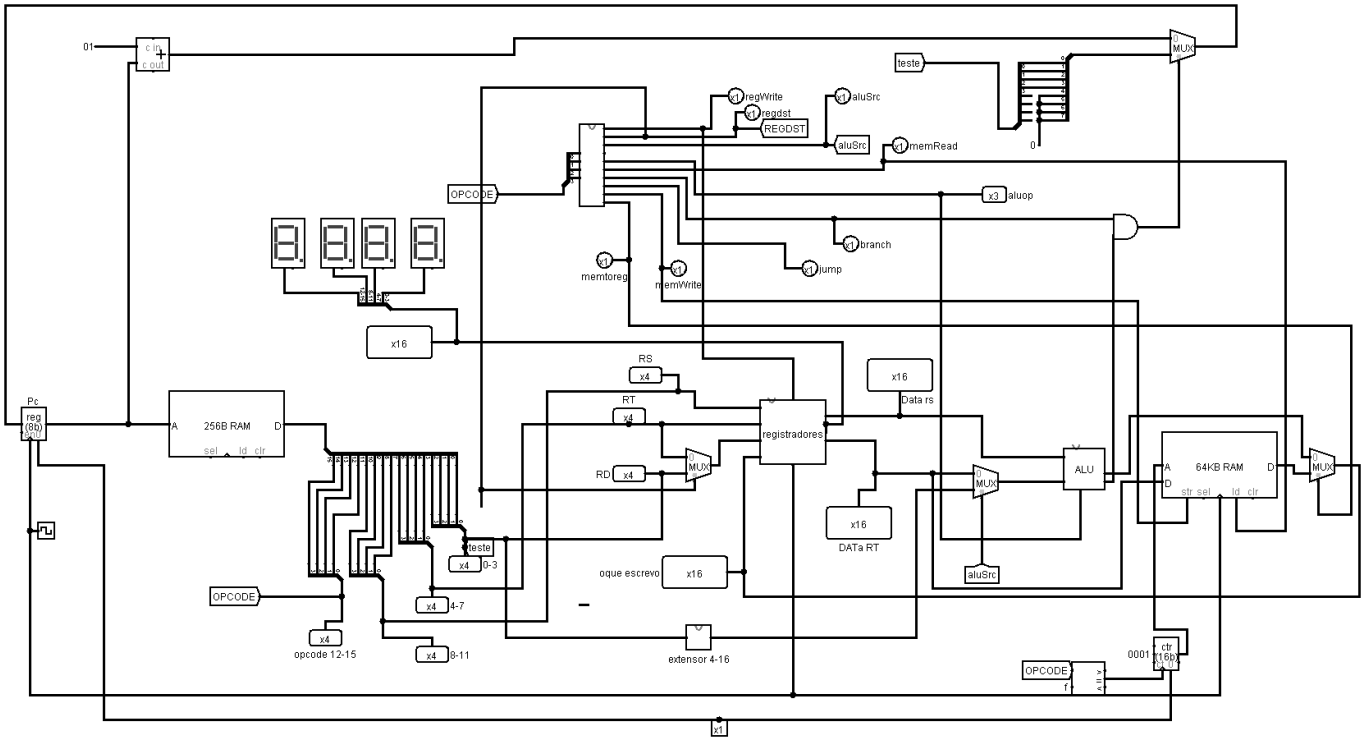
HELL OUTSIDE

Main:

Segue abaixo o nosso processador Hell Outside de 16 bits, que foi inspirado no MIPS monociclo.



Link da imagem no drive:

https://drive.google.com/file/d/1UtQxjOZpt5LtWSuWm8XEYhi8k2bbxWBI/view?usp=sharing

Instruções e seus tipos:

Todas as instruções seguem basicamente o mesmo padrão de 4 bits por código de operação

