Atividade de Laboratório 5

Objetivos

O objetivo deste laboratório é familiarizar a turma com a infraestrutura para montagem e execução de código RISC-V, que será utilizada no restante do curso. Nesta atividade, é esperado que o aluno compreenda como montar, ligar e executar, no simulador, um programa escrito em linguagem de montagem do RISC-V, além de depurar o código em linguagem de montagem.

Descrição

Neste laboratório, você deve fazer um programa em linguagem de montagem do RISC-V que imprima seu nome seguido do seu RA na tela, na forma "Primeiro_nome - raXXXXXX". O código do programa em linguagem de montagem está disponível em modelo.s.

Note que o arquivo modelo.s está bastante comentado. Nessa atividade, **você deve alterar a string "MC404\n" para "Seu_nome - raXXXXXX\n" e o tamanho da string**, que é copiado para o registrador a2.

Executar um programa escrito em linguagem de montagem do RISC-V exige o uso de um simulador RISC-V, pois os computadores do laboratório possuem processadores com conjunto de instruções da família de arquiteturas x86, sendo assim incompatíveis com código RISC-V.

Desse modo, a não ser que se utilize um *hardware* RISC-V, é preciso executar uma sequência de passos para executar seu programa num computador da família x86.

Tal sequência, juntamente de uma breve descrição do funcionamento do simulador, está disposta na seção seguinte. Recomenda-se atenção aos passos aqui descritos, pois as etapas são necessárias para todos os futuros laboratórios.

Simulador RISC-V

O simulador da arquitetura RISC-V usado nessa disciplina foi criado pela empresa Western Digital para simular o funcionamento do processador SweRV Core, que é utilizado no micro-controlador de mesmo nome (https://blog.westerndigital.com/risc-v-swerv-core-open-source/). Sua implementação original foi escrita em C++, foi planejada para ser executada sobre a arquitetura Intel x86_64 e está disponível no GitHub sob a licença GPLv3. Para facilitar o uso da ferramenta, nós modificamos e compilamos o simulador para JavaScript (mais especificamente, para WebAssembly e asm.js, utilizando Emscripten), de forma que ele possa ser executado diretamente no navegador. (**NOTA: Sugerimos FORTEMENTE o uso do navegador Chrome nos laboratórios**)

O micro-controlador SweRV foi planejado para executar apenas em modo máquina, sem a presença de um sistema operacional. No entanto, o simulador disponibiliza os recursos necessários para a utilização de um sistema operacional simplificado. Na primeira etapa do curso, utilizaremos um sistema operacional simulado para oferecer suporte às chamadas de sistema realizadas pela biblioteca Newlib. Tal sistema implementa as *syscalls* mais comuns do Linux, permitindo que um programa, ao executar no simulador, escreva e leia dados de dispositivos.

Para montar um programa escrito em linguagem de montagem do RISC-V, originalmente procedemos da mesma forma que já conhecemos: usa-se um montador e em seguida um ligador (*linker*) para gerar código executável. Contudo, como estamos usando computadores da família x86, vamos utilizar um ambiente de compilação cruzada (*cross compiling*), de modo que usaremos um montador e um *linker* que funcionam nas famílias x86, mas que geram código para a arquitetura RISC-V.

A seguinte sequência de itens sumariza o processo:

1. Escrever um código em linguagem de montagem do RISC-V. Sugerimos utilizar o editor de texto Visual Studio Code com a extensão "RISC-V Support", que provê destacador de sintaxe -- *syntax highlighting* -- para o código); Mas pode ser utilizado qualquer editor de texto.
2. Montar o código escrito na etapa (1), gerando um arquivo objeto (.o);
3. Executar o *linker* para converter o arquivo objeto em executável final;
4. Executar o simulador fornecendo o executável.

A seguir, as etapas serão detalhadas e exemplificadas.

Instalando o *toolkit* RISC-V em seu computador

Para montar e executar o *linker,* você irá precisar do *toolkit* RISC-V do LLVM instalado em seu sistema Linux (que você configurou no lab 01).

Em um sistema Debian-like (como o Ubuntu), você pode instalar o *toolkit* com o seguinte comando:

sudo apt-get install clang lld

Com isso, você irá ter instalado o compilador LLVM conhecido como clang (o qual vai ser usado como montador) e o linker do LLVM conhecido como lld.

Infraestrutura

Montagem e ligação

Para montar seu código em linguagem de montagem, use o comando:

clang --target=riscv32 -march=rv32gc -mabi=ilp32d -mno-relax arquivo_de_entrada.s -c -o arquivo_de_saida.o

Note o parâmetro na ferramenta clang: *--target=riscv32 -march=rv32gc -mabi=ilp32d.* Esse parâmetro indica que queremos que seja gerado um objeto que não é da arquitetura do seu computador (provavelmente x86_64) mas sim para RISC-V de 32 bits. Após essa etapa, tendo o arquivo objeto em mãos, podemos executar o ligador (*linker*) através do seguinte comando:

ld.lld arquivo_de_saida.o -o executavel.x

Simulação

Por fim, procedemos com a simulação em si.

