Atividade de Laboratório 5

Objetivos

O objetivo deste laboratório é familiarizar a turma com a infraestrutura para montagem e execução de código ARM, 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 ARM, além de depurar o código em linguagem de montagem.

Descrição

Neste laboratório, você deve fazer um programa em linguagem de montagem do ARM 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` (./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 `r2`.

Executar um programa escrito em linguagem de montagem do ARM exige o uso de um simulador ARM, pois os computadores do laboratório (e a maioria dos computadores de uso pessoal, exceto dispositivos móveis) possuem processadores com conjunto de instruções da família de arquiteturas x86, sendo assim incompatíveis com código ARM.

Desse modo, a não ser que se utilize um *hardware* ARM, 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 ARM

O simulador da arquitetura ARM usado nessa disciplina foi criado no IC, e usa como base a linguagem descritiva de arquiteturas ArchC (http://archc.sourceforge.net (http://archc.sourceforge.net)), também desenvolvida no IC. A entidade simulada é na verdade uma placa denominada i.MX53, da empresa Freescale (http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=IMX53_FAMILY (http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=IMX53_FAMILY)), que utiliza o processador ARM Cortex-A8, que por sua vez implementa a arquitetura ARM v7. O simulador implementa não só o processador, mas também a unidade de gerenciamento de memória, alguns coprocessadores, dispositivos gerenciadores de UART (interface serial) e cartões SD, além de outros módulos da placa i.MX53.

O funcionamento do simulador pode ser resumido da seguinte forma: ao iniciar a simulação, o endereço do PC (*Program Counter*) é 0 e um código residente em uma *Read-Only Memory* (ROM) deve ser carregado na memória RAM. Tal código, que na placa é um *firmware* responsável por inicializar dispositivos e permitir que um sistema operacional seja carregado, no simulador é chamado `Dumboot` e é similar ao código da placa. Este código inicializa controladores de dispositivos e permite a carga de um sistema operacional.

O sistema operacional usado na placa i.MX53 é o Linux; no simulador, um sistema operacional mais simplificado, chamado `DummyOS`, é utilizado. 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 ARM, 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 ARM v7.

Uma vez tendo o código executável em mãos, devemos criar uma imagem de um cartão SD a ser fornecido ao simulador, para que ele possa executar o código gerado pelo *linker* na etapa anterior. Tal imagem de cartão SD é criada de modo a conter não apenas o programa que desejamos executar, mas também o código do sistema operacional `DummyOS`. Por fim, podemos invocar o simulador fornecendo a imagem de cartão SD e um arquivo que representa o código de inicialização `Dumboot`. A seguinte sequência de itens sumariza o processo:

1. Escrever um código em linguagem de montagem do ARM;
2. Montar o código escrito na etapa anterior, gerando um arquivo objeto (`.o`);
3. Executar o *linker* para converter o arquivo objeto em executável final;
4. Gerar uma imagem de cartão SD contendo o executável gerado na etapa anterior juntamente do sistema `DummyOS`;
5. Executar o simulador fornecendo a imagem de cartão SD e um arquivo que contém o código `Dumboot`.

A seguir, as etapas serão detalhadas e exemplificadas.

Infraestrutura

Todo o ferramental necessário para se trabalhar com o simulador, como montador, *linker*, gerador de imagem de SD e o simulador em si estão disponíveis nos laboratórios em `/home/specg12-1/mc404/simulador`. A primeira etapa antes de sequer usar o montador é executar o seguinte comando:

```
source /home/specg12-1/mc404/simulador/set_path.sh
```

Esse comando inicializa variáveis de ambiente necessárias para o bom funcionamento do conjunto de ferramentas. A partir desse ponto, todos os comandos podem ser executados de qualquer diretório.

