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 quiser fazer uma máquina com transições condicionais, como no multiciclo do livro, há

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.

um pdf no Moodle que descreve como fazer 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; daí rola.

Implementação

Sugiro fazer uma máquina de 3 estados: *fetch, decode* e *execute*, embora usar 2 já seria o suficiente neste lab[3]. No *fetch* a ROM é lida e este valor é escrito no Registrador de Instrução.

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 especificação do processador a implementar e a largura de bits da ROM estão no Moodle.

As instruções a serem codificadas são aquelas escolhidas pela equipe na semana passada. É obrigatório estas instruções *exatas* (não "quase iguais") estarem presentes no *assembly* do processador escolhido pela equipe.

O formato de instruções e *opcodes* é livre e deverá ser decidido, implementado e documentado num arquivo à parte. É permitido mudar os formatos de instrução (*opcodes*) em laboratórios posteriores.

Testes

Na versão final entregue, os pinos visíveis no *top level,* visualizados no *gtkwave*, devem ser:

- reset;
- clock;
- estado;
- PC;
- instrução (saída do Registrador de Instrução);
- saídas do banco de registradores (valores de Reg1 e Reg2);
- saída da ULA.

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

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 o seguinte:

1. Carrega R3 (o registrador 3) com o valor 5
2. Carrega R4 com 8
3. Soma R3 com R4 e guarda em R5
4. Subtrai 1 de R5
5. Salta para o endereço 20
6. No endereço 20, copia R5 para R3
7. Salta para a terceira instrução desta lista (R5 <= R3+R4)

O programa deverá ser documentado no projeto.

Arquivos a Entregar

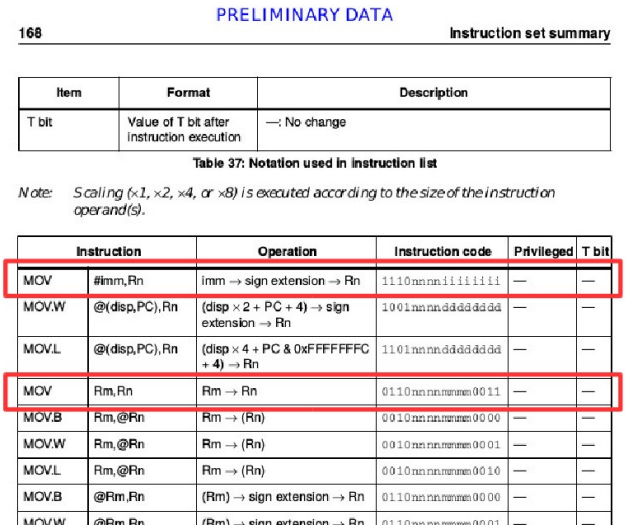
Anexe na entrega:

- Todos os fontes .vhd e os testbenches _tb.vhd respectivos, compactados num arquivo só ou não (O testbench principal (*top-level*) deverá ser chamado "processador_tb.vhd")
- *Datasheet* (manual) do processador escolhido em pdf ou *link* para ele
- Páginas do manual com as instruções escolhidas em destaque, em pdf (veja mais adiante)

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

- Especificação da codificação das instruções em .txt ou .pdf (planilha ou texto)
- 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, não apenas os opcodes*

As páginas que descrevem as instruções implementadas, com destaque se necessário, devem ser entregues à parte. Por exemplo:



PRELIMINARY DATA

168 Instruction set summary

Item	Format	Description
T bit	Value of T bit after instruction execution	—: No change

Table 37: Notation used in instruction list

Note: Scaling (×1, ×2, ×4, or ×8) is executed according to the size of the instruction operand(s).

Instruction		Operation	Instruction code	Privileged	T bit
MOV	#imm,Rn	imm → sign extension → Rn	1110nnnniiiiiiii	—	—
MOV.W	@(disp,PC),Rn	(disp × 2 + PC + 4) → sign extension → Rn	1001nnnndddddddd	—	—
MOV.L	@(disp,PC),Rn	(disp × 4 + PC & 0xFFFFFFFC + 4) → Rn	1101nnnndddddddd	—	—
MOV	Rm,Rn	Rm → Rn	0110nnnnmmmm0011	—	—
MOV.B	Rm,@Rn	Rm → (Rn)	0010nnnnmmmm0000	—	—
MOV.W	Rm,@Rn	Rm → (Rn)	0010nnnnmmmm0001	—	—
MOV.L	Rm,@Rn	Rm → (Rn)	0010nnnnmmmm0010	—	—
MOV.B	@Rm,Rn	(Rm) → sign extension → Rn	0110nnnnmmmm0000	—	—
MOV.W	@Rm,Rn	(Rm) → sign extension → Rn	0110nnnnmmmm0001	—	—

Isso pode ser entregue em pdf, ou um printscreen em jpg, tanto faz.

Também é necessário entregar um documento com a codificação utilizada, ou seja, o formato das instruções e *opcodes* usados. Inclua explicitamente qualquer adaptação que se decidiu fazer (ex.: não usar o $0 como constante ou então mapear o acumulador para o $1, se for o caso).

▶	Crie uma codificação de instruções para o seu processador. Anote-a num arquivo texto, pdf ou planilha, 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*. Decimal é para os fracos.

Dica: 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 dentro do registrador, campeão. Se a internet sugeriu que você usasse em outro lugar, *ignore-a* e **não use if.**"