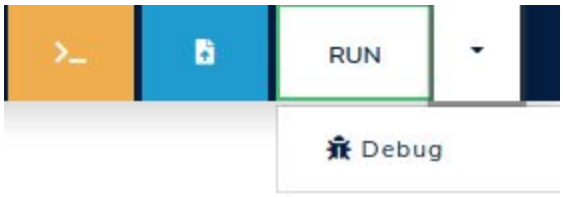
1. Acesse a página do Simulador RISC-V:
2. Clique em  *(*no canto superior direito*)* e selecione o executável
3. Clique em *Run* para iniciar a execução.
4. Observe a saída do seu código (se houver) no painel que irá abrir.
5. Encerre a execução clicando em *Stop*

Depuração

Se você adicionar a flag -g em ambos os comandos clang e lld e, ao executar o simulador, escolher a opcão debug ao invés de run, você poderá executar iterativamente sua aplicacão e ver os resultados instrução por instrução.

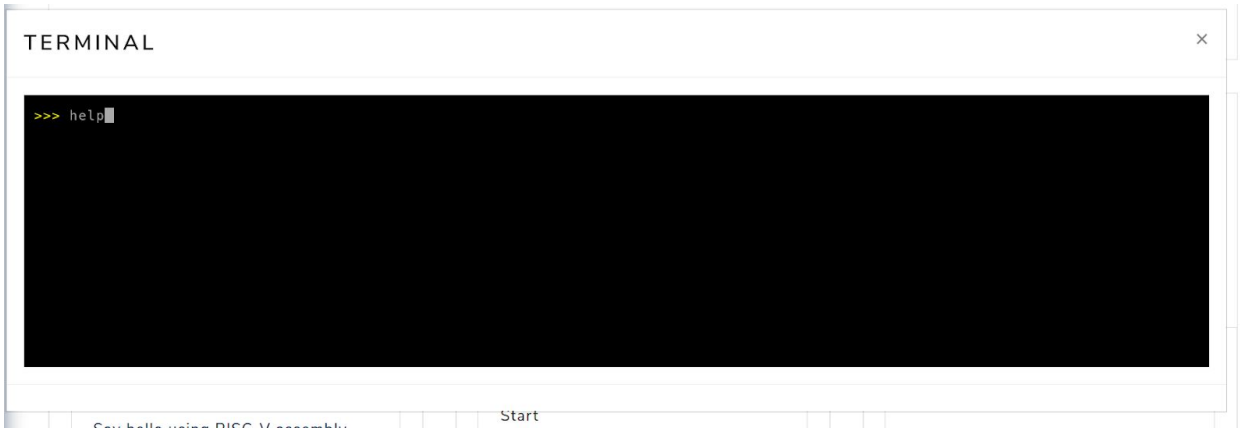
Para executar uma aplicação no modo iterativo:

1)



<- opcão de execução para debug

2)



^ terminal iterativo

Os comandos do terminal iterativo são:

```
run
  Run till interrupted.
until <address>
  Run until address or interrupted.
step [<n>]
  Execute n instructions (1 if n is missing).
peek <res> <addr>
  Print value of resource res (one of r, f, c, m) and address addr.
  For memory (m) up to 2 addresses may be provided to define a range
  of memory locations to be printed.
  examples: peek r x1   peek c mtval   peek m 0x4096
peek pc
  Print value of the program counter.
```

```
peek all
  Print value of all non-memory resources
poke res addr value
  Set value of resource res (one of r, c or m) and address addr
  Examples: poke r x1 0xff  poke c 0x4096 0xabcd
symbols
  List all the symbols in the loaded ELF file(s).
quit
  Terminate the simulator
```

Os principais comados que irão te ajudar nos próximos labs e portanto você deve se familiarizar são:

- run (executa a aplicação);
- until (executa a aplicação até um endereço específico);
- step (executa as próximas n instruções);
- peek (escreve na tela o valor de um registrador ou endereço de memória); e
- poke (seta o valor de um registrador ou endereço de memória).

Por exemplo, para executar uma instrução do seu programa e escrever na tela o valor do registrador x10:

```
$ whisper /working/modelo.x --newlib --setreg sp=0x7FFFFFC --interactive --isa acdfimsu
wasm streaming compile failed: TypeError: Failed to execute 'compile' on 'WebAssembly': Incorrect response MIME type. Expec
lication/wasm'.
falling back to ArrayBuffer instantiation
Calling stub instead of sigaction()
>>> step 1
#1 0 000110b4     4505 r 0a         00000001  c.li     x10, 0x1
>>> peek r 10
0x00000001
>>>
```

Entrega e avaliação

- **Você deve submeter APENAS um arquivo denominado raXXXXXX.s** (em que XXXXXX é seu RA com 6 dígitos) que contenha o código em linguagem de montagem.

  A entrega da atividade deverá ser feita no sistema Moodle.