TURMA B 2007/2

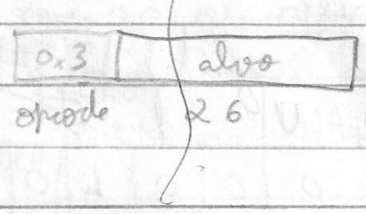
OAC-B

Prova 2

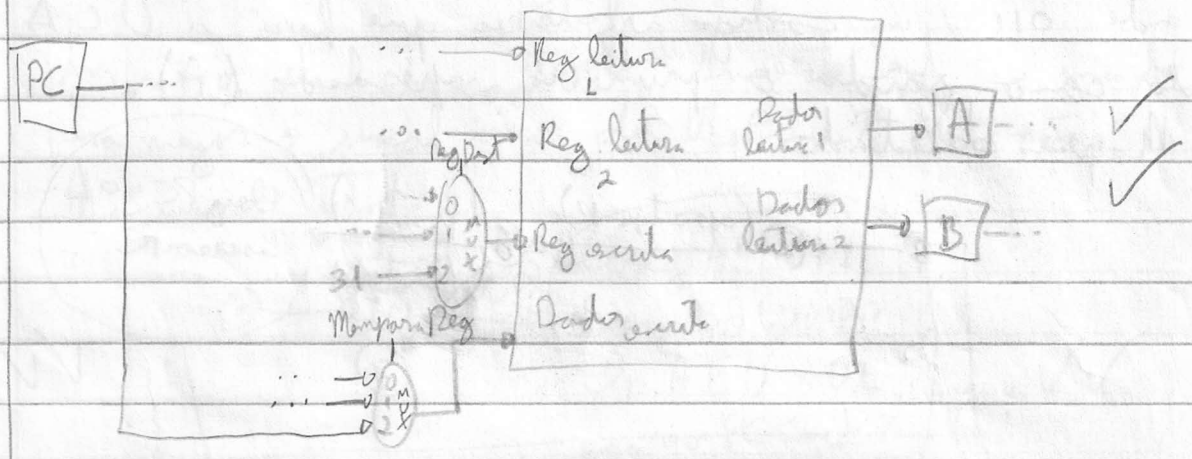
Gabarito

①

② jal LABEL

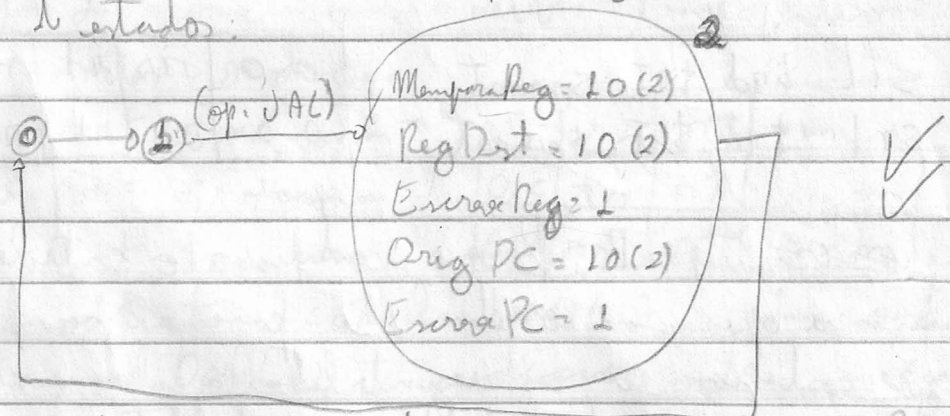


Caminho de dados:



Registrador de escrita pode receber tanto o valor 31 ($ra). Os dados de escrita no banco de registradores podem vir tanto do PC, RegDst e MemparaReg possuem 2 bits.

Maquina de estados:



O conteudo do PC é escrito em $31 ($ra). O PC recebe o endereço de jump.

(b) jr $rs

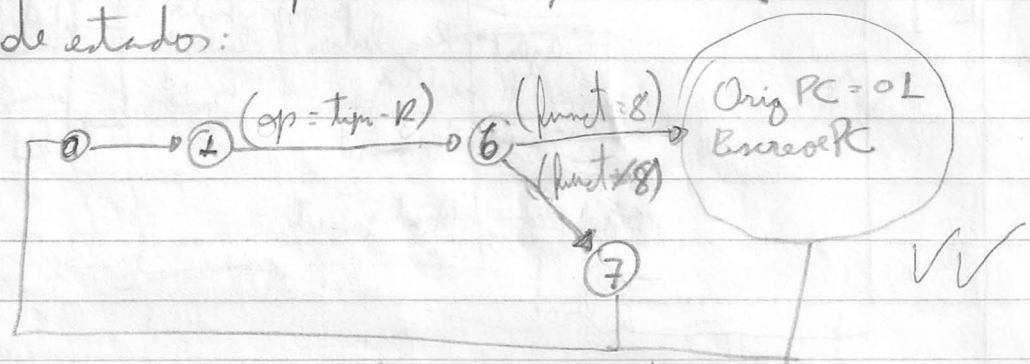
0x0	rs	0	8
opcode 5		15	funct

Operação tipo-R. Devemos modificar o controle da ULA para responder ao campo funct = 8. Desta maneira, adicionamos a seguinte linha à tabela verdade do controle da ULA