Montagem e ligação

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

```
arm-eabi-as arquivo_de_entrada.s -o arquivo_de_saida.o
```

Note o prefixo `arm-eabi-` na ferramenta `as` (GNU *assembler*) - esse prefixo indica que estamos usando um executável diferente do montador (*assembler*) nativo, capaz de realizar compilação cruzada. Após essa etapa, tendo o arquivo objeto em mãos, podemos executar o ligador (*linker*) através do seguinte comando:

```
arm-eabi-ld arquivo_de_entrada.o -o arquivo_de_saida_do_ligador -Ttext=0x77802000 -Tdata=0x77803000
```

Observe que há dois parâmetros não usuais na chamada do ligador: `-Ttext` e `-Tdata`. Tais parâmetros indicam os endereços iniciais de montagem do código e dos dados descritos nas seções `.text` e `.data`, respectivamente. Você deve certificar-se de que o endereço inicial da seção de `.data` é maior do que a soma do endereço inicial com o tamanho da seção `.text`. Você também deve tomar cuidado para não colocar um endereço muito distante, pois isso tornaria o tamanho da imagem do cartão SD demasiadamente grande, tornando a simulação mais lenta. A diferença entre os dois valores (no nosso caso, 0x77802050 - 0x77802000 = 50) pode ser estendida no caso de programas com muitas instruções. Para a maioria dos casos, no entanto, esses valores são bons.

Geração da imagem do cartão SD

Uma vez gerado o executável na etapa anterior, vamos criar uma imagem de cartão SD que inclua o executável e o `DummyOS`. Para tanto, devemos usar o seguinte comando:

```
mksd.sh --so /home/specg12-1/mc404/simulador/dummyos.elf --user
arquivo_de_saida_do_ligador
```

Note que o arquivo `dummyos.elf`, que representa o sistema operacional `DummyOS`, está sendo passado como argumento, junto do arquivo de saída do ligador. Um arquivo denominado `disk.img` será gerado no diretório corrente como resultado da execução do comando acima. **Não se esqueça de, a cada vez que modificar seu código-fonte, gerar novamente a imagem do cartão SD**, caso contrário, você não estará atualizando de fato o executável que será simulado. Dica: crie um `Makefile` para gerenciar esta tarefa automaticamente para você!!!

Simulação

Por fim, procedemos com a simulação em si. Para tanto, basta executar o seguinte comando:

```
arm-sim --rom=/home/specg12-1/mc404/simulador/dumboot.bin --sd=disk.img
```

Note que os argumentos passados para a ferramenta `arm-sim` (executável do simulador) são o código `Dumboot` e a imagem de SD criada no passo anterior. A simulação então ocorre - o simulador é bastante "verborrágico", sendo que a parte essencial da saída aparece após a linha "`Booted DummyOS.`". Uma característica importante a ser notada aqui é: o sistema operacional `DummyOS` funciona baseado num laço infinito, que espera por processos a serem executados. No nosso caso, uma vez que o executável é finalizado, o sistema operacional permanece no laço indefinidamente, esperando novos processos. **Assim, deve-se terminar a simulação com o comando `ctrl+C`**.

Depuração

Por vezes, encontrar um erro num código-fonte em linguagem de máquina não é trivial. Podemos usar a ferramenta GNU gdb para permitir a execução passo a passo do programa e encontrar o erro mais facilmente. Em geral, as etapas para se depurar um programa com o gdb são:

1. Compilar/montar o código-fonte com o parâmetro de depuração (`-g`) ativado;
2. Ligar o(s) arquivo(s) objeto também com o parâmetro de depuração ativado;
3. Executar o comando `gdb seu_programa` para invocar o gdb e começar a depuração.

No nosso caso, depuraremos uma aplicação escrita em linguagem de montagem do ARM num computador da família x86, logo algumas etapas adicionais são necessárias. Em primeiro lugar, é preciso compilar e ligar seu código com a *flag* de depuração `-g` ativada. Para tanto, use os seguintes comandos:

`arm-eabi-as arquivo_de_entrada.s -g -o arquivo_de_saida.o`

`arm-eabi-ld arquivo_de_entrada.o -g -o arquivo_de_saida_do_ligador -Ttext=0x77802000 -Tdata=0x77802050`

