Insper

Elementos de Sistemas - Projeto F - Ferramental - Assembly

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

Março, 2018

TODO de inicialização: Scrum Master:

1. Atualizar arquivo ScramMaster.txt
2. Atualizar repositório com o upstram
3. Adicionar o novo script de teste ao travis : F-Assembly/scripts/testeAssembly.py
4. Criar projeto no github
5. Atribuir tarefas e acompanhar o desenvolvimento

Esse projeto tem como objetivo introduzir a linguagem assembly e a arquitetura do computador Z01.

Simulador

Nesse projeto iremos criar programas em assembly que serão executados no computador que estamos construindo (usando as memórias e ULA recém criadas), nesse projeto estamos pulando uma etapa que seria a de conectar a ULA com as memórias, os registradores e unidade de controle. Temos o objetivo de deixar mais claro qual o real papel dos módulos na hora de montarmos a CPU.

O nosso código assembly pode ser executado em hardware (FPGA) porém nesse primeiro momento iremos trabalhar em um ambiente simulado que nos dará maior facilidade de programação e depuração. O livro texto (The Elements Of Computer System) disponibiliza um simulador da CPU porém possui código fechado e pouco espaço para customização, em 2017 o Prof. Luciano Pereiro iniciou a criação de um simulador Z0 (versão anterior) em Java, porém percebemos alguns pontos negativos de utilizar um simulador em Java sendo o principal que : qualquer alteração no Hardware iria demandar uma alteração no simulador, sendo necessário mantermos dois projetos independentes e sincronizados.

Nesse curso, iremos utilizar um simulador que utiliza o nosso próprio código VHDL como descrição da CPU (e de tudo envolvido), uma alteração no código irá automaticamente alterar o simulador e o comportamento do computador. Para isso, fazemos uso do ModelSim, um software da Mentor Graphics que

executa simulaçòes em VHDL (o mesmo utilizado nos projetos anterios), desenvolvemos uma série de APIs e configurações desse simulador para funcionar para a disciplina.

> As APIS de interface do simulador foi desenvolvido por mim (Rafael) e a interface gráfica pelo Prof. Eduardo Marossi.

O simulador possui a estrutura ilustrada a seguir :

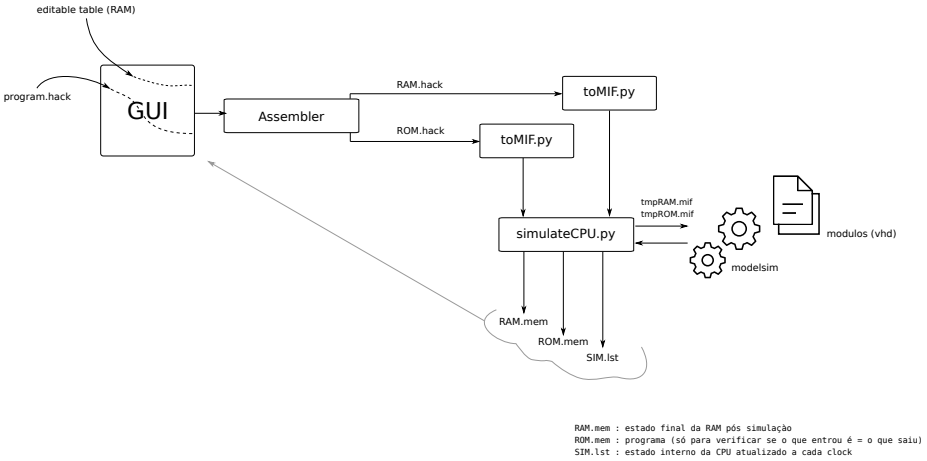

Figure 1: Simulador

O modelsim possui de entrada a memória RAM e a memória ROM e executa por um determinado tempo (definido pelo usuário), após a simulação é exportado diversos sinais internos da CPU, esses sinais são então lidos pela interface gráfica e exibida de uma forma amigável.

Dependências

A interface gráfica do simulador foi feita em PyQt5 e será necessário instalarem via pip para poderem utilizar o simulador :

```
pip install –user PyQt5
```

Arquivos

O simulador está localizado nas pastas Z01/tools/ :

- Z01-Simulator-GUI : Parte gráfica do simulador

- Z01-Simulator-RTL : Projeto ModelSim para executar o código

> Para inicializar o simulador basta executar o script localizado na pasta
> `python F-Assembly/scripts/Z01simulador.py`

Interface do Simulador

O simulador possui a interface a seguir, onde a coluna da esquerda é referente a memória ROM (programa), a coluna da direita referente a memória RAM (dados).

