Escola Politécnica da Universidade de São Paulo

PCS3722 - Organização e Arquitetura de Computadores II

Augusto Barbosa Villar Silva	**11831853**
Emilly da Silva Arcanjo	**11808105**
Thiago Moreira Yanitchkis Couto	**11804081**
Victor de Almeida Santana	**11806718**

RELATÓRIO

Projeto - PCS3732

15/08/2023

SUMÁRIO

1. INTRODUÇÃO

O projeto final definido para a disciplina PCS3732 - Laboratório de Processadores, tem por objetivo a implementação de uma solução simples de tradutor que permita ao usuário inserir uma sequência de instruções em um formato de instruções (A32 ou T16) e receber como saída a sequência de instruções equivalentes em outro formato (T16, A64 ou A32).

Foram definidas para o escopo do projeto as seguintes traduções:

- A32 para T16.
- A32 para A64.
- T16 para A32.

2. DEFINIÇÃO DE ESCOPO E MOTIVAÇÕES

2.1. Motivações

A proposta de projeto se baseia no conteúdo estudado na aula 8 da disciplina, cujo tema principal foi o conjunto de instruções Thumb (T16), o qual consiste num conjunto de instruções de 16 bits que objetiva diminuir a quantidade de memória necessária para armazenar um programa, além de melhorar a vazão, permitindo a execução de um maior número de instruções num espaço menor de tempo.

Assim, a partir do estudo do uso das instruções Thumb em sala de aula, assim como o intercâmbio entre ARM e Thumb, surge a motivação do desenvolvimento de um tradutor que facilite a conversão de um conjunto de instruções de um formato para outro, dado que é bastante comum no processo de programação, conhecer uma lógica a ser implementada, mas enfrentar gargalos quanto a implementação em si, numa certa linguagem ou formato não conhecido.

2.2. Definição de Escopo

Inicialmente, desejava-se definir um escopo ainda maior do que o apresentado para o projeto, apresentando uma extensão do visualizador de instruções[1] apresentado nos materiais da disciplina, que permitiria a tradução de uma única instrução aliada à sua visualização nos formatos T16, A32 e A64. Porém, a escolha do grupo seguiu pelo desenvolvimento de uma estrutura mais similar aos tradutores linguísticos utilizados no dia-a-dia, com o comportamento de input e output, assim como a seleção de linguagens (no caso, formato).

Dessa forma, o escopo do projeto definido compreende uma interface Web desenvolvida em React, que apresenta um campo de input para receber um conjunto de instruções, cada qual separada uma da outra por quebra de linha; um campo para output, e um campo do tipo drop-down que permite selecionar a que conjunto as instruções de input pertencem.

Vale ressaltar que o escopo definido está sujeito aos prazos previstos para a disciplina, de modo que seria possível acrescentar as ideias inicialmente propostas, no caso de futuras extensões e próximos passos no desenvolvimento do trabalho.

3. O PROJETO

3.1. Considerações Gerais

Devem ser destacadas algumas considerações e hipóteses assumidas no desenvolvimento do projeto.

Em primeiro lugar, assume-se que o usuário do programa desenvolvido tenha mínima noção de linguagem assembly, de modo que o tratamento de absurdos não recebe tanta atenção, mas ainda sim há um indicativo de instruções não existentes ou que não possuem tradução. Entretanto, caso o usuário tenha por objetivo não usar o programa corretamente, há grandes chances do comportamento do mesmo apresentar absurdos, não tratando todos os potenciais problemas.

A hipótese anteriormente levantada é consequência da necessidade de uma testagem mais extensiva do programa desenvolvido, o que não pôde ser compreendido dentro do escopo do curso. Desse modo, o programa foi testado apenas em cenários em que assume-se um usuário minimamente bem intencionado e competente na linguagem assembly.

Em segundo lugar, foram assumidas como prioridade as traduções que envolvem o formato Thumb, dado a extensão muito significativa das instruções do formato A64. Dessa forma, a recíproca da tradução do A32 para o T16 foi implementada no escopo do programa, o que não se verifica para o conjunto A64. Entretanto, uma extensão possível para o programa envolve justamente a implementação da recíproca faltante.

Por fim, a preocupação com a beleza do código e desempenho foram preteridas em função do funcionamento do programa, ou seja, o grupo objetivou priorizar a apresentação de um projeto funcional e completo dentro do escopo previsto. Entretanto, é válido ressaltar que escolhas de desempenho foram realizadas em relação às estruturas de códigos desenvolvidas, de modo que, mesmo que não seja o principal propósito, o projeto se encontra relativamente preparado para escalabilidade em termos de desempenho, embora apresente estruturas de código que permitem extensões, mas tornam difíceis modificações.

