µProcessador 5 "Calculadora Programável"

Partindo da ROM, PC, UC, ULA e Banco de Registradores dos laboratórios anteriores, executar um programa para executar uma lista de instruções aritméticas.

Implemente apenas instruções da ULA, de carga de constantes, transferência de valores entre registradores, salto incondicional e *nop;* outras instruções ficam para depois.

Também acrescente um Registrador de Instrução no circuito, que apenas armazena a instrução lida da ROM, que será executada nos clocks seguintes. Note que juntar tudo isso dá mais trabalho do que parece.

Sugestão: pra agilizar o *debug*, que pode começar a ficar tedioso, faça um *script* (no *bash* ou um .bat) ou *makefile* para compilar e simular, e use um arquivo para a lista de sinais do *gtkwave* como mencionado no laboratório #3.

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 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 pra comparar com "11" depois.[2] Deste jeito aqui facilita.

Se algum item sorteado for sensível a rampa de descida, utilize falling_edge(clk) ao invés

1 Ou seja, absolutamente idêntico.

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.

de rising_edge(clk).

Se quiser fazer uma máquina com transições condicionais, como no multiciclo do livro, não recomendo; é uma solução geralmente mais difícil de fazer funcionar. Você pode até usar uma Máquina de Moore, mas tenha certeza de *pensar bem, cuidadosamente,* na solução que está tentando implementar; daí rola.

Implementação

Sugiro fazer uma máquina de 3 estados: *fetch, decode* e *execute*, embora usar 2 já seria o suficiente neste lab[3]. O sorteio do seu processador pode alterar isso, verifique os itens.

O resto do trabalho é decodificar as instruções e gerar os sinais adequados para cada uma, ligando todos os componentes seguindo aproximadamente o esquema visto em aula.

O tamanho das instruções é sorteado para a equipe, e é igual à largura de um dado da ROM.

As instruções a serem codificadas são aquelas especificadas por email para a equipe. É obrigatório estas instruções *exatas* (não "quase iguais") estarem presentes no *assembly* do processador escolhido pela equipe.

Tanto o formato interno dos bits das instruções quanto os *opcodes* são livres e deverão ser decididos, implementados e documentados num arquivo à parte. É permitido mudar os formatos de instrução (os *opcodes*) em laboratórios posteriores.

Faça aos poucos, uma implementação incremental, senão fica muito difícil depurar.

Testes

Na versão final entregue, os pinos visíveis no *gtkwave* devem ser, preferencialmente nesta ordem:

- reset;
- clock;
- estado;
- PC;
- instrução (saída do Registrador de Instrução, ou, se não houver, da ROM);
- saídas do acumulador e valores internos de todos os registradores, em ordem;
- saída da ULA.

Pode usar outros pinos durante a implementação e a fase de *debug*, mas retire-os para a apresentar para o professor.

Também espero que você teste vários programas durante o desenvolvimento mas, para a entrega, o *testbench* e a ROM devem estar configurados para executar um programa que faz as seguintes ações a partir do endereço zero:

A. Carrega R3 (o registrador 3) com o valor 5
B. Carrega R4 com 8
C. Soma R3 com R4 e guarda em R5
D. Subtrai 1 de R5
E. Salta para o endereço 20
F. Zera R5 (nunca será executada)
G. No endereço 20, copia R5 para R3
H. Salta para o passo C desta lista (R5 <= R3+R4)
I. Zera R3 (nunca será executada)

O programa em linguagem assembly deverá ser documentado no projeto (a lista de instruções como "ADD A,R1", não o binário), obedecendo obrigatoriamente às restrições sorteadas para a equipe. Em especial, fazer o passo C (R5 <= R3+R4) pode exigir várias instruções e movimentação com o

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

acumulador.

A sequência de valores de R5 observada na execução é 12, 19, 26, 33, 40, 47,... (ou 0x0C, 0x13, 0x1A, 0x21, 0x28,... em hexadecimal).

Arquivos a Entregar

Anexe na entrega, compactados num arquivo só ou não:

- Todos os fontes .vhd e os testbenches _tb.vhd respectivos (o testbench principal (*top-level*) deverá ser chamado "processador_tb.vhd");
- Especificação da codificação das instruções (planilha ou texto, em .txt ou xls ou .pdf ou similar);
- 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 .txt à parte); repetindo: *eu quero as instruções em assembly, não apenas os opcodes.*

Exemplo mínimo de um processador sorteado (trechos; está meio tosco, poderia ser melhor, mas é suficiente):

"codificação.txt"

```
MSB b15              b0 LSB
      |              |
ADD:  0100 ddd sss xxxx
SUBI: 0101 ddd sss cccc
...
onde
      ddd  identifica o registrador destino
      sss  identifica o registrador fonte
      cccc identifica a constante de 4 bits em complemento de 2
      xxxx é irrelevante
```

"assembly.txt"

```
      LI  R3,5
      LI  R4,8
C:    MOV A,R3 ; o passo C precisa de 3 instruções neste meu µProcessador
      ADD A,R4
      MOV R5,A
D:    SUBI A,1
      ...
```

Dica esperta: se você explicitar uma string como binária (colocando B na frente) dá pra usar sublinhado como separador pra visualizar melhor, sem alterar o valor:

```
result_s <= B"0010_101_110";       -- ou seja, vale "0010101110"
```

Lembrando: para concatenar dois std_logic_vector (ou unsigned) basta usar &:

```
final_s <= canal_i & "00" & sel_s; -- sendo std_logic_vector ou unsigned
```

Última sugestão: releia rapidamente o FAQ do Moodle pra ver os erros comuns, alguns deles costumam surgir neste momento. Em especial tem esse aqui:

> Você usou if? *Você usou* ***if?!*** Peralá, só use isso para construir um registrador simples, sem nenhuma lógica. Se a internet ou o ChatGPT sugeriu que você usasse em outro lugar, *ignore-os* e **não use if**. Provavelmente *when-else* é o que você quer.

Avaliação

Neste lab a avaliação não é binária como os anteriores. Critérios:

- Se não funcionar o programa, mas o VHDL parece ok: -20%
- Usou `process` e `if` ou `case/when` sem dominar o uso correto: -30%
- Se executar instrução seguinte ao desvio: -10%
- Se o opcode 0x00 não for NOP: -10%
- Se a instrução no endereço 0x0 for NOP: -10%
- Se não executar a instrução no endereço 0x0: -10%
- Se não tem a listagem assembly (em txt ou como comentários na rom.vhdl): -20%
- Se uma instrução ou assembly não corresponde ao sorteado: -10%

Caso o sorteio da sua equipe contradiga algo da avaliação, fique livre para escolher com fazer este item particular.

O professor irá verificar o funcionamento pessoalmente em sala na aula seguinte à entrega deste lab.

Verificação da Lógica Sequencial

Por favor desenhe manualmente em uma folha ou no caderno as formas de onda dos seguintes sinais (com o nome que você escolheu), para a execução completa de uma instrução: clock, wr_en do PC, valor do PC, wr_en do acumulador (se houver), valor do acumulador, wr_en dos registradores e valor do registrador escrito.

Em especial, identifique o estado de cada ciclo de clock e a transição dos valores, como por exemplo: