nRisc MIPS

Kayky Moreira Praxedes, Carlos Ernesto Cardoso dos Reis

1. **Tema do projeto:**

Trata-se de um projeto de um processador nRisc monociclo com instruções de 8 bits que rodará um programa embarcado de cifra de césar simulado em Verilog HDL.

2. **Instruções em binário e Assembly:**

- Tipo 2R

Instruções	OpCode	Registrador 1	Registrador 2	funct
add	000	00 - 11	00 - 11	0
sub	000	00 - 11	00 - 11	1
mult	010	00 - 11	00 - 11	*don't care* (0)
ld	011	00 - 11	00 - 11	0
st	011	00 - 11	00 - 11	1
beq	101	00 - 11	00 - 11	0
slt	101	00 - 11	00 - 11	1

```
1 - add $c0, $c1          $c0 = $c0 + $c1
2 - sub $c0, $c1          $c0 = $c0 - $c1
3 - mult $c0, $c1         $c0 = $c0 * $c1
4 - ld $c0, $c1           $c0 = memória[$c1]
5 - st $c0, $c1           memória[$c1] = $c0
6 - beq $c0, $c1          $re = $c0 == $c1 ? 1 : 0
7 - slt $c0, $c1          $re = $c0 < $c1 ? 1 : 0
```

- Tipo 1R

Instruções	OpCode	Registrador	Imediato
addi	001	00 - 11	Imediato com sinal (3 bits)
jr	100	00 - 11	*don't care* (000)
result	110	00 - 11	Imediato com sinal (3 bits)

```
8  - addi $c0, i          $c0 = $c0 + i
9  - jr $c0               PC = $c0
10 - result $c0, i        $PC = $re == i ? $c0 : PC + 1
```

- Tipo 0R

Instruções	OpCode	*don't care*	funct
nop	111	*don't care (0000)*	0
hlt	111	*don't care (0000)*	1

```
11 – nop                  Ignora a linha (PC = PC + 1)
12 - hlt                  finaliza o programa
```

`i` é um imediato entre `-3 (11111101)` e `3 (00000011)`

3. Registradores:

Todos os registradores, de uso geral e reservados, possuem 8 bits.

- Uso geral:

```
00: r1 = $c0
01: r2 = $c1
10: r3 = $c2
11: r4 = $c3
```

- Reservados pelo processador:

`$re`: armazena o resultado de instruções de comparação (*`slt`* e *`beq`*) assumindo o valor 1 caso a comparação seja verdadeira, 0 se a comparação for falsa e 11111100 (-4) por padrão.

4. Sinais de controle:

- OpCode

Instrução	Op2	Op1	Op0	funct
add	0	0	0	0
sub	0	0	0	1
addi	0	0	1	X
mult	0	1	0	0
ld	0	1	1	0
st	0	1	1	1
jr	1	0	0	0
beq	1	0	1	0
slt	1	0	1	1
result	1	1	0	0
nop	1	1	1	0
hlt	1	1	1	1

- Flags do controle (binário):

Instrução	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	Jump	NextOp	End	ALUOp
0000 add	0	0	1	0	0	0	0	0	0	00
0001 sub	0	0	1	0	0	0	0	0	0	00
0010 addi	1	0	1	0	0	0	0	0	0	01
0011 addi	1	0	1	0	0	0	0	0	0	01
0100 mult	0	0	1	0	0	0	0	0	0	10
0101 **SEM OP**	X	X	X	X	X	X	X	X	X	01
0110 ld	0	1	1	1	0	0	0	0	0	01
0111 st	0	0	0	0	1	0	0	0	0	01
1000 jr	0	0	0	0	0	0	1	0	0	01
1001 **SEM OP**	X	X	X	X	X	X	X	X	X	01
1010 beq	0	0	0	0	0	0	0	0	0	11
1011 stl	0	0	0	0	0	0	0	0	0	11
1100 result	1	0	0	0	0	1	0	0	0	00
1101 **SEM OP**	X	X	X	X	X	X	X	X	X	01
1110 nop	X	X	0	0	0	0	0	1	0	01
1111 hlt	X	X	0	0	0	0	0	0	1	01

- `ALUSrc`: Indica se o segundo operando vem de um registrador (0), ou se trata de um imediato (1).

- `MemToReg`: Indica se o resultado a ser escrito no registrador vem da ULA (0) ou se vem da memória (1).

- `RegWrire`: Indica se vai haver escrita no registrador (registrador 1) (1) ou não (0).

- `MemRead`: Indica se vai haver leitura de memória (1) ou não (0).

