Relatório do Microprocessador

Professor: Giovanni Venâncio
Disciplina: Projetos Digitais e Microprocessadores
Nome: Gabriel Henrique Polo GRR: 20241558

1. Componentes utilizados e problemas

Trabalho realizado na versão 3.8 do Logisim.

Componentes prontos usados: ROM, MUX. DEMUX, multiplicador de 32 bits, distribuidor, tunel.

Componentes criados: Flip Flop tipo D; Registradores; Banco de Registradores; Extensão de sinal e de zero; Detector de Zero; Somador, subtrador e complemento de 32 bits; Somador e registrador de 24 bits; PC.

Estado Indeterminado: o programa inicialmente está indefinido, então para o correto funcionamento, realizei a criação do botão reiniciar. O qual está interligado no Banco de Registradores, no PC, e no Display.
Banco de Registradores: para seu funcionamento adequado, recebo o endereço de a, b e c além do sinal de instrução de escrita (W), o clock e a entrada do botão reiniciar.
Operação Branch e Jump: para poder alterar o PC com um desse comandos, criei um mux no qual o Jump é ativado se o sinal sair da memória de controle. Para o Branch é necessário também que seja detectado na ULA que esteja saindo um valor nulo.
Operação Halt: Para conseguir fazer com que o PC parasse, o Halt interfere com a escrita do registrador de 24 bits, por meio de um mux. caso ele seja ativado a memoria que será escrita é 0 com isso o processador para de ler novas memorias de instrução, a não ser que seja resetado.
Operação Print: para eu conseguir fazer o display aparecer o valor, criei o sinal de controle print, o qual através de um mux (com o seletor sendo o botão de reiniciar), ele ativa o seletor do meu registrador e atualiza o novo valor.

2. Formato da Instrução e Sinais de Controle

Opcode	a	b	c	Imed
4 bits	4 bits	4 bits	4 bits	16 bits

Para controle foram criados 7 sinais de controle:

W	Halt	Imed	Jump	Branch	Print	Funct
1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	3 bits

O Funct é o único que precisa de mais bits para coordenar as funções da ULA.

3. Memoria de Controle

Instrução	Opcode	W	Halt	Imed	Jump	Branch	Print	Funct
nop	0000	0	0	0	0	0	0	000
li c, imm	0001	1	0	1	0	0	0	000
Add c, a, b	0010	1	0	0	0	0	0	000
Sub c, a, b	0011	1	0	0	0	0	0	001
Mult c, a, b	0100	1	0	0	0	0	0	010
And c, a, b	0101	1	0	0	0	0	0	011
Or c, a, b	0110	1	0	0	0	0	0	100
Xor c, a, b	0111	1	0	0	0	0	0	101
Sll c, a, b	1000	1	0	0	0	0	0	110
Addi c, a, imm	1001	1	0	1	0	0	0	000
Subi c, a, imm	1010	1	0	1	0	0	0	001
Jump imm	1011	0	0	1	1	0	0	000
Branch a, b, imm	1100	0	0	0	0	1	0	101
Show a	1101	0	0	0	0	0	1	000
Halt	1110	0	1	0	0	0	0	000

4. Código em C convertido para Assembely:

```
addi r1, r0, 10    // N = 10
addi r2, r0, 7     //  a = 7
addi r3, r0, 18    // d = 18
addi r10, r0, 0    // soma = 0
addi r4, r0, 0     // inicio i = 0
loop:
branch r4, r1, 7  // condição if (i >= N) sai do loop
mult r5, r4, r3    // i  * d
add r6, r2, r5     //  a + i * d
add r10, r10, r6  // soma += (a + i * d)
show r10          // printf da soma
addi r4, r4, 1     // i++
jump -6           // jump para for
show r10          // printf da soma
halt              //fim do algoritmo
```

4.1 Conversão de Assembely para Binário

Assembely	Mem. Instrução	Opcode	a	b	c	Imed
addi r1, r0, 10	0x000000	1001	0000	0000	0001	0000000000001010
addi r2, r0, 7	0x000001	1001	0000	0000	0010	0000000000000111
addi r3, r0, 18	0x000002	1001	0000	0000	0011	0000000000010010

addi r10, r0, 0	0x000003	1001	0000	0000	1010	0000000000000000
addi r4, r0, 0	0x000004	1001	0000	0000	0100	0000000000000000
branch r4, r1, 7	0x000005	1100	0100	0001	0000	0000000000000111
mult r5, r4, r3	0x000006	0100	0100	0011	0101	0000000000000000
add r6, r2, r5	0x000007	0010	0010	0101	0110	0000000000000000
add r10, r10, r6	0x000008	0010	1010	0110	1010	0000000000000000
show r10	0x000009	1101	1010	0000	0000	0000000000000000
addi r4, r4, 1	0x00000A	1001	0100	0000	0100	0000000000000001
jump -6	0x00000B	1011	0000	0000	0000	1111111111111010
show r10	0x00000C	1101	1010	0000	0000	0000000000000000
halt	0x00000D	1110	0000	0000	0000	0000000000000000