Sendo o padrão de

Opcode : Bit 15 - 12

Rs: Bit 11 - 8

Rt: Bit 7 - 4

Rd / Imediato / Address : Bit 0 – 3

Segue abaixo exemplo de instruções as divisões dos bits em cada instrução e sua referência em hexadecimal

addi				
Opcode	**Rs**	**Rt**	**Imediato**	
4	**4**	**4**	**4**	
addi	**$t0**	**$t0**	**4**	**hex decimal**
0001	**0001**	**0001**	**0100**	**1114**

slt				
Opcode	**Rs**	**Rt**	**Rd**	
4	**4**	**4**	**4**	
slt	**$t2**	**$t3**	**$t4**	**hex decimal**
0100	**0011**	**0100**	**0101**	**4345**

bne				
Opcode	**Rs**	**Rt**	**Addres**	
4	**4**	**4**	**4**	
bne	**$t1**	**$tzero**	**loop/addres**	**hex decimal**
0101	**0010**	**0000**	**1111**	**520f**

lw				
Opcode	**Rs**	**Rt**	**Imediato**	
4	**4**	**4**	**4**	

lw	$t0	$t3	0	hex decimal
0111	0001	0100	0000	7140

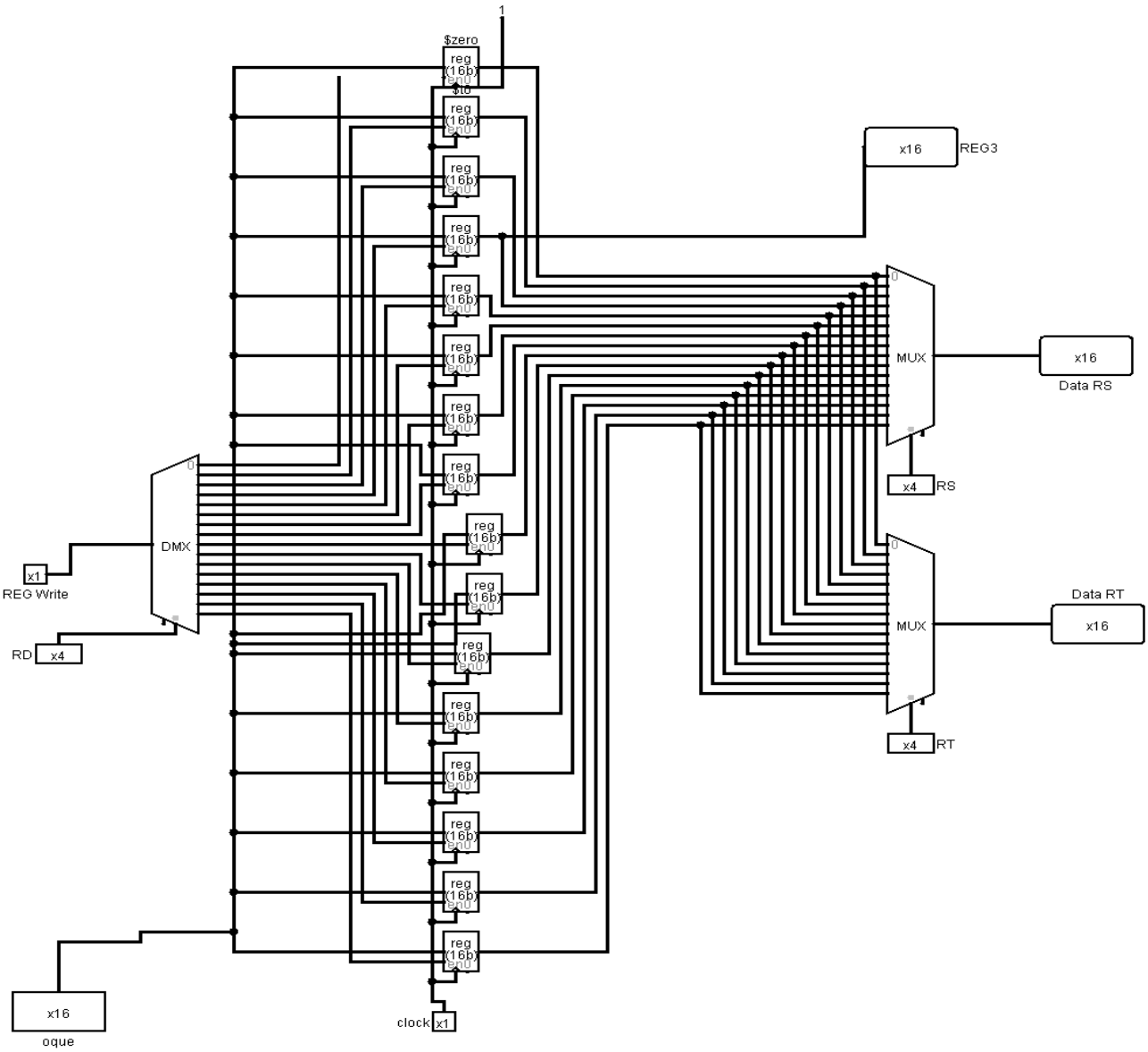
movn				
Opcode	rs	Rt	Rd	
4	4	4	4	
movn	t3	t4	t2	hex decimal
0111	0100	0101	0011	7453

