Insper

Elementos de Sistema - Projeto E - Lógica Sequêncial

Rafael Corsi - rafael.corsi@insper.edu.br

Março - 2018

```
Prazo original : Quarta Feira - 28/3/2018
Atrasado até   : Quarta Feira - 04/4/2018
```

Descrição

Neste projeto você terá que desenvolver os componentes de memória que serão utilizados no computador Z01.

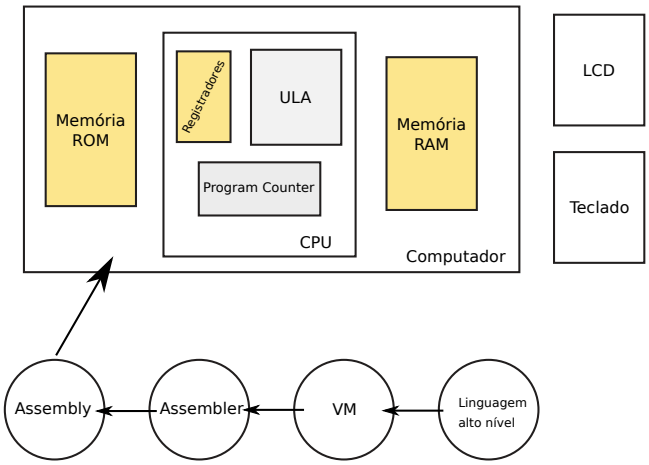

Figure 1: ULA

Integrantes

Este projeto deve ser realizado por todos os integrantes do grupo. As tarefas devem ser criadas no Kanban e os alunos poderão executá-las em duplas, prati-

cando o pair-programming. Porém, evite sempre trabalhar na mesma dupla, ou seja, cada projeto deverá ser realizado por uma nova dupla. Se organizem para tentar desenvolver o projeto de forma que todos aprendam o máximo possível.

Controle de Tarefas e Repositório

Atualize o seu Fork do projeto com o repositório do github:

- https://github.com/Insper/Z01

(Nas) discussões com o grupo, o scrum master deverá definir os módulos que cada integrante irá desenvolver. Crie uma rotina para commits e pull-requests. Sempre teste os módulos e verifique se está funcionando como o esperado.

Funções do Facilitador (Scrum Master)

- Organizar o github + issues + project;
- Fazer a atualização do fork com o upstream;
- Gerenciar o grupo (atribuir tarefas);
- Verificar e aceitar os pull-requests;
- Fazer o relatório de desempenho de cada integrante do grupo;
- Entregar/Apresentar o projeto no final.

Funções dos Desenvolvedores

- Realizar as tarefas atribuídas pelo scrum-master;
- Realizar os pull-requests;
- Implementar e restar os códigos.
- Gerar as formas de onda
- Gerar o RTL

Instruções

Entendendo a Organização do Projeto

A pasta do projeto D, no repositório Z01, possui a seguinte estrutura:

```
/E-LogicaSequencial
    /Quartus
    /script
        testeLogicaSequencial.py
    /src
        /rtl
```

```
        *.vhd
/tests
    /tst
        *.vhd
```

1. Quartus: Projeto Quartus que faz uso dos arquivos VHDL localizados em src/rtl/*.vhd;
2. scripts: Scripts em python que automatizam a execução dos testes;
3. src/rtl/*.vhd: Arquivos VHDL que serão implementados pelo grupo;
4. tests/tst/*.vhd: Arquivos VHDL que realizam o teste lógico nos arquivos do rtl.

Executando o Script de Teste

Abra o terminal na pasta *E-LogicaSequencial/script* e execute o script python localizado nessa pasta:

```
$ python testeLogicaSequencial.py
```

O mesmo irá compilar os arquivos src/rtl/*.vhd e executar os testes unitários em cada um deles. Nesse momento do teste, como os módulos não estão implementados, o resultado deverá ser falho.

Esse comando executa um teste unitário em cada um dos módulos, verificando se sua implementação está correta. O resultado é exibido na tela como : **pass** ou **fail**.

> **!** Lembrando que o arquivo tests/config.txt define quais testes serão executados.

Projeto

- Deve-se gerar uma imagem com a forma de onda desses módulos.
- Deve-se gerar um RTL para cada módulo do preto (E-Ferramental.pdf)
- Note que é possível reaproveitar, via port map, os módulos dos projetos anteriores, basta declarar o componente e usar o módulo.

Módulos

> **!** Esses arquivos estão localizados em E-LogicaSequencial/src/rtl/

O FlipFlopD.vhd é um bloco elementar e sua implementação é criada no ferramental do projeto (E-Ferramental).