3.2. A32 para T16

Para realizar a tradução de instruções ARM 32 para Thumb 16, foi tomada por base a documentação[5] do instruction set do Thumb. Sendo assim, optou-se pela priorização das instruções existentes em Thumb 1, de modo que as extensões presentes no Thumb 2 não foram abarcadas pelo projeto.

A base tomada foi em relação a instruções com tradução direta, ou seja, aquelas em que, uma instrução em ARM gera uma instrução em Thumb, de modo que, instruções cuja tradução se daria de modo a criar mais de uma instrução em Thumb, foram preteridas em prol de garantir que o tratamento das instruções escolhidas fosse o mais completo possível. Sendo assim, as instruções cuja tradução não é definida, foram mapeadas para a expressão UND (undefined), gerando uma mensagem para o usuário da não existência de um equivalente direto.

Ainda no que diz respeito às prioridades definidas para a tradução do ARM para o Thumb, definiu-se o não tratamento de pseudo-instruções. Ressalta-se que a lógica para o tratamento de pseudo-instruções seguiria a mesma definida para o tratamento do escopo determinado, de modo que extensões seriam possíveis, entretanto, a escolha se deu pelo fato de restringir o projeto a um escopo menor e funcional, em vez da extensão a um escopo maior e mais problemático que poderia prejudicar o pleno funcionamento no prazo estipulado.

Vale ressaltar que as instruções mapeadas envolveram o tratamento de condicionais e extensões via sufixos, sendo mapeadas para seus equivalentes em Thumb, de modo que o usuário pode não se preocupar com tratar à parte a maioria dos sufixos existentes para cada instrução.

A manipulação de registradores também foi tratada, garantindo que uma instrução só seria válida quando a manipulação dos registradores High e Low estivesse acordada com o definido no instruction set do Thumb. Sendo assim, instruções em que ocorre uma manipulação inadequada dos registradores também retornam um feedback visual para o usuário que permite entender o formato esperado para a instrução desejada.

3.3. A32 para A64

3.3.1. Visão geral do A64

O A64 possui 31 registradores de propósito geral, que podem ser usados como um registrador de 64 bits (X0 - X30) ou de 32 bits (W0 - W30). O registrador X30 é usado como LR e, diferente do A32, no A64 o PC e o SP não são registradores de propósito geral e possuem uma restrição quanto ao seu acesso.

As únicas instruções que podem ler o PC são as que o utilizam como base de endereços relativos, como load, store e branch. Além disso, seu valor só pode ser modificado por instruções de realização ou retorno de desvios, ou aquelas de geração ou retorno de exceções. Já o SP só pode ser usado como endereço base

em loads e stores, como registrador de origem ou destino em instruções de adição ou subtração, e como registrador de destino em instruções lógicas.

As operações lógico-aritméticas do A64 possuem, em geral, o mesmo formato do A32: o destino e o primeiro operando são sempre registradores e o segundo operando pode ser um registrador e incluir um shift ou pode ser um imediato. Uma diferença nesse caso, é que a quantidade de shifts só pode ser determinada por um imediato e não pelo valor de um registrador, como é possível no A32. O sufixo “S” também pode ser usado para acionar as flags, que no caso do A64 ficam em um registrador de 64 bits chamado NZCV.

As operações de load e store também, basicamente, possuem o mesmo formato do A32. No caso do A64, a diferença é o que o registrador base deve ser sempre um de 64 bits (X) e que ao usar um registrador para obter o valor de offset do endereço, ele só pode ser alterado por um shift do tipo LSL.

As instruções no A64 não são todas condicionais como no A32, as únicas que podem utilizar condições e possuem representantes análogas no A32 são as instruções de desvio, de adição ou subtração com carry (ADC, SBC) e comparações condicionais (CCMP e CCMN). Outras operações condicionais do A32 podem ser substituídas por uma instrução condicional que a representa, seguida pela instrução condicional “CSEL <Wd>, <Wn>, <Wm>, <cond>”, que escreve o valor do primeiro registrador de entrada no registrador de destino se a condição for verdadeira, ou escreve o valor do segundo registrador de entrada, caso seja falsa.