Após isso, **gere normalmente a imagem do cartão SD**. Em seguida, será necessário que você utilize 2 terminais, um para o simulador e outro para o gdb. No terminal do simulador, execute o seguinte comando:

`arm-sim --rom=/home/specg12-1/mc404/simulador/dumboot.bin --sd=disk.img -g`

Nesse momento, o simulador será iniciado, mas irá aguardar uma conexão do gdb na porta 5000. Para efetuar essa conexão e iniciar a simulação, é preciso executar o programa gdb no outro terminal com a seguinte linha de comando:

`arm-eabi-gdb arquivo_de_saida_do_ligador`

O gdb é então inicializado, e podemos entrar com comandos para que ele se conecte ao simulador e permita realizar a depuração do código. Para tanto, entre com o seguinte comando no gdb:

`target remote localhost:5000`

Esse comando conecta o gdb ao simulador; note, no outro terminal, que o simulador está a ponto de começar a simulação, não tendo de fato começado devido ao gdb, que está no controle da aplicação. Para analisarmos o código da aplicação de interesse e não do sistema `DummyOS`, vamos atribuir um *breakpoint* na função `_start`, através do comando **`b _start`**, no gdb (`b` é abreviação de *break*). Em seguida, podemos prosseguir com a execução usando, para isso, o comando **`continue`** no gdb. Repare que o simulador irá executar até a primeira linha de sua função `_start`, então o gdb irá parar a execução para que você possa efetuar a depuração. Nesse ponto, use o comando **`si`** (abreviação de *step instruction*) no gdb para executar seu programa passo a passo. Pode-se usar o comando **`info register`** para ver o valor dos registradores. Note que a saída e a entrada padrão da sua aplicação continuará ligada à janela do terminal que está executando o `arm-sim`, e não à janela do terminal que está executando o `gdb`. Mais comandos do gdb estão disponíveis no documento `apostila_ARM.pdf` (./apostila_ARM.pdf) e no manual da ferramenta gdb em `https://sourceware.org/gdb/current/onlinedocs/gdb` (https://sourceware.org/gdb/current/onlinedocs/gdb).

Informações importantes/Dicas

- Para redirecionar a saída do simulador para um arquivo, execute o seguinte comando:
  `(arm-sim --rom=/home/specg12-1/mc404/simulador/dumboot.bin --sd=disk.img 2>&1) > raXXXXXX.out`
  Note que você ainda precisa disparar o comando `ctrl+C` para terminar a simulação! Após o término, o arquivo `raXXXXXX.out` conterá a saída do simulador.
- Para maiores informações sobre o simulador, inclusive como utilizá-lo em conjunto com o GDB para depuração, veja o documento `apostila_ARM.pdf` (./apostila_ARM.pdf).
- Alguns comandos ficam extensos devido a caminhos longos, como `/home/specg12-1/mc404/simulador/dummyos.elf`. Pode-se utilizar a ferramenta `ln` para criar *links* simbólicos e reduzir o tamanho dos comandos. Podemos, por exemplo, fazer:
  `ln -s /home/specg12-1/mc404/simulador/dummyos.elf dummylink`
  de modo que agora não precisaremos mais colocar todo o caminho para `dummyos.elf`, bastando colocar

`dummylink` no lugar. O comando para gerar a imagem do SD ficaria então: `mksd.sh --so dummylink --user arquivo_de_saida_do_ligador`.

Entrega e avaliação

Endereço da atividade no sistema SuSy: https://susy.ic.unicamp.br:9999/mc404abef/05ab (https://susy.ic.unicamp.br:9999/mc404abef/05ab) ou https://susy.ic.unicamp.br:9999/mc404abef/05ef (https://susy.ic.unicamp.br:9999/mc404abef/05ef).

- **Você deve submeter APENAS um arquivo comprimido denominado raXXXXXX.tar.gz** (em que XXXXXX é seu RA com 6 dígitos) que contenha tanto o código em linguagem de montagem quanto a saída do simulador. Para comprimir os arquivos, use o comando "`tar -czf raXXXXXX.tar.gz raXXXXXX.s raXXXXXX.out`".