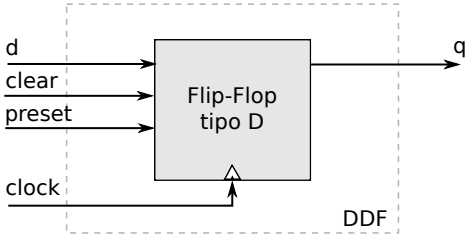

Figure 2: FlipFlop D

Deve-se implementar os seguintes circuitos sequenciais :

- Binary Digit
  - **Arquivo** : BinaryDigit.vhd
  - **Dependência** : FlipFlopD e Mux2Way
  - **Descrição** : É um registrador feito para armazenar um único bit de informação (0 ou 1). A interface do módulo consiste em uma entrada (d) para o bit a ser armazenado, um sinal de *load* para indicar quando o bit de entrada deve ser armazenado um sinal de *clock* e a saída *output* que é o bit armazenado :

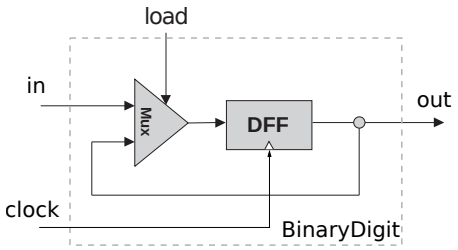

Figure 3: Binary Digit

- Register 8
  - **Arquivo** : Register8.vhd
  - **Dependência** : BinaryDigit
  - **Descrição** : É um registrador de 8 bits criado a partir do binaryDigit porém agora para armazenar um vetor de entrada de 8 bits de tamanho.

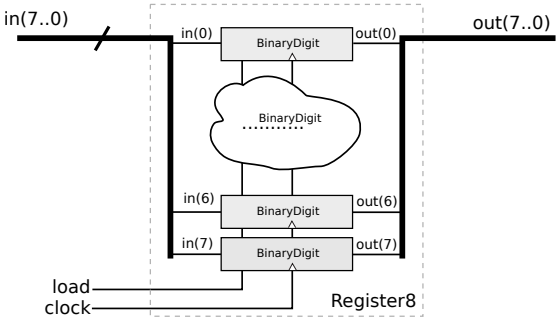

Figure 4: Register 8

- Register 16
  - **Arquivo** : Register16.vhd
  - **Dependência** : Register8
  - **Descrição** : É um registrador de 16 bits criado a partir do Register8 porém agora para armazenar um vetor de entrada de 16 bits de tamanho.

- Register 32
  - **Arquivo** : Register32.vhd
  - **Dependência** : Register16
  - **Descrição** : É um registrador de 32 bits criado a partir do Register16 porém agora para armazenar um vetor de entrada de 32 bits de tamanho.

- Register 64
  - **Arquivo** : Register64.vhd
  - **Dependência** : Register32
  - **Descrição** : É um registrador de 64 bits criado a partir do Register32 porém agora para armazenar um vetor de entrada de 64 bits de tamanho.

- Program Counter
  - **Arquivo** : PC.vhd
  - **Dependência** : -
  - **Descrição** : O program counter será o nosso endereçador de memória da CPU, ele será responsável por apontar para a próxima instrução a ser executada. Como normalmente um código segue um fluxo sequencial (uma linha na sequência da outra) o PC possui a habilidade de se auto incrementar a cada clock (apontando assim para a próxima instrução), mas ele tem que suportar *condição* (if, while, ...) rompendo com esse fluxo contínuo.

Sula lógica é descrita no pseudo código a seguir :

```
If reset(t-1) then
   out(t)=0
else if load(t-1) then
   out(t)=in(t-1)
else if inc(t-1) then
   out(t)=out(t-1)+1
else
   out(t)=out(t-1)
```

- Ram8
  - **Arquivo** : Ram8.vhd
  - **Dependência** : Register16, Mux8Way16, Dmux8Way
  - **Descrição** : É uma memória de 8 endereços com 16 bits de largura. O componente possui como entrada o vetor input de 16 bits, o endereço a ser armazenado (address) o sinal *load* que indica quando é para ser armazenado e o clock. Como saída temos o valor lido no endereço especificado quando load for igual a 0. Note que sinal LOAD tem como função similar o do READ/WRITE, quando zero, indica que queremos ler o valor armazenado, quando 1 indica que queremos escrever (write) nessa posição.

- Ram64
  - **Arquivo** : Ram64.vhd
  - **Dependência** : Ram8, Mux8Way16, Dmux8Way
  - **Descrição** : Similar a RAM8 porém com 64 endereços.

