Novas Instruções

Novas operações:

Opcode	Tipo	Mnemonic	Nome	Operação
0101	I	lui	Load Upper Imm.	R[0]= IMM(bits 4-7),R[0](bits 0-3).
0110	IR	addhi	Add Half Imm	R[ra] = R[ra] + Imm
0111	J	jal	Jump and Link	R[ra] = PC + 1, PC = R[rb]

Lui => Escolha feita para conseguir montar imediato de 8 bits de forma simples.

Addhj => Escolha feita para poupar um registrador para decrementos/incrementos pequenos.

Jal => Escolha feita para ter um controle melhor sobre saltos e chamada de funções.

Novos tipos:

Tipo IR							
Bits	7	6	5	4	3	2	1	0
	Opcode				**Ra**		**Imm.**	

Tipo J							
Bits	7	6	5	4	3	2	1	0
	Opcode				**Ra**		**Rb**	

Tabela U.C

Tabela U.C:

Opcode	DATA	IMM-JUMP	BRAN-CH	MEM-WE	BGWE	IMM-TYPE	ITYPE	SUB	JAL	OP	HEX
0000	XX	0	1	0	0	X	0	X	0	X	202
0001	XX	1	0	0	0	1	1	X	0	X	462
0010	00	0	0	0	1	X	0	X	0	X	82
0011	XX	0	0	1	0	X	0	X	0	X	102
0100	10	0	0	0	1	1	1	0	0	000	10e0
0101	10	0	0	0	1	1	1	X	0	111	10e7
0110	10	0	0	0	1	0	1	0	0	000	10a0
0111	01	0	0	0	1	0	0	X	1	X	88a
1000	10	0	0	0	1	X	0	X	0	110	1086
1001	10	0	0	0	1	X	0	0	0	001	1081
1010	10	0	0	0	1	X	0	0	0	010	1082
1011	10	0	0	0	1	X	0	0	0	011	1083
1100	10	0	0	0	1	X	0	0	0	000	1080
1101	10	0	0	0	1	X	0	1	0	000	1090
1110	10	0	0	0	1	X	0	0	0	100	1084
1111	10	0	0	0	1	X	0	0	0	101	1085

Códigos Assembly

Trabalho 1:

```
addi 5
or r2,r0
or r3,r0        #r2 = endereço de saida da função sde inicializar e valor para acessar vetores diferentes.
addi -3
slr r2,r0



add r3,r2
or r1,r2        #r1 = 20(elementos a serem inseridos)
addi -1         #r0 = 1 (elementos para incrementos e decrementos)
slr r3,r0       #r3 = endereços de alocamento do vetor (41-50 impar, 51-60 resultante, 61-70 pares)



loop_impar:
    sub r1,r0
    add r3,r2
    st r1,r3
    ji aux
loop_par:
    ji loop_impar           #Inicializa os vetores Par e Impar simultaneamente para aproveitar os
valores;
aux:
    sub r1,r0
    sub r3,r2
    st r1,r3
    sub r3,r0
```

```
    brzr r1,r2
    ji loop_par
srr r2,r0           #r2 = 10 para acessar o vetor resultante
add r3,r2           #ajustanto r3 para apontar no final do vetor


soma:
    ld r1,r3        #r1 = load Par
    add r3,r2       #r0 = load Impar
    add r3,r2
    ld r0,r3
    sub r3,r2
    add r0,r1
    ji aux1
aux_sum:
    ji soma
aux1:
    st r0,r3
    sub r3,r2
    xor r0,r0       #reajusta r0 = 1 para usar incremento/decremento;
    addi 1
    sub r3,r0
    brzr r1,r3      #quando r1 = 0, r3 = 40
    ji aux_sum
or r0,r0
or r0,r0
or r0,r0
ji 0;
```

Trabalho 2:

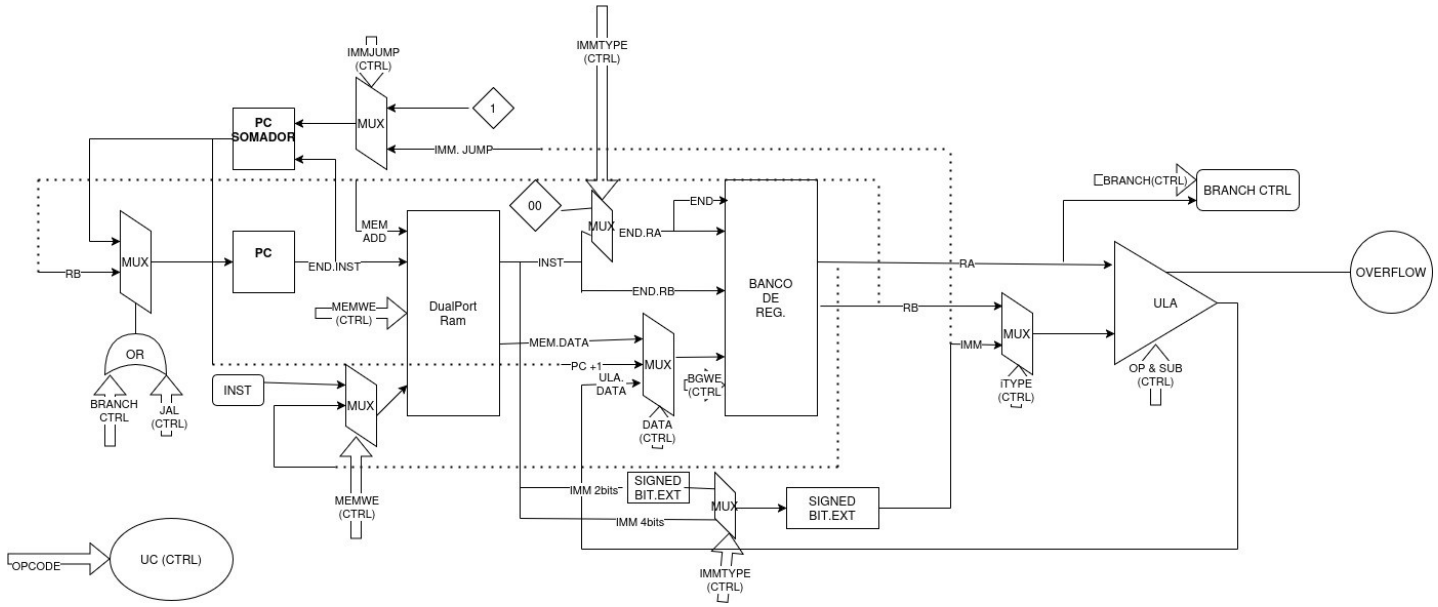
```
addi 4;
lui 1;    #r1 recebe 20 (valores a serem guardados);
or r1,r0;

not r0,r0;
lui 3;    #Calcula endereço de alocação dos vetores;
or  r2,r0;          #r2 = end de aloc

or r0,r0;
or r0,r0;
or r0,r0;    #Calcula endereço para função de inicialização;
or r3,r1  #r3 = end da func1 (20);

addi -1;
lui 0;    #r0 recebe 10 para tratar os endereços de alocação;

jal r3,r3;        #Pula para função que inicia vetores A e B e r3 recebe retorno;

addi 5;
lui 1;    #Calcula endereço para função de soma;
or r3,r0;     #r3 = end da func2;

addi -5;  # r0 rece 10
lui 0;



addhi r2,-1
jal r3,r3;    #Pula para função de soma;
```

```
ji 0;  #Encerra função;


func1:
    loop_impar:
        addhi r1,-1  #end 20
        st r1,r2;
        ji loop_par;

    aux0:
        sub r2,r0;
        ji loop_impar;

    loop_par:
        addhi r1,-1;
        add r2,r0;
        st r1,r2;
        addhi r2,-1;
        brzr r1,r3;
        ji aux0;

func2:
    load:
        ld r1,r2;  #end 31
        add r2,r0;
        ld r0,r2;        #r1 load impar, r0 load par;
        add r1,r0
        xor r0,r0
        ji store
```

