µProcessador 6 Condicionais e Desvios

Inclua no circuito instruções de desvio condicional e não condicional, de acordo com o *assembly* do processador escolhido. Alguns chamam de *jump*, outros de *branch*. A maioria inclui tanto instruções com endereço de salto absoluto[1] quanto com endereço de salto relativo[2].

O processador deverá ser capaz de comparar dois números e descobrir qual deles é o menor. Pode alterar sua ULA se quiser.

Cuidado: o *debug* desta prática pode ser *brutal*, apesar da implementação ser simples. Mantenha a ordem no VHDL, no *assembly* e na simulação. Ajuda.

Implementação

Processadores RISC são parecidos com o MIPS e não devem causar confusão.

Já os processadores CISC em geral usam *flags:* elas são *flip-flops* independentes que guardam uma condição. Veja como funciona uma comparação no 8051:

```
        CLR C         ; limpa flag C para não interferir na subtração
        SUBB A,32     ; subtrai 32 do reg. A, setando a flag Z se iguais
        JZ IGUAL      ; salta para IGUAL apenas se a flag Z está setada
DIF:    NOP           ; entra nesta linha só se A ≠ 32
```

Para maior e menor é usada a *flag* (C, de *carry)* e instrução JC; as negativas (JNZ e JNC) também estão presentes.

Atenção: saltos relativos têm que pular tanto para frente como para trás ("branch -5" volta cinco instruções). Se for feita extensão de sinal do operando, basta fazer normalmente a soma ao PC: ela vai funcionar, pelas propriedades de complemento de 2.

Nota: usualmente o endereço destino de um *branch* relativo é calculado a partir do endereço da instrução seguinte: um *beq $0,$0,30* no MIPS pula para PC+4+30. Verifique seu processador.

Detecção de Estouro

Como estamos trabalhando apenas com números não negativos, precisamos produzir apenas o *carry* (o vai-um da soma). O *overflow* (estouro sinalizado) pode ser ignorado se você desejar.

Para detectar o *carry* numa soma em VHDL, somos obrigados a usar um bit adicional na operação[3]. Veja um trecho da arquitetura:

```
   signal in_a_17,in_b_17,soma_17: unsigned(16 downto 0);
begin
   in_a_17 <= '0' & in_a;       -- passamos in_a para 17 bits
   in_b_17 <= '0' & in_b;       -- idem in_b
   soma_17 <= in_a_17+in_b_17;
   carry_soma <= soma_17(16);  -- o carry eh o MSB da soma 17 bits
```

Se não usarmos números negativos, para determinar o *carry* na subtração "in_a – in_b" basta fazer uma comparação direta:

```
   carry_subtr <= '0' when in_b<=in_a else
                  '1';
```

1 Usa o endereço destino como operando: "jump 34" vai pular para o endereço 34 na memória.
2 Usa delta de intruções a saltar: "branch 5" vai pular cinco instruções pra frente de onde ele está.
3 Lembrete: a concatenação é o operador &.

Para outras operações que você tenha implementado (multiplicação, incremento), os estouros podem ser ignorados, se você quiser.

Testes

Mantenha os pinos visíveis no *top level* como descritos no laboratório passado.

Para a entrega, o *testbench* e a ROM devem estar configurados para executar um programa que faz o seguinte:

1. Carrega R3 (o registrador 3) com o valor 0
2. Carrega R4 com 0
3. Soma R3 com R4 e guarda em R4
4. Soma 1 em R3
5. Se R3<30 salta para a instrução do passo 3 *****
6. Copia valor de R4 para R5

***** ***Requisito obrigatório:*** o salto para trás deve ser *relativo*

Sugiro fortemente escrever no papel (ou planilha) programas de teste, detalhando em cada linha a instrução em *assembly* e a codificação binária/hexadecimal e endereço na memória respectivos. Ajuda quando dá pau.

Se desejar, construa um montador simples que produz uma listagem hexadecimal a partir de um programa *assembly*.

Arquivos a Entregar

Anexe no email:

- ✔ Arquivos .vhd compactados (inclua o arquivo .gtkw de sinais, caso você tenha usado)
- ✔ Páginas do manual com as instruções de desvio escolhidas em destaque, como no laboratório passado
- ✔ Especificação atualizada da codificação das instruções
- ✔ Código *assembly* e codificação na ROM para o programa-teste pedido (pode estar apenas dentro do arquivo "rom.vhd" se você desejar, ou num arquivo à parte); repetindo: *eu quero as instruções em assembly do programa em ROM*