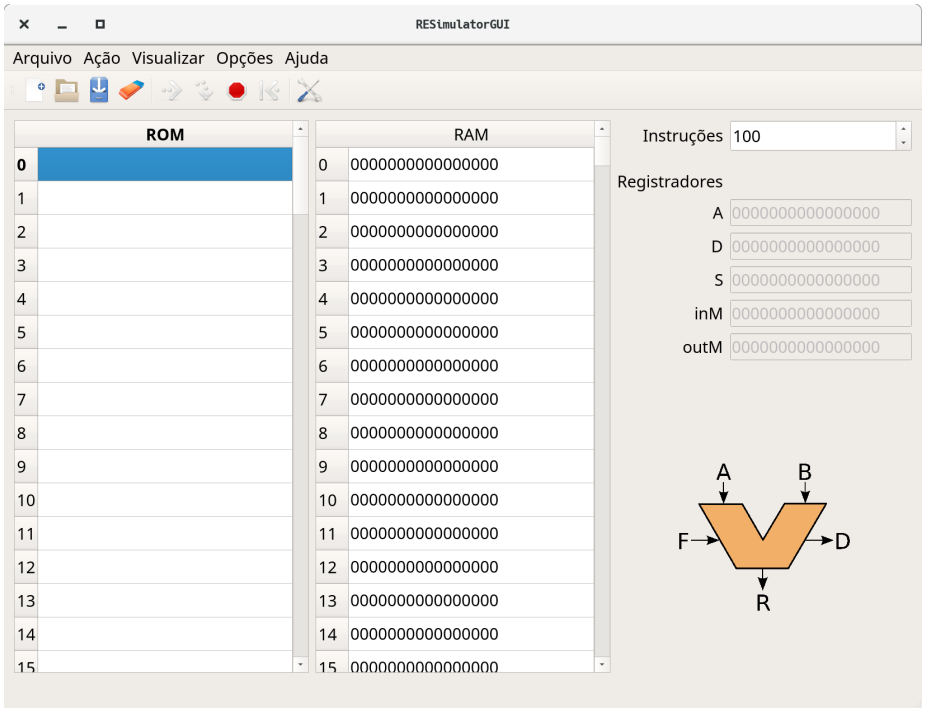

Figure 2: Simulador GUI

Toda vez que houver uma alteração em algum dos parâmetros do simulador (RAM/ROM/Instruções,...) o código será novamente simulado para obtermos um resultado atualizado com as entradas. Isso pode dar a sensação de "lerdeza" mas lembre da complexidade do sistema : estamos executando um código em um hardware simulado.

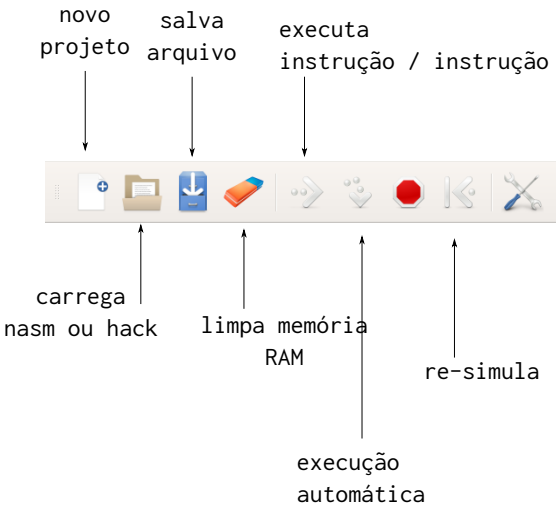

Figure 3: tool

Programando

Abra o simulador e insira o seguinte código em nasm :

```
leaw $R1,%A
movw (%A),%D
leaw $R0,%A
addw (%A), %D, %S
leaw $R2, %A
movw %S, (%A)
```

Esse código soma o valor que está salvo na memória RAM endereço 0 com o valor da memória RAM endereço 1 e salva no endereço 0 :

```
RAM[2] = RAM[0] + RAM[1]
```

> **!** Não continue caso não entendeu o código !

Agora será necessário colocarmos valores iniciais na memória RAM para validarmos o nosso código, para isso altere a memória RAM endereços 0 e 1 com o valor 5 e 8 respectivamente, como demonstrado a seguir : .

	RAM
0	0000000000000101
1	0000000000001000
2	0000000000000000

! Com a memória alterada você pode agora executar a simulação, verifique se o valor da memória 2 é a soma dos endereços 0 e 1.

Treinando

Altere o código para armazenar o resultado no endereço RAM[5]

Altere o código para não usar o registrador %S

Faça a seguinte operação : RAM[0] = RAM[3] - RAM[2]

Script automático de testes

Além da interface gráfica do simulador, possuímos um script de teste automatizado (similar ao do VHDL), esse script: *F-Assembly/scripts/testeAssembly.py* compila os códigos que estão na pasta *F-Assembly/src/nasm* para a pasta *F-Assembly/bin/hack* e executa os testes localizados em *F-Assembly/tst/*. Somente os arquivos configurados no **config.txt** serão testados.

config.txt

O arquivo de configuração dos testes é um pouco diferente, possui além do nome do módulo que será testado um segundo parâmetro que indica quantos testes serão executados para esse módulo e quantos microsegundos ele ficará na simulação (microsegundos suposto de um sistema real).

Exemplo do config.txt

```
# nome | quantidade de testes | us de execucao
#add 1 1000
```

Remova o comentário do módulo add

Implementando o add.nasm

Os arquivos a serem implementando estão na pasta F-Assembly/src/nasm/ lá você vai encontrar todos os códigos fontes que deverão ser feitos nesse projeto.

Edite o arquivo add.nasm com a implementação do add anterior

Agora com o módulo implementando podemos testar-lo, para isso execute o script testeAssembly.py . Esse script irá compilar o nasm e gerar os arquivos .hack e .mif (salvos no /bin/hack/) que serão carregados no simulador junto com uma configuração inicial da memória RAM (como no gui do simulador), ao término da simulação um arquivo com o estado final da RAM é salvo na pasta /tests/tst/add/add0_end.mif.

Executamos um script que compara o estado final da RAM com o um esperado (add0_tst.mif), em caso de algum erro, o scripr irá reportar falha.

Se tudo ocorrer bem você deverá ter a seguinte saída :

```
- Testando ....
-------------------------
Starting  sub0 ....
pass    sub0
Starting  add0 ....
pass    add0
==== Summary ===================================
pass    sub teste: 0
pass    add teste: 0
```

Formato de arquivos

A seguir uma lista dos principais formatos de arquivos utilizados :

- .nasm : Arquivo assembly
- .hack : Linguagem de máquina (arquivo com zeros e uns)
- .mif : Arquivo .hack que pode ser salvo na memória da FPGA
- .lst : saída do simulador com os estados da CPU