- Ram512
  - **Arquivo** : Ram512.vhd
  - **Dependência** : Ram64, Mux8Way16, Dmux8Way

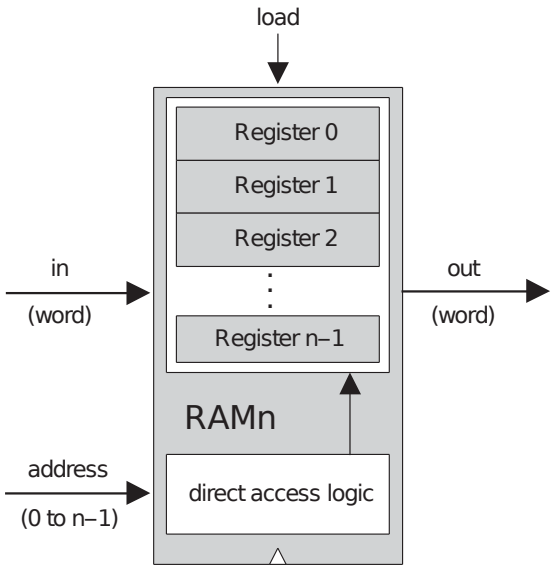

Figure 5: RAM 8

– **Descrição** : Similar a RAM8 porém com 512 endereços.

- Ram4k
  - **Arquivo** : Ram4k.vhd
  - **Dependência** : Ram512, Mux8Way16, Dmux8Way
  - **Descrição** : Similar a RAM8 porém com 4512 endereços.

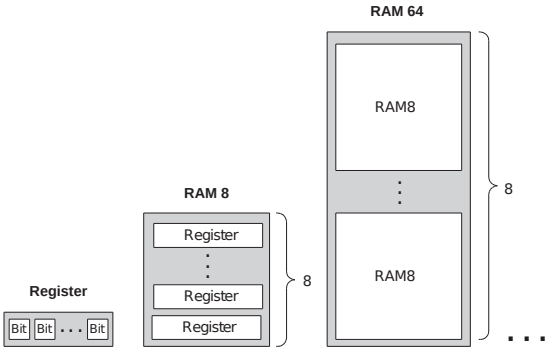

Figure 6: RAM

> **!** Todos esses módulos estão super bem documentados no livro The Elements of Computer System. Cap 3.

Forma de onda

Para cada teste realizado, deve-se carregar a interface gráfica e tirar um print da forma de onda do módulo com os testes aplicados a ele (D-Ferramental.pdf). Essa imagem deve ser salva na mesma pasta dos arquivos vhdl (src/rtl/) e com o mesmo nome dos módulos.

> Não basta só gerar a imagem, você precisa analisar e entender.

Testando em Hardware

No hardware você deverá desenvolver um cenário de teste para o Program-Counter ou para para a RAM8.

Rubricas para avaliação de projetos

Cada integrante do grupo irá receber duas notas: uma referente ao desenvolvimento total do projeto (Projeto) e outra referente a sua participação individual no grupo (que depende do seu papel).

> Notem que houve alteração nas rubricas, agora iremos ter apenas dois conceitos para os papeis : Insatisfatório (I) ou satisfatório (A). Isso acontece dado a complexidade de classificarmos todos os alunos em 5 conceitos distintos.

Projeto

Conceito	
I	- Menos da metade dos módulos funcionando
D	- Ao menos um módulo não está feito e não passa no testes.
C	- Configurou o travis para operar com o novo projeto. - Todos os módulos sendo testados no Travis. - **O ramo master passaa nos testes do travis** - Todos os módulos passam nos testes. - Possui a forma de onda de todos os módulos (.png).
B	- Usou sempre que possível outros módulos para criar um novo (hierarquia) - Exemplo : usou o inc16, mux16 e reg8 para criar PC
A	- Gravou e testou a memória RAM8 na FPGA. (gravou um vídeo para mostrar o funcionamento) ou o ProgamCounter

Desenvolvedor

Conceito	
Insatisfatório	
(I)	- Se comprometeu a fazer algum desenvolvimento (kanban) e não fez. - Criou o branch mas não fez o desenvolvimento completo.
Satisfatório	
(A)	- Desenvolveu as rotinas atribuídas pelo Facilitador para passarem nos testes! - Acompanhou o Kanban board (github project). Ex: Puxou tarefas, etc. - Submeteu alterações por pull requests.

Facilitador

Conceito	
Insatisfatório	
(I)	- Não acompanhou o projeto, deixando os colegas sozinhos. - O relatório das atividades não é condizente com o real desempenho dos integrantes (analisado via git).
Satisfatório	
(A)	- Atualizou o repositório pelo Fork. - Fez a manutenção do Kanban board (GitHub project). Ex: cria cards, atribui tarefas, da feedback de inssues. - Aceitou os pull-requests. - Resolveu conflitos de merge nos pull requests. - Acompanhou o desenvolvimento do grupo, dando o suporte sempre que necessário - **Entregou o branch master sem nenhum erro (passando no travis)** - Fez o relatório das atividades descrevendo o papel de cada integrante com clareza.