Banco de Registradores:

Nosso banco de registradores tem um total de 16 registradores, sendo 1° o $zero, depois 5 registradores são temporários, 4 para salva dados. Além de ter 6 registradores há definidos para outros usos retorno de funções, jump, entre outros.

Registrador			
Número	Name	Uso	Valor bit
0	$ZERO	constante 0	0000
1	$t0	Temporário	0001
2	$t1	Temporário	0010
3	$t2	Temporário	0011
4	$t3	Temporário	0100
5	$t4	Temporário	0101
6	$s0	Save	0110
7	$s1	Save	0111
8	$s2	Save	1000
9	$s3	Save	1001
10			1010
11			1011
12			1100
13			1101
14			1110
15			1111

Segue abaixo os 16 registradores as entradas e saídas do Banco de registradores:


1
$zero
reg (16b)
$10
x16
REG3
MUX
x16
Data RS
x4
RS
DMX
x1
REG Write
RD
x4
MUX
Data RT
x16
x4
RT
x16
oque
clock
x1

Unidade de controle:

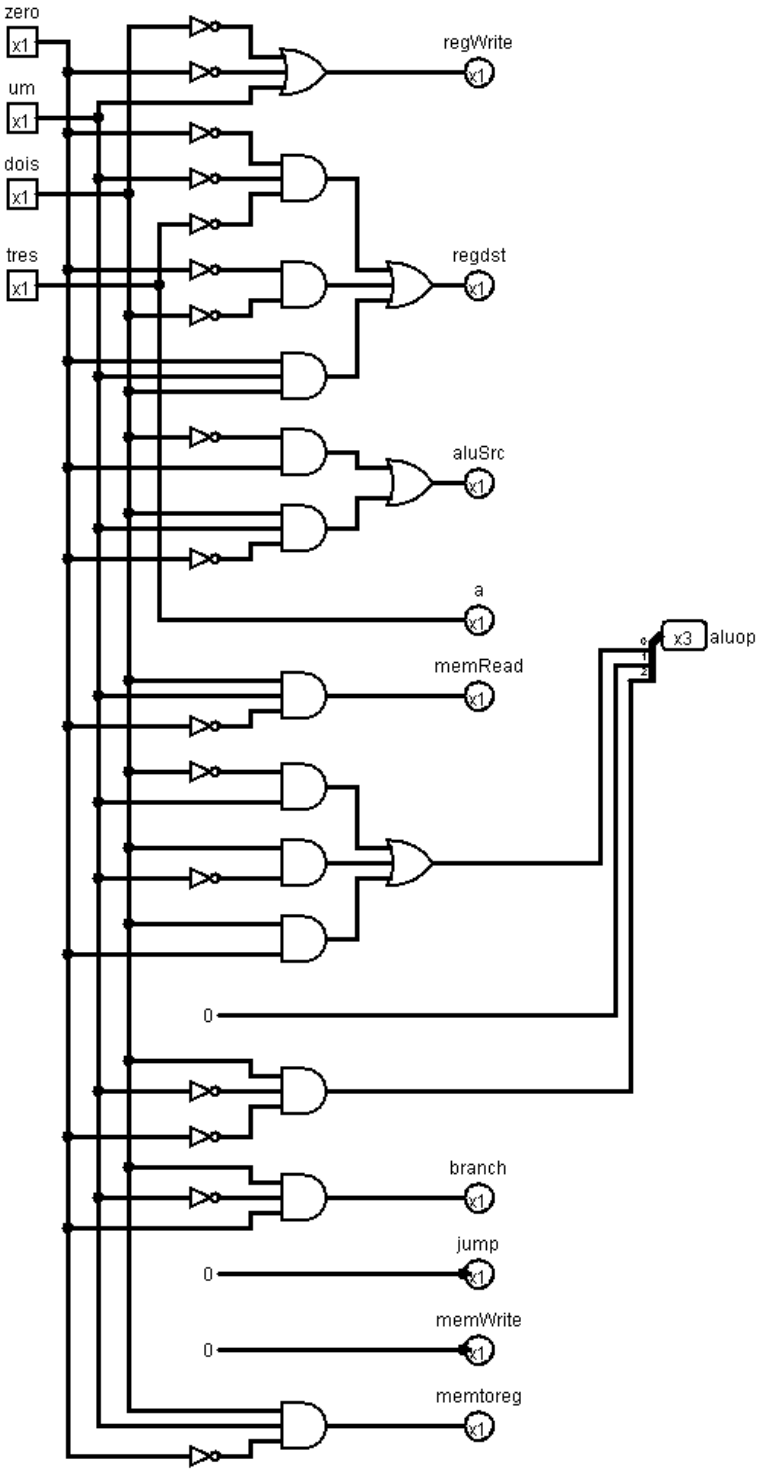
A nossa unidade de controle tem um total 8 instruções sendo elas: add, Addi, Sub, Subi, Slt, Bne, Lw, movn. Com a opção de expansão de instruções como jump e mais instruções, abaixo segue a nossa tabela verdade da unidade de controle