Assim como no A32, `B <label>` realiza um desvio incondicional para o endereço PC+offset, sendo o offset representado pela label. Entretanto, a instrução que realiza um desvio para um endereço presente em um registrador é diferente da do A32, ela usa um registrador de 64 bits e possui um opcode diferente: `BR <Xn>`. As instruções de desvio condicional também possuem um formato diferente: `B.cond <label>`. As condições que podem ser usadas são as mesmas do A32 (sufixos como EQ para *equal*, NE para *not equal*, dentre outros).

Por fim, não existem mais instruções que manipulam múltiplos registradores (LDM, STM, PUSH, POP), só instruções de load-store para um par de registradores não contíguos. O conceito de coprocessadores também foi removido da arquitetura.

3.3.2. Implementação do tradutor A32 => A64

A definição do escopo do tradutor de instruções do A32 para A64 foi feita visando abranger as instruções mais comuns e que possuem tradução direta de uma instrução para outra, como load/store, instruções lógico-aritméticas e de desvio. Tais parâmetros permeiam a maior parte do conjunto definido, mas existem exceções, como algumas instruções condicionais do A32 que não possuem tradução direta, mas foram incluídas.

A limitação do escopo de instruções foi definida por conta da complexidade do conjunto de instruções do A64 e da dificuldade em encontrar comparações diretas entre as instruções A32 e A64, sendo necessário verificar uma por uma na documentação do ARM. Assim, devido a essa dificuldade, junto ao prazo para realização do projeto, a equipe priorizou as instruções mais comuns.

As instruções do A32 aceitas são: ADC, ADD, AND, B, BIC, BL, BX, CMN, CMP, EOR, LDR, MLA, MOV, MUL, MRS, MSR, MVN, ORR, SBC, STR, SUB e TST. Sendo que elas podem ser inseridas com os sufixos condicionais ou o “S” que modifica as flags, e podem usar o shift, quando cabível. Caso alguma instrução fora desse escopo seja inserida, o programa mostrará um aviso: “Não foi possivel encontrar a instrução”.

No caso em que a instrução possui um formato não aceito para ser traduzido para o A64, o programa informa outros tipos de erro, especificando mais a causa. Por exemplo, para uma instrução do A32 com shift no segundo operando e com a quantidade de shifts sendo determinada por um registrador, o que no A64 só pode ser determinado por um imediato, aparecerá o erro “Erro na disposição dos registradores e imediatos para a tradução”.

Para implementar a tradução, utilizou-se um dicionário para mapear os nomes das instruções A32 ao seu correspondente no A64, e outro com o nome da instrução no A64 mapeada a um array, cujos valores indicam como é o formato dela: quantidade e tipo de operandos (imediato/registrador), e posição do shift (caso aceite). Além disso, são usados vetores para guardar os sufixos que as instruções podem ter e os tipos de shift que podem ser aplicados em alguns casos.

Ao receber a instrução escrita pelo usuário, verifica-se o nome da instrução e se ela possui algum sufixo, para então realizar a tradução a partir das estruturas mencionadas, verificando número de operandos e se o tipo é compatível com o conjunto de instruções do A64. Isso é feito basicamente com estruturas de loop e estruturas condicionais no código (switch/case, if/else e for). No fim, é feito o

tratamento dos registradores, que são traduzidos para o registrador de 32 bits do A64 (W), sempre que possível (em alguns casos o registrador de 64 bits (X) tem que ser usado, como quando representa o registrador base de um endereço). Ademais, as instruções condicionais do A32, com exceção das de desvio, são traduzidas para 2 outras instruções, já que no A64, basicamente, só as instruções de desvio são condicionais.

3.4. T16 para A32

Para realizar a operação inversa ao apresentado no tópico 3.2, ou seja, Thumb 16 para ARM 32, tomou-se por base a documentação do instruction set do Thumb, de modo que as instruções foram traduzidas visando atingir somente o grupo de instruções cuja tradução se dá de maneira direta do Thumb 1 para o A32, ou seja, instruções presentes no ARM que não possuem equivalência no Thumb não foram priorizadas no escopo definido para o projeto.

O tamanho e formato das instruções inseridas no input são tratados tal como o que é realizado no caso da tradução de A32 para T16, ou seja, no caso de serem inseridas instruções com formato incorreto, ou desconhecido, a interface emite no output um aviso concernente ao erro enfrentado.