OpALU	funct						ALUop		
1	0	0	0	1	0	0	0	1	1

onde 011 é um código arbitrário que fará a ULA dar como saída o primeiro operando (A). ✓✓

Máquina de estados:



000 add
001 or
010 add
011 ✓
100
101
110 sub
111 slt

✓✓

Na finalização de qualquer tipo R, o resultado deve ser escrito no PC, não no banco de registradores. É necessário que a unidade de controle receba o campo funct.

(c) sll $rd, $rt, shamt
srl $rd, $rt, shamt

0x0	rs	rt	rd	shamt	0
0x0	rs	rt	rd	shamt	4
opcode 5	5	5	5	5	funct

Operações tipo-R. Devemos modificar o controle da ULA para realizar a operação correta para funct = 0 e funct = 4. Desta maneira, adicionamos as seguintes linhas à tabela verdade da ULA.

opALU	funct						ALUop		
1	0	0	0	0	0	0	1	0	0
1	0	0	0	0	1	0	1	0	1

✓✓

Onde 100 e 101 são códigos arbitrários para que a ULA realize, respectivamente, shift à esquerda e à direita no seu segundo operando (B), baseado no campo shamt da instrução, que deve ser uma entrada da ULA. Não há mudança na máquina de estados ✓✓.

(d) bne $rs, $rt, label

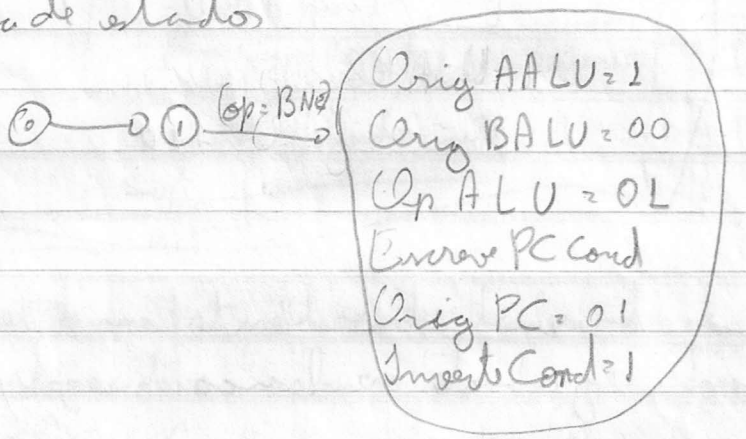
5	rs	rt	offset
opcode 5	5		16

Caminho de dados

✓✓

Um novo sinal de controle, InverteCond, é usado para negar o sinal zero da ULA nesta operação

Maquina de estados



✓✓

Na etapa de conclusão do desvio, basta ativar o sinal de inverter condição

(e)