Unidade de controle										
instrução	Opcode	Reg write	Regdst	Alu Src	mem read	aluop	Branch	jump	mem Write	memtoreg
add	0000	1	1	0	0	000	0	0	0	0
addi	0001	1	0	1	0	000	0	0	0	0
sub	0010	1	1	0	0	001	0	0	0	0
subi	0011	1	0	1	0	001	0	0	0	0
slt	0100	1	1	0	0	101	0	0	0	0
bne	0101	0	0	0	0	001	1	0	0	0
lw	0110	1	0	1	1	000	0	0	0	1
movn	0111	1	1	0	0	001	0	0	0	0
	1000									
	1001									
	1010									
	1011									
	1100									
	1101									
	1110									
	1111									

Unidade de controle:

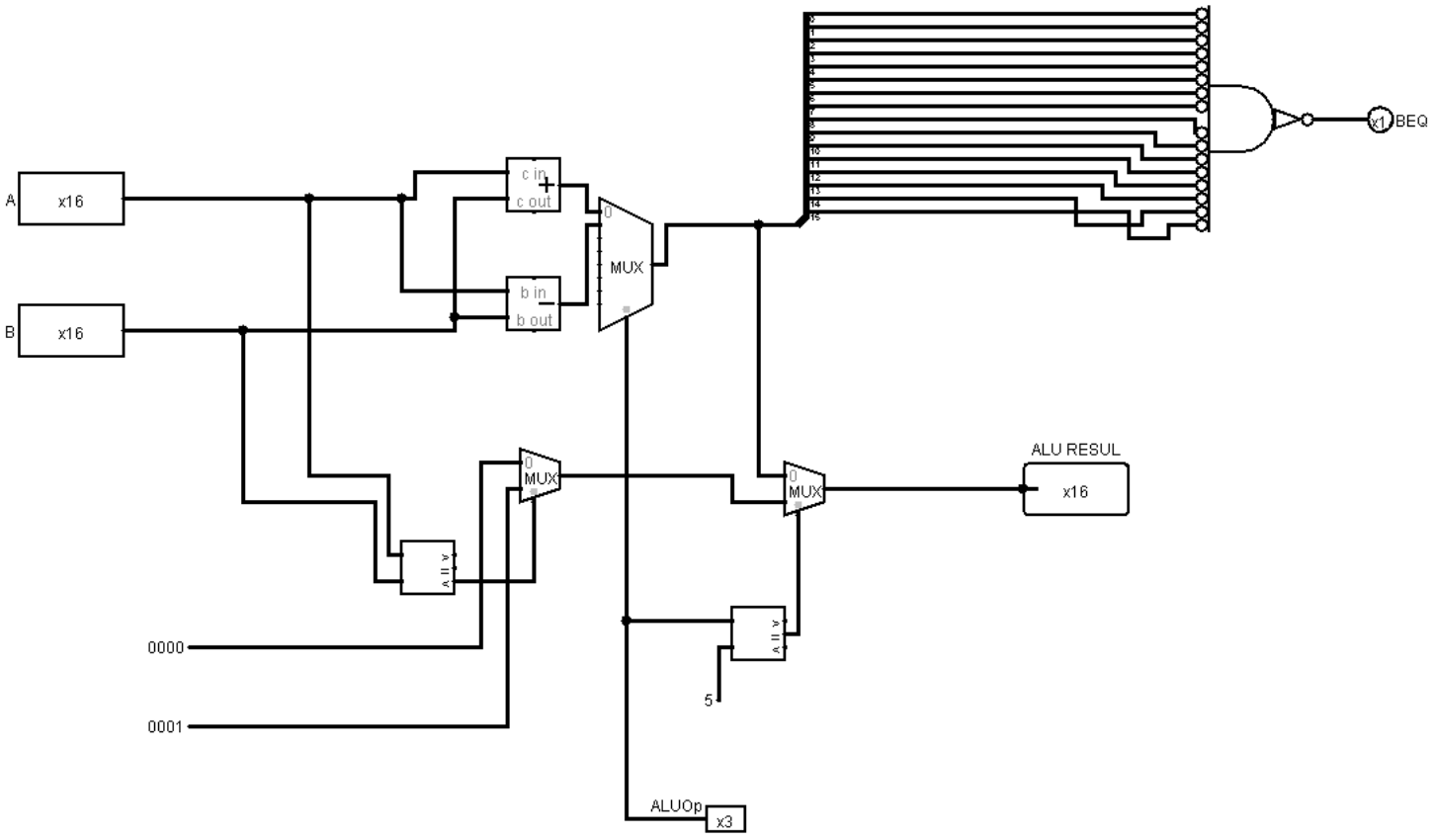
Recebe 4 bits do Opcode e resulta nas saídas da unidade de controle, foi usada a tabela verdade do logisim gerou nossa unidade de controle, que no momento o jump e o memWrite não são usados e ficam como