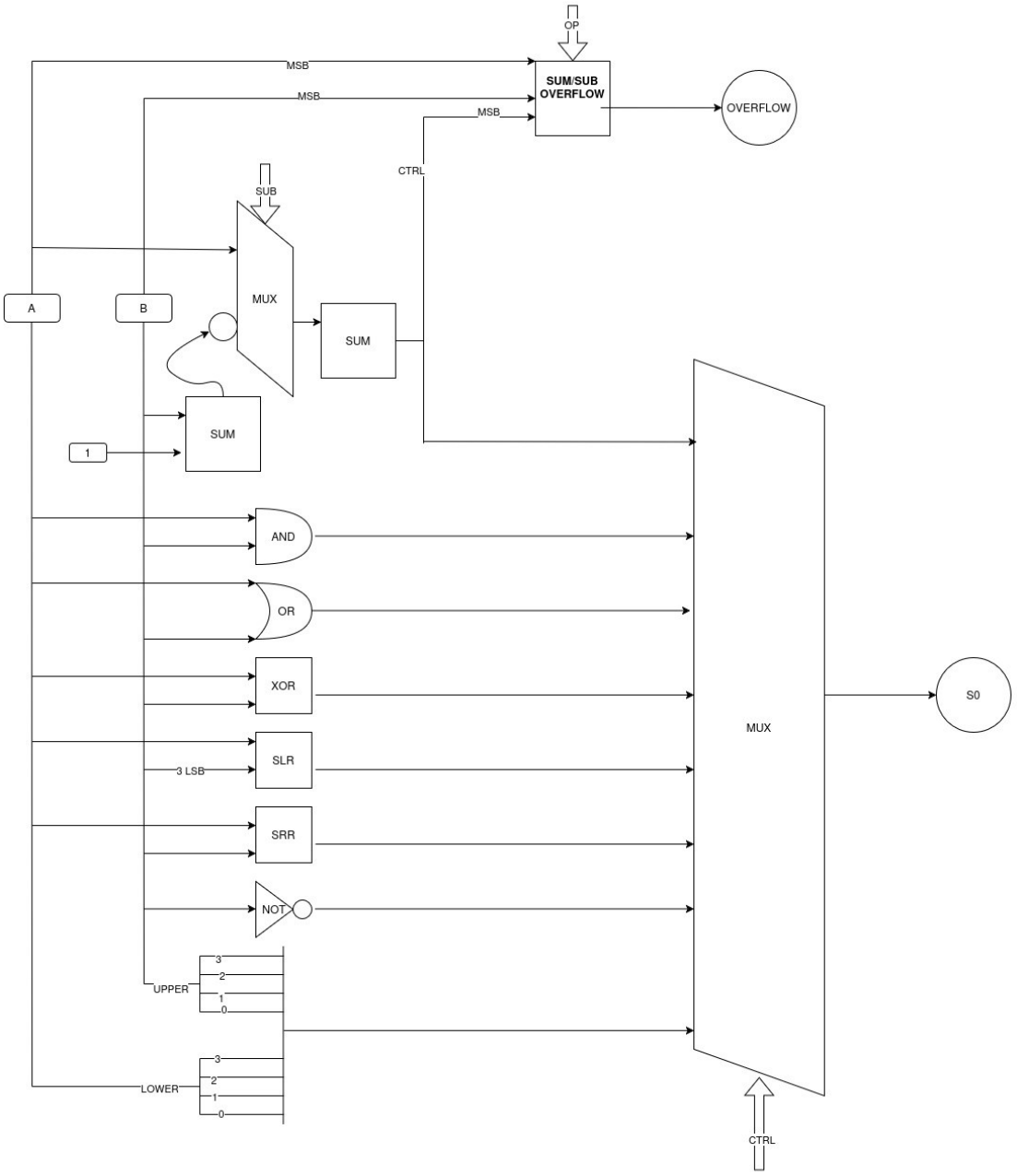
```
jump_load:
    ji load;

aux1:
    addhi r2,-1;
    sub r2,r0;
    sub r2,r0;
    ji jump_load;

aux2:
    ji aux1;

store:
    addi 10;                    #redefine r0 em 10;
    lui 0;
    add r2,r0;
    st r1,r2;
    addhi r1,-1
    brzr r1,r3;             #se r1 = 0, pula para encerramento.
    ji aux2;
```

