UFPR - DInf - Bacharelado em Ciência da Computação
CI210 - Projetos Digitais e Microprocessadores
Aula de revisão e exercícios para a segunda prova — 2020-pE2

Ex. 1 Traduza para *assembly* do MIPS o trecho de programa abaixo. Seu código *assembly* deve empregar as convenções de programação do MIPS. *Para facilitar a correção indique os registradores que não são dedicados em funções (*`a0,v0,ra`*) como* `re`, `rp`, *etc.*

```
1  #define NIL ((void *)0)
2
3  typedef struct elem {
4    struct elem *next;
5    int  vet[3];
6  } eType;
7
8  eType *head, *x;
9  eType str[256];
10   ...
11   x = insert(head, &(str[j]));
12   x->vet[1] = 512;
13   ...
```

```
14 eType
15   *insert(eType *h, eType *e) {
16
17   eType *p;
18
19   p = h;
20   while ((void *)p != NIL) {
21     p = p->next;
22   }
23   p = e;
24   e->next = NIL;
25   return (e);
26 }
```

Ex. 2 Traduza para *assembly* do MIPS o trecho de código C.

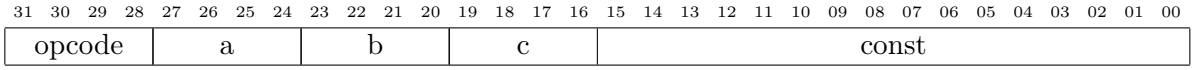
Para facilitar a correção indique os registradores como `ra`, `rb`, *etc.*

```
1  int a,b,i; int x[NNN], y[MMM], z[KKK];
2
3  a = x[10] + x[ y[3] ] + x[ y[ z[5] ] ];
4  i = a/4;
5  b = x[i] + x[ y[2*i] ];
```

Ex. 3 Mostre como implementar a instrução BRANCH-AND-LINK definida abaixo. Sua resposta deve conter:
(i) indicação clara das modificações no circuito;
(ii) a tabela com os sinais de controle;
(iii) um diagrama de tempos *completo* da execução desta instrução.
No comentário a vírgula significa "execução simultânea".

`bal desl` `# R[31]` $\leftarrow$ `PC+8 , PC` $\leftarrow$ `(PC+4)+(ext(desl)`$\ll$`2)` (formato I)

A Tabela **??** mostra o conjunto de instruções do Mico-X. A codificação das instruções, para o trabalho entregue, é mostrada abaixo.



Sua tarefa é alterar a codificação das instruções para que esta fique "mais parecida" com a codificação do conjunto de instruções MIPS32. As modificações são:

(i) a instrução **nop** mantém o opcode com 32 zeros;

(ii) as instruções **add,sub,mul,and,or,xor,not,sll,srl** passam para o opcode `b"0000"`;

(iii) a instrução **show** muda para **perif**, com um registrador e um endereço de 16 bits: `ES[E] ← R(a)`.
O vetor `ES[]` suporta até 2^{16} endereços de dispositivos periféricos (Entrada/Saída);

(iv) a instrução **jalr** (*jump and link register*) faz: `IP ← E , R(c) ← IP+1`;

(v) a instrução **jr** (*jump register*) faz: `IP ← R(a)`;

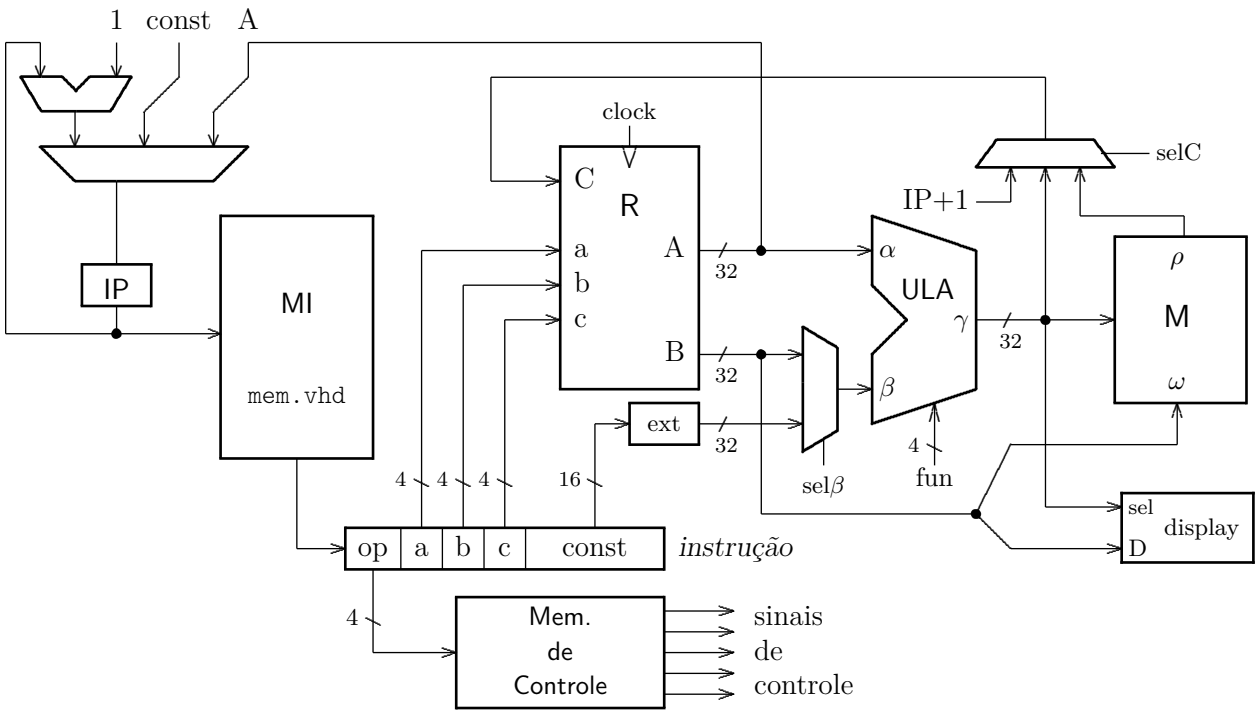
(vi) a instrução **bne** (*branch not-equal*) faz: `IP ← ((R(a)!=R(b))? E : IP+1)`

Tabela 1: Instruções do Mico X.

opcode	instrução	semântica	comentário
0	`nop`	`--`	*no operation*
1,2,3	`op c, a, b`	`R(c) ← R(a) op R(b)`	`op ∈ {add, sub, mul}`
4,5,6	`op c, a, b`	`R(c) ← R(a) op R(b)`	`op ∈ {and, or, xor}`
7	`not c, a`	`R(c) ← not(R(a))`	complemento
8	`sll c, a, b`	`R(c) ← R(a) ≪ R(b)`	deslocam. lógico *left* †
9	`srl c, a, b`	`R(c) ← R(a) ≫ R(b)`	deslocam. lógico *right* †
a	`ori c, a, K`	`R(c) ← R(a) ∨ extZero(K)`	∨ com constante lógica
b	`addi c, a, K`	`R(c) ← R(a) + extSinal(K)`	+ constante aritmética
c	`show a`	`display ← R(a)`	exibe valor na saída
d	`bal a, b, E`	`(R(a)==R(b))?`	
		`IP ← E , R(15) ← IP+1 : IP ← IP+1`	*branch-and-link*
e	`beq a, b, E`	`IP ← ((R(a)==R(b))? E : IP+1)`	desvio condicional
f	`halt`	`IP ← IP + 0`	termina a simulação
			† Completa com zero(s).

Ex. 4 Mostre a tabela com a nova codificação das instruções.

Ex. 5 Mostre a nova tabela de controle.


1
const
A
clock
C
R
a
b
c
A
B
32
32
32
selC
IP+1
IP
MI
mem.vhd
ext
32
α
ULA
β
γ
32
ρ
M
ω
4
fun
selβ
4
4
4
16
sel
display
D
op
a
b
c
const
instrução
4
Mem.
de
Controle
sinais
de
controle