µProcessador 6 **"Calculadora Programável"** **rev 6**

Partindo da ROM, PC, UC, ULA e Banco de Registradores dos laboratórios anteriores, rodar um programa para executar uma lista de instruções aritméticas de um algoritmo hipotético (programa mostrado mais abaixo).

Levando em conta a ISA do microprocessador escolhido, implemente apenas instruções de soma (*add*) e subtração (*sub*) e resto da divisão (*rem*) da ULA, de carga de constantes (*addi*, *ld* ou similar), transferência de valores entre registradores (podendo ser um *mov* ou então manter com *add*, usando o R0 como constante). Também acomode o salto incondicional (*jmp*) e *nop* (já implementados no laboratório anterior).

Escolhas de codificação e largura do barramento devem ser negociadas com o professor. Outras instruções ficam para depois.

Ampliação da máquina de estados - Contadores em VHDL

Uma máquina de estados simples é apenas um contador. Em VHDL, ele é similar a um registrador:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity maq_estados is
   port( clk,rst: in std_logic;
         estado: out unsigned(1 downto 0)
   );
end entity;

architecture a_maq_estados of maq_estados is
   signal estado_s: unsigned(1 downto 0);
begin
   process(clk,rst)
   begin
      if rst='1' then
         estado_s <= "00";
      elsif rising_edge(clk) then
         if estado_s="10" then         -- se agora esta em 2
            estado_s <= "00";          -- o prox vai voltar ao zero
         else
            estado_s <= estado_s+1;    -- senao avanca
         end if;
      end if;
   end process;
   estado <= estado_s;
end architecture;
```

O código acima produz uma contagem de 0 a 2. Reproduza-o *ipsis literis*[1], não

1 Ou seja, absolutamente idêntico.

invente mudanças.

Perceba a comparação para o fim da contagem: primeiro vem “if estado_s="10" then” e só depois há o incremento. Não tente incrementar antes e comparar com “11” depois.[2] Deste jeito aqui facilita.

É possível, também fazer uma máquina com transições condicionais, como no multiciclo do livro, uma máquina de Moore, que é uma solução geralmente mais difícil de fazer funcionar. Você pode até usar, mas tenha certeza de *pensar bem, cuidadosamente*, na solução que está tentando implementar; e de modelar bem as transições para cada instrução .

Implementação

Sugiro fazer uma máquina de 3 estados: *fetch*, *decode* e *execute*, embora usar 2 já seria o suficiente neste lab[3]. O resto do trabalho é decodificar as instruções e gerar os sinais adequados para cada uma; e ligar todos os componentes seguindo aproximadamente o esquema do livro-texto.

A unidade de controle vai gerar sinais como o jump_enable e outros de seleção para mux em função do ciclo e do opcode da instrução, numa estrutura do tipo:

```
???_en <= '1' when opcode="???" and estado = ‘?’ else  '0';