A lógica de tradução segue o mesmo desenvolvido na tradução inversa, ou seja, foram definidos um vetor de opcodes para o A32 e para o T16 e houve um mapeamento de um para o outro, havendo o devido tratamento no que diz respeito ao formato, tamanho, registradores usados, dentre outros aspectos no corpo da função de tradução, a qual faz uso de funções auxiliares que permitem o tratamento de casos mais complexos e específicos, cuja tradução não se dá de maneira tão direta.

De modo geral, o tratamento dos opcodes é mais direto (comparado ao de A32 para T16), precisando de pouca diferenciação entre as diferentes instruções de um mesmo opcode.

Uma das instruções cujo tratamento muda é a instrução de branch, seja ela condicional ou não. Para tais instruções é necessário separar o opcode (“B”) dos sufixos condicionais (“GT”, “LE”, “AL”, ...). A separação permite o tratamento de todas as instruções de branch de uma vez só, a partir do opcode “B”.

Outra diferenciação de tratamento ocorre no opcode “ADD”, quando ele é utilizado com o Stack Pointer (SP ou R13) e com um imediato negativo (formato 13 do instruction set), ele é traduzido para ARM 32 bits como um sub com o imediato positivo.

4. CONCLUSÃO

O projeto desenvolvido para a disciplina de Laboratório de Processadores explora os conceitos aprendidos ao longo do curso, com foco na tradução de conjuntos de instruções entre diferentes formatos, nomeadamente A32, T16 e A64.

O tradutor apresentado oferece uma interface amigável para que os usuários possam inserir sequências de instruções em um formato e obter suas contrapartes nas outras arquiteturas suportadas. As traduções A32 para T16, A32 para A64 e T16 para A32 foram abordadas com enfoque nas instruções mais comuns e de tradução direta. O projeto também considerou limitações temporais e a complexidade das arquiteturas, priorizando um conjunto abrangente e funcional de traduções, sem comprometer a qualidade do resultado final.

Durante a implementação, foi necessário enfrentar desafios relacionados à complexidade das arquiteturas, compatibilidade de operandos e formatos, bem como a determinação das traduções adequadas. O uso de estruturas de dados como dicionários e vetores, juntamente com lógica condicional, permitiu uma abordagem sistemática para traduzir as instruções e seus detalhes específicos. Também é importante destacar que, embora tenha havido um foco prioritário em funcionalidade e abrangência, considerações de desempenho e modularidade também foram tidas em mente.

Em suma, este projeto proporcionou uma oportunidade valiosa para aplicar os conhecimentos adquiridos em sala de aula a um contexto prático, enfrentando desafios reais encontrados em diferentes arquiteturas de processadores. Embora o escopo tenha sido delimitado para atender aos requisitos da disciplina, o projeto demonstra potencial para expansão e refinamento futuros, incluindo a implementação de traduções adicionais e melhorias na usabilidade da interface e traduções de programas inteiros.

5. REFERÊNCIAS BIBLIOGRÁFICAS

1. Bruno Basseto. ASMCLICK. Disponível em: https://www.wise-ware.com.br/aulas/arm/. Acesso em: 11 ago. 2023.

2. Documentation – Arm Developer - Learn the architecture - A64 Instruction Set Architecture. Arm.com. Disponível em: <https://developer.arm.com/documentation/102374/0101/?lang=en>. Acesso em: 13 ago. 2023.

3. Documentation – Arm Developer - Arm A64 Instruction Set Architecture. Arm.com. Disponível em: <https://developer.arm.com/documentation/ddi0596/latest/>. Acesso em: 13 ago. 2023.

4. ARMv8 Instruction Set Overview ARMv8 Instruction Set Overview Architecture Group. [s.l.: s.n.], 2011. Disponível em: <https://www.cs.princeton.edu/courses/archive/spr19/cos217/reading/ArmInstructionSetOverview.pdf>.

5. Thumb Instruction Set. Disponível em: http://bear.ces.cwru.edu/eecs_382/ARM7-TDMI-manual-pt3.pdf?ref=zdimension.fr. Acesso em: 13 ago. 2023.

6. Documentation – Arm Developer - Arm A-profile A32/T32 Instruction Set Architecture. Arm.com. Disponível em: <https://developer.arm.com/documentation/ddi0597/2023-06/Base-Instructions?lang=en>. Acesso em: 13 ago. 2023.

7. Documentation – Arm Developer - Arm A-profile Architecture Registers . Arm.com. Disponível em: <https://developer.arm.com/documentation/ddi0601/2023-06/AArch64-Registers/NZCV--Condition-Flags?lang=en>. Acesso em: 13 ago. 2023.