UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Relatório do trabalho de Thumb

Arquitetura e Organização de Computadores II

Autores:

David Machado - 475664

Antonio César - 473444

Anderson Moura - 473070

Professor: Roberto Cabral

Quixadá - Ceará
24/02/21

Conteúdo

1 Introdução

O conjunto de instruções THUMB é um "rearranjo" de 16 bits das instruções ARM de 32 bits. Tem como objetivo, principalmente, ser usado em programas que possuem espaço limitado e em microcontroladores com barramento de dados de 16 bits. Um código implementado em THUMB ocupa, em média, 30% menos espaço que o mesmo código implementado em ARM de 32 bits, o que, como dito antes, é perfeito para sistemas em dispositivos móveis de telefonia e PDAs, por exemplo, que possuem restrição de memória. Além de também aumentar o desempenho em sistemas com barramento de dados de 16 bits. O mesmo não ocorre caso o sistema tiver um barramento de 32 bits.

Este trabalho tem como objetivo entender a estrutura das instruções THUMB, implementando um tradutor de instruções em assembly para sua versão em hexadecimal.

2 Organização e estrutura

A linguagem escolhida para escrever os códigos no trabalho foi C e o no total teve três arquivos .c e .h e um arquivo Makefile para execução dos códigos. Os três arquivos cada um possuem um objetivo, os três arquivos são nomeados: main.c , map.c e Decode.c. A main.c tem como objetivo servir de interface com o usuário e receber o arquivo .in de entrada e produzir o .out. A map.c é responsável por direcionar o número hexadecimal para a função que vai escrever instrução equivalente a ele. O arquivo Decode.c tem as funções que quando recebem o número depois de ser mapeado pela map.c, vão gravar no arquivo de saída a instrução equivalente.

3 Arquivos

Um objetivo do trabalho é receber um arquivo de entrada que possui o mapa de memória e com isso gerar um arquivo texto de saída com as instruções em THUMB. Na linguagem C e em especifico na biblioteca stdio, existe um conjunto de funções para lidar com arquivos.

Para poder ler e escrever dados em arquivos no C, é necessário criar ponteiros do tipo FILE que estão apontados para os arquivos. Com isso na main.c do trabalho existe dois ponteiros, ptr_input e ptr_output que vão ser os ponteiros para os arquivos input.in e output.text respectivamente, isso foi feito utilizando a função fopen e foi passado o parâmetro "r"para o ptr_input e com isso só permitido leitura no arquivo de entrada e com o ptr_output foi passado o parâmetro o "w"que permite a escrita no arquivo.

Logo depois é feito uma verificação para ver se a função fopen funcionou e isso é feito fazendo duas verificações com os ponteiros para ver se os valores deles e NULL. Com isso, agora é obter as linhas do arquivo de entrada e isso é feito com a função fgets() que salva a linha em uma string que no trabalho é chamada de line, e fazendo um while com a condição que fgets() não seja NULL, permite que se obtenham todas as linhas do arquivo de entrada. Antes da chamada da função para mapear, é feito uma conversão de string para um número hexadecimal e assim chamar a função de mapear.

4 Função de mapeamento

A função de mapeamento, basicamente tem o papel de redirecionar os números hexadecimais para suas funções de decodificação correspondentes, pois vale lembrar que as instruções feitas por nossa equipe estão encapsuladas em funções de decodificação que serão explicadas nos tópicos posteriores desse relatório.

A função de mapeamento foi pensada de forma similar a uma arvore, onde temos uma raiz e dela várias ramificações até suas folhas.

Para isso, foi preciso contar com duas coisas:

1. Função de exceção
2. Caminhos

Começando pelos caminhos, temos que o primeiro caminho que ramifica as funções de decodificação é o número formado pelos bits 12,13,14 e 15. Dele podemos derivar folhas (por exemplo a função DecodeLSL_LSR_LD_LM_IMM5 que irá levar por exemplo a instrução "*LSL Ld, Lm, #<immed5>*").

Assim como também, podemos derivar um grupo, por exemplo "*fgrupo1*"que é um nó onde dele podemos nos ramificar nas funções DecodeASR_LD_LM_IMM5, DecodeADD_SUB_LD_LN_LM, DecodeADD_SUB_LD_LN_IMM3 a partir de um novo caminho que será o número formado pelos bits 10 e 11. Porém, vale lembrar que um grupo além de poder ramificar se em folhas (funções de decodificação), ele também poderá se ramificar em outro(s) grupo(s), que é o que chamamos de subgrupos daquele nó.

Outro ponto importante é como esse caminho nos levar até nossa funções de decodificação. Isso é feito com o uso de matrizes de ponteiros que referenciam ou "apontam" para funções de decodificação. E assim como deslocamos em uma matriz ou array por um índice de 0 até n-1, faremos o mesmo a cada nó tomando esses caminhos como um índice que será um offset no ponteiro que irá referenciar as funções.

Entretanto, a tabela com as instruções thumb usada possui certos números hexadecimais que não tem uma instrução correspondente, por exemplo 0x4400. E aqui surge a necessidade de funções de exceção. Onde para os grupos (ou subgrupos, ou nós) contamos com "*if*","*else*" e a função underfined para se adequar apenas as instruções em tabela.

Por exemplo, as verificações feita na função mapFunction para tratar os números hexadecimais que alterassem o valor "*poff*".

Por fim, vale ressaltar duas coisa:

1. Caminhos serão números formados por certos bits que terão o papel de índice nos ponteiros que referenciam para funções.
2. Funções de exceção serão tratamentos para adequar nossa função de mapeamento apenas para as instruções em tabela.

5 Compilação e execução

O trabalho vem acompanhado de um Makefile que permite a execução dos códigos de maneira simples, basta executar o comando make que vai ser feita a compilação e gerar o executável exec. Antes de executar o comando ./exec, necessário que o nome do arquivo de entrada seja input e que é um .in, e com isso o programa pode ser executado com o comando ./exec.
Comandos para executar o programa com o Makefile:

```
$ make
$ ./exec
```

Agora para compilar sem o Makefile

```
$ gcc -c Decode.c
$ gcc -c map.c
$ gcc -o exec main.c Decode.o map.o
$ ./exec$
```

No inicio do programa vai aparecer um menu com duas opções: executar e sair. Aperte 1 para executar e gerar/alterar o output ou aperte 2 para sair. Quando o programa for executando, vai ser mostrado uma mensagem que o foi terminado o processo com sucesso e que foi finalizado com sucesso.

6 Conclusão

Foi feito o mapeamento completo das instruções THUMB, assim como sua tradução, porém ainda ficaram algumas dúvidas sobre valores incluídos em algumas instruções. Deixando esses problemas de lado, foi uma experiência significativa para o entendimento da estrutura dessas instruções e de como elas são interpretadas pela máquina.

7 Referência

(2020). Armconverter. https://armconverter.com/?disasm&code

Andrew N.Sloss, Dominic Symes, e.a. (2004). ARM System Developer's Guide Designing and Optimizing System Software. *Elservier*.

ARM. (2000). ARM Architecture Reference Manual.