- `MemWrite`: Indica se vai haver escrita na memória (1) ou não (0).

- `Branch`: Indica se vai haver um desvio condicional (1), ou não (0)

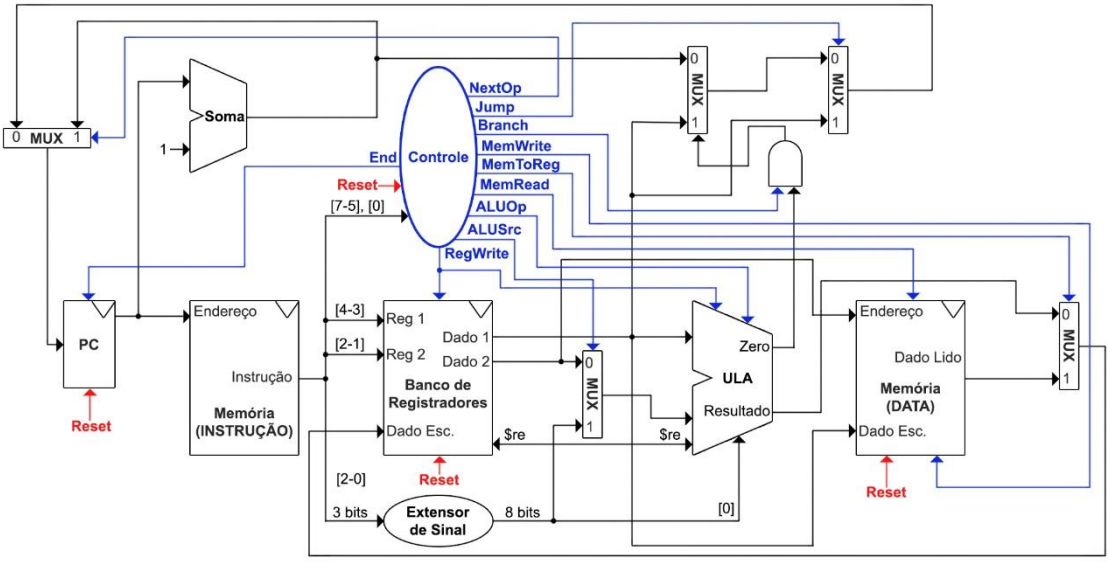
- `Jump`: Indica se vai haver um desvio incondicional (1), ou não (0)

- `NextOp`: Ignora a linha instrução.

- `End`: Encerra o programa.

- `ALUOp1` e `ALUOp0`: 2 bits que indicam qual vai ser o tipo de operação a ser realizado pela ULA: `add, sub ou result` (00), `addi` (01), `mult` (10) e `beq ou slt` (11).

5. Diagrama do nRisc:



6. Mudanças e atualizações de versões anteriores:

- v1:

- Foi retirada a instrução “div” (não tinha função para a cifra de césar).

- Depois de ser usado o result, o valor padrão guardado em $re passa a ser -4, não mais 0 (caso contrário poderia ser feito desvio condicional pelo método padrão, e desvio incondicional se usado “result $cn, 0”);

- Antes a instrução store (st) tinha modelo “st r1, r2” para armazenar na memória[r1 + 128] = r2. Agora, tanto na instrução load (ld) quanto na instrução store (st), o segundo registrador é o referente ao endereço de memória a ser acessado.

- v2:

- Separação dos flags Branch em Branch e Jump, e EndOrNext em NextOp e End.

- O controle agora recebe os bits [7, 5], [0] (3 bits de OpCode e um bit de funct).

- A ULA agora também recebe o RegWrite para a função result.

- v3:

- A memória é separada em 128 bits de instrução e 128 bits para data (sem necessidade do registrador $sp (Stack Point)).

- v4:

- Endereço da memória a ser acessado na memória de dados é achada pelo Dado1, não pela saída da ULA (mudança no diagrama da ULA completo).

- v6:

- Mudança nos flags do Controle (casos XX do ALUOp sendo trocados por 01, afim de evitar casos 00 e 11 nos casos XX, já que eles alteram o valor do $re, causando comportamento indesejado).

- Correções na ULA (o result não estava sendo corretamente calculado).

- Os códigos em Verilog dos componentes estão junto dos testbenchs e dos diagramas.

- Correções no código em binário e assembly que estavam funcionando de maneira inesperada quando necessária a correção do range da letra.

- v7:

- Correção no diagrama do nRisc (não mostrava a conexão do $re com a ULA e com o banco de registradores).

- Corte de redundâncias no pdf.