constante 0. segue abaixo imagem da unidade de controle do Logisim.



ALU:

Segue abaixo nossa ALU, que não precisou de uma alu-control apenas recebe da unidade controle os valores em 3 bits da aluop, que faz as operações de somar, subtrair, comparar bits e recebe o valor da aluop.



Memória de instrução:

A codificação para memória funciona em hexadecimal, desse modo temos as instruções em binário transformada em hexadecimal e carregada na memória. Abaixo estão nossas intruções em binário e sua versão em hexadecimal

Instruções	
Binário	**Hexadecimal**
0001 0000 0010 0101	**1027**
0001 0000 0001 0001	**1011**
0110 0011 0011 0011	**6333**
1111 0000 0000 0000	**f000**
0110 0100 0100 0100	**6444**
0100 0011 0100 0101	**4345**
0101 0101 0000 1010	**550a**
0011 0010 0010 0001	**3221**
0101 0010 0001 0011	**5213**
0101 0010 0000 1111	**520f**
0001 0100 0011 0000	**1430**
0101 0010 0001 0111	**5217**

Descrição das instruções do Programa:

1027 addi $2 = $0 + 7; guarda a quantidade de valores do vetor no registrador 2

1011 addi $1 = $0 + 1

6333 lw %3 = mem; carrega o primeiro valor do vetor em $3

f000 atualiza o valor atual do vetor para ser o próximo

6444 lw $3 = mem; carrega o valor atual em $4

4345 slt $5 = ($3 < $4) ? 1 : 0

550a bne; pula para a instrução no endereço a se $5 != $0

3221 subi $t2 = $t2 – 1; decrementa o contador de iterações restantes

5213 bne; pula para a instrução no endereço 3 se $2 != $1

520f bne; pula para a instrução no endereço f se $2 != $0

1430 addi $3 = $4 + 0; atualiza o maior elemento se necessário

5217 bne; pula para a instrução no endereço 7 se $2 != $1

Exemplo:

Utilizando os valores do vetor, em hexadecimal: 2 1 5 3 d a 6., o programa exibe o valor d (13 em decimal) no registrador $3.

Caso queira alocar mais valores no vetor, executar a quantidade de addi necessário ou guardar a quantidade na memória e executar um lw, guardando sempre no registrador $2.

Conclusão:

Nosso Hell Outside consegue executar a busca de maior elemento de um vetor com sucesso. Como um processador completo, porém, ele é falho. Instruções e pseudo instruções presentes no MIPS (processador que inspirou a criação do Hell Outside) não existem no Hell Outside pois não foram necessárias para a tarefa exigida.