DATAPATHS

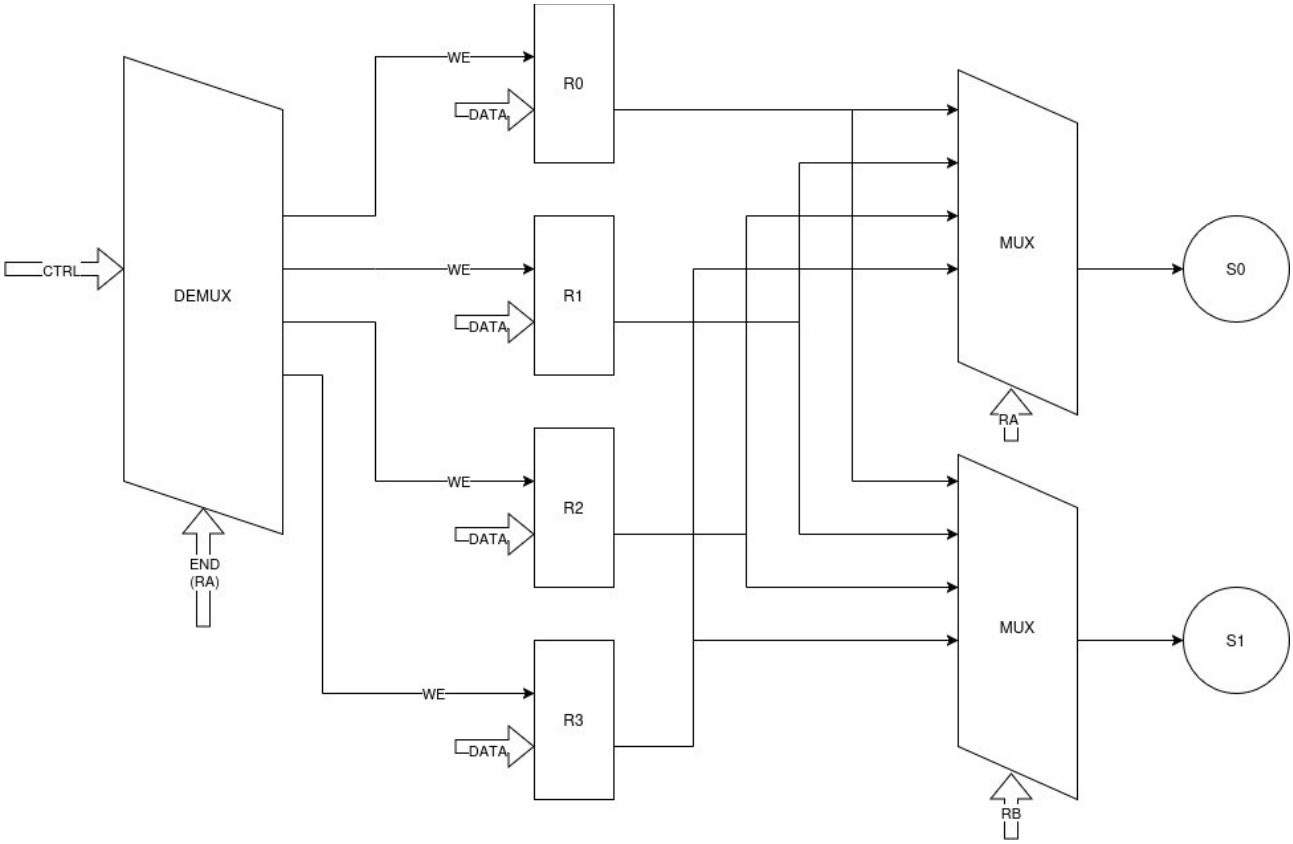
Processador:



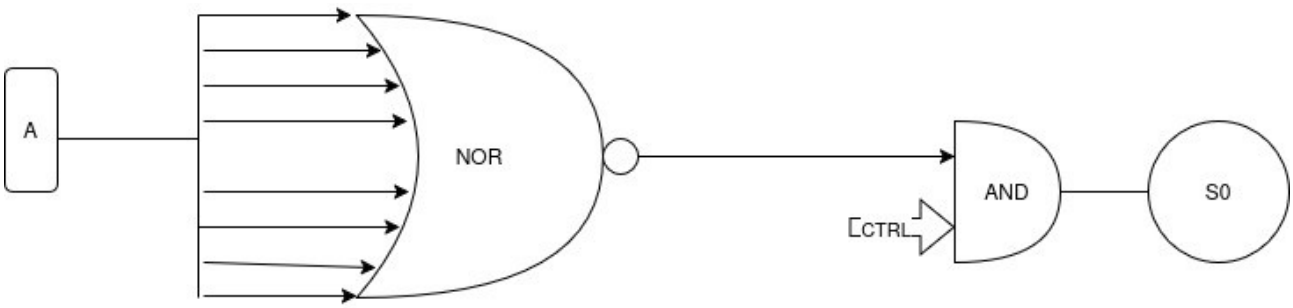
ULA:


OP
MSB
SUM/SUB
OVERFLOW
OVERFLOW
MSB
MSB
CTRL
SUB
MUX
A
B
SUM
SUM
1
AND
OR
XOR
SLR
3 LSB
SRR
NOT
3
2
1
0
UPPER
3
2
1
0
LOWER
MUX
S0
CTRL

Banco de Registradores:



Branch Ctrl:



Detector de Overflow(Soma e Sub):