???_sel <=  “00” when opcode="???" and estado = ‘?’ else
            “01” when opcode="???" and estado = ‘?’ else  “11”;
```

O formato de instruções e *opcodes* implementados devem ser documentados num arquivo .txt à parte (*EqNN-ISA.txt*). É permitido mudar os formatos de instrução (*opcodes*) em laboratórios posteriores.

Programa em ROM

Na versão final a ROM devem estar configurados para executar um programa que faz o seguinte.

1. Carrega R4 (o registrador 4) com o valor 3
2. Carrega R5 com 7
3. Soma R4 com R5 e guarda em R7
4. Subtrai 2 de R7
5. Salta para o endereço 17
6. No endereço 17, copia R7 para R4
7. No endereço 18, calcula o resto da divisão de R7 por 3 e guarda em R6
8. Salta para a terceira instrução desta lista (R7 ← R4+R5)

O programa deverá ser documentado no arquivo da ROM com as instruções no assembly desenvolvido (não apenas os *opcodes*).
Se necessário, consulte e simule no MARS para verificar o comportamento esperado do programa. Se não houver as mesmas instruções, pode-se adaptar o algoritmo, mas o resultado em R6 deve ser igual.

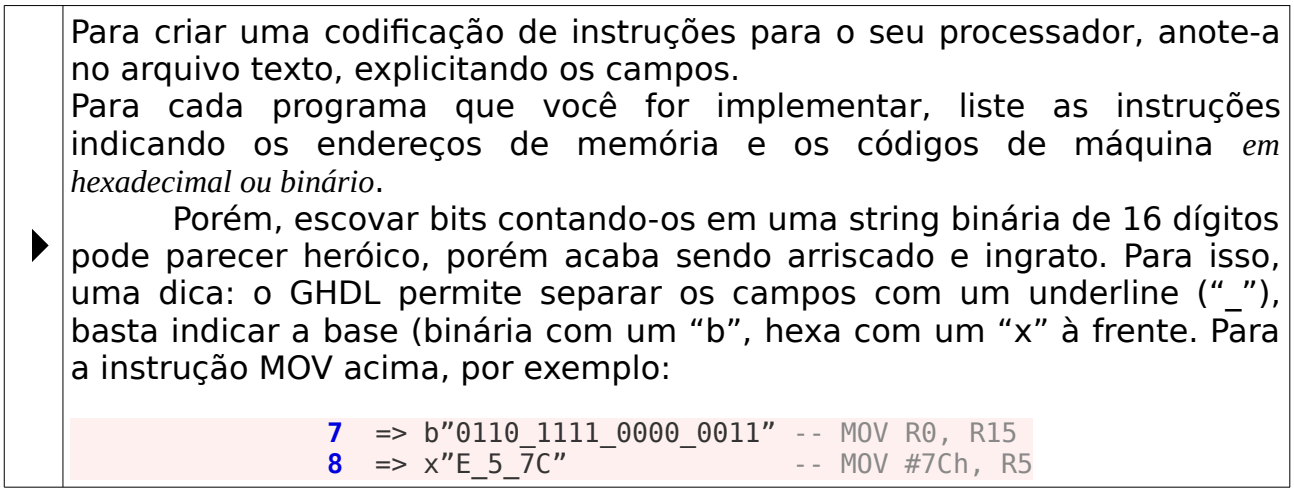
2 O “signal” só vai ser atualizado ao final do ciclo de simulação (o “end process;”), então deve-se comparar com o valor original, ainda não atualizado. Para atualizações imediatas, deve-se usar “variable”, que possui sintaxe levemente diferente. Evite estas complicações se possível.

3 Você pode alterar isso depois, mas se quiser usar já mais estados (a RAM pode ficar mais fácil com 4 estados, p. ex.), fique à vontade.

Testes

Os testes podem ser feitos com programas mais simples, inicialmente. Na versão final entregue com os resultados da simulação *EqNN-Calc.ghw* e *EqNN-Calc.gtkw*, os pinos sinais visualizados no *gtkwave*, devem ser:

- reset, clock, estado, PC, instrução (saída da ROM);
- as entradas da ULA (valores de in_A e in_B);
- saída da ULA e o valor dos registradores envolvidos (R4,R5, R6 e R7).



Para criar uma codificação de instruções para o seu processador, anote-a no arquivo texto, explicitando os campos.

Para cada programa que você for implementar, liste as instruções indicando os endereços de memória e os códigos de máquina *em hexadecimal ou binário*.

Porém, escovar bits contando-os em uma string binária de 16 dígitos pode parecer heróico, porém acaba sendo arriscado e ingrato. Para isso, uma dica: o GHDL permite separar os campos com um underline ("_"), basta indicar a base (binária com um "b", hexa com um "x" à frente. Para a instrução MOV acima, por exemplo:

```
7  => b"0110_1111_0000_0011" -- MOV R0, R15
8  => x"E_5_7C"              -- MOV #7Ch, R5
```

Reimpressão do FAQ do lab #3: Erros Comuns ("*O que diachos está acontecendo?*")

Abaixo, coisas que o compilador não pega direito:

- Erros bizarros podem ser gerados se você errar o nome da entidade na arquitetura:
  ```
  architecture a_porta_tb of porta is   -- Era porta_tb! Erro de copy-paste
  ```
  O compilador pode indicar que os pinos do componente estão errados ou que não acha o componente.
- Esquecer um pino no *port map* significa "deixar ele em aberto" e então *o programa não avisa isso...* Se alterar a pinagem duma entidade, confira imediatamente os "component" dela em outros arquivos.
  ```
  uut: porta port map( in_a  => in_a,
                       -- in_b  => in_b, <== Isso não é avisado!!!
                       a_e_b => a_e_b);
  ```
- Você usou if? *Você usou* ***if?!*** Peralá, só use isso dentro do registrador, campeão. Se a internet sugeriu que você usasse em outro lugar, *ignore-a* e **NÃO USE IF.**
- Tem um sinal com duas atribuições? Tipo, mesmo em arquivos diferentes ou lugares diferentes? Como abaixo?
  ```
  d <= b or c;
  -- ...
  d <= a;
  ```
  Erro mortal. Você está tentando dar duas definições diferentes para o mesmo ponto do circuito (dois valores para o mesmo fio ou ponto de medição). Se fosse na bancada, provavelmente veríamos fumaça nessa situação.