opcode. Outro benefício dessa escolha foi não precisar me preocupar com o espaço geralmente reservado para a função (func), que é utilizada na **ULA**, aplicando isso a todos os tipos de formatos.

As **instruções do tipo R** funcionam no padrão: Registrador destino recebe ele mesmo com uma operação lógica-aritmética (RD <- RD + RA).

As **instruções do tipo I** possuem um registrador fixo onde será salvo o imediato de 8 bits, o que possibilita a não utilização de outro registrador. Uma das complicações encontradas é que essa abordagem impõe a necessidade de zerar o registrador fixo sempre que for adicionar outro imediato com a instrução ADDI, a fim de evitar a soma indevida dos imediatos.

As **instruções do tipo J** funcionam de maneira similar, onde, além do opcode, é incluído o endereço para onde se deseja pular.

As instruções para **Store**, **Load** e **Branch** ficaram similares. Além do opcode, utilizo 4 bits para um registrador que sofrerá a comparação (no caso de Branch) ou que receberá/salvará um valor (para Load e Store). Os 4 bits restantes são destinados a um imediato, que servirá para realizar o salto ou indicar a posição na memória.

Formato das instruções:

Tipo R:

OPCODE	RD	RS1		
4	4	4		

Tipo I:

OPCODE	IMM
4	8

Tipo J:

OPCODE	IMM
4	8

Tipo S/B:

OPCODE	RS1	IMM
4	4	4

A convenção feita para essa ISA foi:

r0 > zero

r1 > retorno

(r2 - r7) > temporários

(r8 - r11) > argumentos

(r12 - r13) > salvos

r14 > stack

r15 > salva imm

Na implementação dentro do logisim, a **Unidade de Controle (UC)** vai usar essa tabela para gerenciar o circuito:

INST	OPCODE	M_IMM	M_ULA	WE_BR	FUNCT	SAVE	WE_ME	PC_OP	HEXADECIMAL CODE
ADD	0000	00	0	1	000	01	0	00	108
SUB	0001	00	0	1	001	01	0	00	1128
MUL	0010	00	0	1	010	01	0	00	2148
DIV	0011	00	0	1	011	01	0	00	3168
AND	0100	00	0	1	100	01	0	00	4188
OR	0101	00	0	1	101	01	0	00	51A8
ADDI	0110	01	1	1	000	01	0	00	6708
LW	0111	00	0	1	000	00	0	00	7100
sw	1000	00	0	1	000	01	1	00	810C
BEQ	1001	00	0	0	000	01	0	10	900A
J	1010	10	0	1	000	10	0	01	A911

Instruções detalhadas:

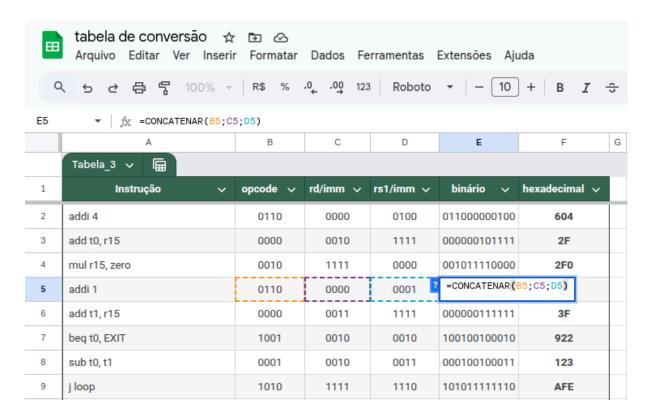
ADD	RD <- RD + RS1
SUB	RD <- RD - RS1
MUL	RD <- RD * RS1
DIV	RD <- RD % RS1
AND	RD <- RD & RS1
OR	RD <- RD RS1
ADDI	R15 <- IMM
LW	RS1 <- V[R1]
SW	RS1 -> V[R1]
BEQ	RS1 == 0
JUMP	IMM

Código feito para testar as minhas instruções :

*Decrementa o T0, que tem o valor 4 até que seja zero, então sai do loop.

```
codigo.asm
          addi 4
                              # r15 ← 4
          add t0, r15
                             # t0 ← r15(4)
          mul r15, zero
                              # zera r15
          addi 1
                              # r15 ← 1
          add t1, r15
                              # t1 \leftarrow r15(1)
          beq t0, EXIT
                              # t0 = 0 ? -> EXIT (pula 2)
          sub t0, t1
                              # t0 - t1(1)
                              # pula -2 instrucoes
          j loop
```

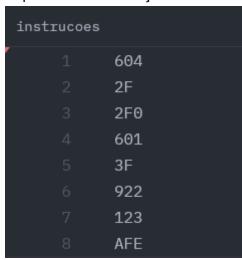
Criei uma planilha com fórmulas para traduzir automaticamente as instruções. Nela, inseri os 12 bits, e a planilha os converte para binário e hexadecimal.



Em momentos em que essa planilha não funcionava devido ao excesso de bits, que atingia o limite das fórmulas, utilizei este site para a conversão:

https://www.rapidtables.com/convert/number/binary-to-hex.html?x=100000100001

Arquivo com as instruções traduzidas:



Arquivo com os sinais de controle da UC:

```
unidade de controle
1 0: 108 1128 2148 3168 4188 51A8 6708 7100 810C 900A A911
```