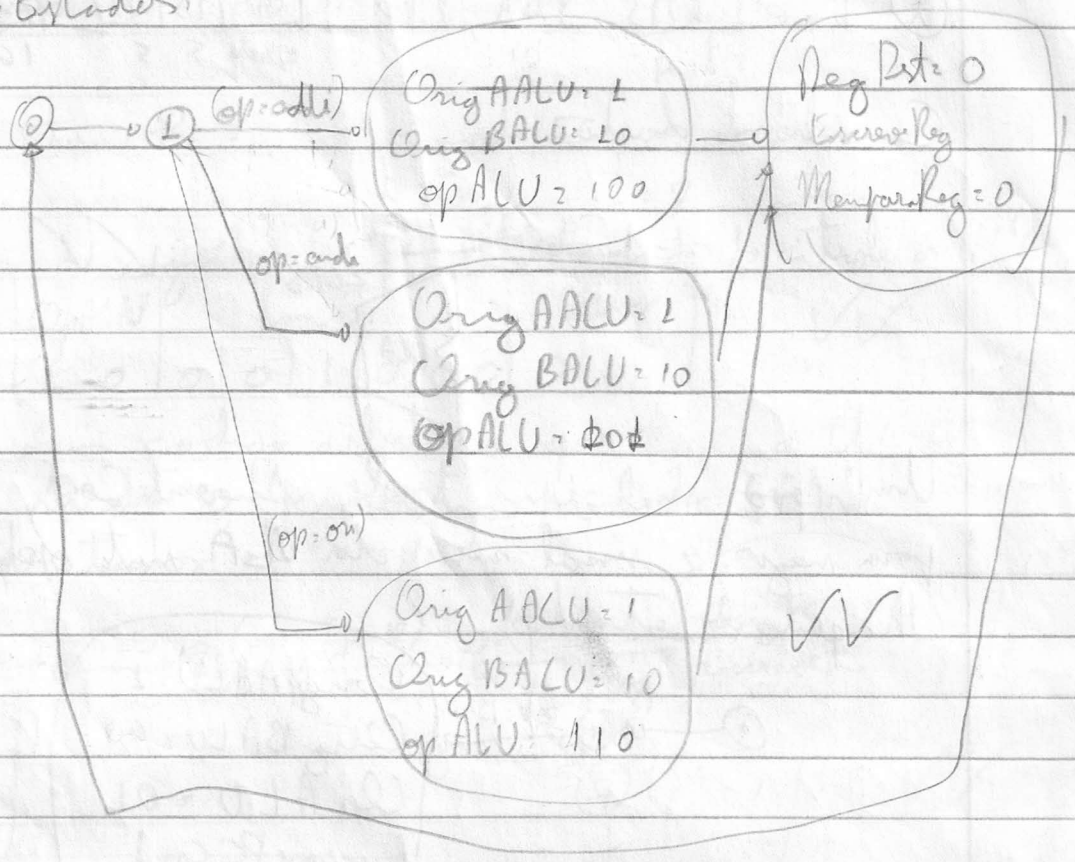
addi $rt, $rs, imm	0x8	rs	rt	imm
andi $rt, $rs, imm	0xc	rs	rt	imm
ori $rt, $rs, imm	0xd	rs	rt	imm

Devemos modificar o controle da ULA; adicionando 1 bit ao sinal OpALU e adicionando os seguintes bits a sua tabela verdade

	opALU	funct	ALUop
addi	1 0 0	x	0 0 0
andi	1 0 1	x	0 0 0
ori	1 1 0	x	0 0 1

✓✓

Maquina de Estados:



Cada operação tem um estado diferente para a terceira etapa mas todas gravam igualmente no banco de registradores no final

		Multiciclo	Predict c/ bolhas	Pipeline
② 1	Lw $s0, 10($fp) #s0=13	5	5	5
2	Lw $s1, 4($fp) #s1=8	5	1	1
3	add $t0, $s0, $s1 #t0=21	4	1+2	1+1
4	Lw $s2, 18($fp) #s2=3	5	1	1
5	sub $t0, $t0, $s2 #t0=18	4	1+2	1+1
6	slt $t1, $t0, $zero #t1=0	4	1+2	1
7	beq $t1, $zero, LOOP # V	3	2+1+3	1+1
8	sub $t0 $t0, $s2 t0=15	4	1	1
9	slt $t1, $t0, $zero t1=0	4	1+2	1
10	beq V	3	2+1+3	1+1
11	sub t0=12	4	1	1
12	slt t1=0	4	1+2	1
13	beq V	3	2+1+3	1+1
14	sub t0=9	4	1	1
15	slt t1=0	4	1+2	1
16	beq V	3	2+1+3	1+1
17	sub t0=6	4	1	1
18	slt t1=0	4	1+2	1
19	beq V	3	2+1+3	1+1
20	sub t0=3	4	1	1
21	slt t1=0	4	1+2	1
22	beq V	3	2+1+3	1+1
23	sub t0=0	4	1	1
24	slt t1=0	4	1+2	1
25	beq V	3	2+1+3	1+1
26	sub t=-3	4	1	1
27	slt $t1, $t0, $zero t1=1	4	1+2	1
28	beq F	3	2+1+3	1 → Previsão correta!
29	j MAIS	3	1+1	1+1
30	or $t1, $s3, $s5 ?	4	1	1

(a) Unicido:

IF + ID + EX + MEM + WB

lw: 210 + 30 + 120 + 210 + 70 = 640 ps

sw: 210 + 30 + 120 + 240 + 0 = 600 ps

R = 210 + 30 + 120 + 0 + 70 = 430 ps

Como todas as instruções levam o mesmo tempo para ser executadas, temos que o ciclo de clock é 640 ps

$t_{ex} = 30 \cdot 640 = 19200$ ps = 19,2 ms

(b) Maior etapa escrita na memória com 240 ps

Portanto, este é o nosso ciclo de clock

Todas as etapas levam 1 ciclo de clock para ser executada

Portanto

$t_{ex} = 1139 \cdot 240$ ps $= 273360$ ps $= 2736$ ms

(c) Maior etapa escrita na memória com 240 ps

Portanto, este é o nosso ciclo de clock

Idealmente temos a razão de 1 instrução / ciclo, mais a latência. Portanto:

$t_{ex} = (30 + 4) 240$ ps $= 8160$ ps $= 8,16$ ms

(d) linha 3: hazard de dados, leitura de $s1, Resolvido com 2 bolhas forwarding da etapa WB de lw para EX de add

linha 5: hazard de dados, leitura de $s2 na linha 4. Resolvido com 1 bolha e forwarding da etapa WB de lw para EX de sub. Só ocorre na primeira execução do laço

linha 6: hazard de dados, cálculo de $t0 na linha 5. Resolvido com forwarding da etapa MEM de sub

linha 7: hazard de dados, cálculo de $t1 na linha 6 Resolvido com forwarding da etapa MEM de slt

hazard de controle: desvio. Resolvido com previsão de desvio, e inserção de uma bolha, caso a previsão esteja errada. Cálculo do desvio realizado na segunda etapa

linha 8: jump: inserir 1 bolha

(e) Com inserção de bolhas, leva 94 ciclos para executar. Como o ciclo dura 240 ps:

$t_{ex1} = 94 \cdot 240 = 22560$ ps = 22,56 ns ✓✓

Com redução de hazards, leva 44 ciclos

$t_{ex2} = 44 \cdot 240 = 10560$ ps = 10,56 ns ✓✓

(f) Sim é possível. Da seguinte maneira, por exemplo

```
    lw s0, 10(fp)
    lw s1, 14(fp)
    lw s2, 18(fp)        ✓
    add t0, s0, s1
LOOP: sub t0, t0, s2
    slt t1, t0, zero
    beq t1, zero         ✓
    or t1, s3, s5
```

Pois atraso devido ao lw será eliminado, além da economia de um jump. Portanto, o código executa em 44 ciclos, ou ✓✓

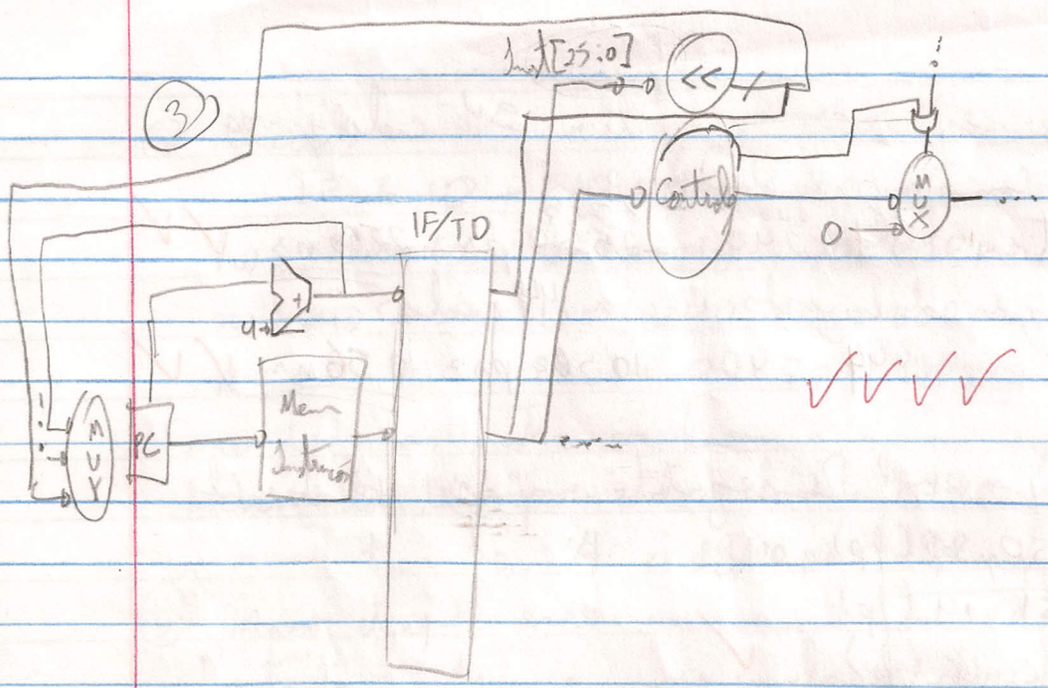
$t_{ex} = 40 \cdot 240 = 9600$ ps = 9,6 ns

(g) Unicido: 240 ps

Multiciclo: entre 720 ps e 1200 ps ✓

Pipeline: 1200 ps

Porque o pipeline executa instruções paralelamente, tendo uma razão de 1 instrução por ciclo, idealmente. Como o ciclo do pipeline dura menos da metade do tempo do ciclo da arq. unicido, o tempo de execução tende a ser menor, apesar do aparato contra duração ✓



√√√√

O endereço de jump para ser uma das entradas possíveis do PC junto com o endereço de desvio e PC+4. A unidade de controle gera um sinal extra para apagar a instrução que for estar no pipeline, da mesma maneira que o unidade de detecção de hazards faz.

√√√√