UNIVERSIDADE DE SÃO PAULO
Escola Politécnica

PCS3732 – Laboratório de Processadores

Professor Bruno Basseto

Relatório de Prova Prática - Tradutor de instruções THUMB

Henrique D'Amaral Matheus	**N° USP: 11345513**
Rafael Katsuo Nakata	**N° USP: 11803819**
Vinicius de Castro Lopes	**N° USP: 10770134**

SÃO PAULO, 15 de Agosto de 2023

Índice

1. Introdução.....3
2. Requisitos.....3
3. Implementação.....3
3.1 Tecnologias utilizadas..... 3
3.2 Estratégia de Tradução..... 3
3.3 Exemplos.....7
4. Testes.....8
5. Próximos passos..... 9
6. Conclusão.....9
7. Referências.....10

1. Introdução

Existem diversas ISAs (Instruction Set Architecture) relacionadas ao ARM. Na disciplina, estudamos primariamente a ARM32, mas uma que é usada mais comumente no mundo real (e.g. smartphones, desktops e embarcados) é o conjunto Thumb. Num primeiro momento, a diferença entre os dois é que o ARM32 usa instruções de 32 bits, e o Thumb usa primariamente 16 bits por instrução.

Como aprendido em aula, muitos processadores que suportam ambos os conjuntos traduzem instruções Thumb para ARM32 em tempo real.

Inspirado nisso, decidimos fazer um tradutor didático de instruções em linguagem de máquina de Thumb para ARM32.

2. Requisitos

- Traduzir codificação binária
- Visualização das instruções Thumb e ARM32 em assembly.
- Atualização dinâmica da tradução
- Exibir mensagens didáticas e úteis à tradução (e.g. erros)

3. Implementação

3.1 Tecnologias utilizadas

A implementação do tradutor de instruções THUMB para ARM32 é feita em javascript, com a interface utilizando a biblioteca React Js. As escolhas se deram pela simplicidade de implementação, boa adequação ao propósito do projeto e facilidade para realizar testes durante a implementação.

3.2 Estratégia de Tradução

Para a implementação das traduções, optou-se por realizá-la formato a formato, pois compreendeu-se que, desta maneira, o processo seria mais testável, de mais fácil manutenção e o código seria mais compreensível. Essa separação está de acordo com o manual de referência da ISA do Thumb.

Uma vez que a entrada (instrução THUMB) e a saída (instrução ARM32) são pré-definidas pelos manuais dos conjuntos de instruções, basta que se compreenda a maneira como os campos de bits são associados pelo compilador para que se possa utilizar o Desenvolvimento Guiado por Testes (Test Driven Development), facilitando assim o processo.

Para a montagem final da instrução ARM32, definiram-se instruções padrões de cada tipo para serem utilizadas pelos formatos irregulares das instruções encontradas no conjunto THUMB (19 formatos no total). Essas instruções base levam em conta limitações do conjunto THUMB, por exemplo:

- Só há acesso à memória de maneira pré-indexada.

- Por padrão, sobrescrevem bits de execução condicional.

Identificação do formato

Primeiramente ocorre a identificação do formato da instrução por meio de máscaras de bits, e, logo após, ocorre a chamada da função correspondente que tratará de traduzir a instrução. Caso a configuração de bits não corresponda a nenhum formato, é retornado o valor nulo 0x0.

Identificação dos campos da instrução

Após a identificação do formato, procura-se pelos campos contidos nele e procede-se com a adequação do campo THUMB para o conjunto ARM32. Abaixo listam-se os campos e as respectivas ações necessárias.

- **opcode (Op):** campo encontrado nos cinco primeiros formatos de instruções thumb (instruções para o processamento de dados), frequentemente utilizado como configurador do opcode correspondente para instruções ARM32, isto é, não é utilizado diretamente nos bits resultantes.
    - Há também o uso para configuração do barrel shifter (formato 1). com correspondência direta para os deslocamentos LSL, LSR, e ASR. A instrução utilizada é o MOV.
    - No formato 4 encontramos opcodes com correspondência direta entre os dois conjuntos de instruções, com exceção das instruções de deslocamento, que novamente são convertidas para instruções MOV
- **Destination register (Rd/Hd):** usado como registrador de destino, encontrado ou nos bits menos significativos da instrução, ou no bits 10 a 8. É estendido para 4 bits.
- **Source register (Rs/Hs):** usado pelos formatos 1, 2, 4 e 5 para obter o operando de uma instrução de processamento de dados, encontrado nos bits 5 a 3. É estendido para 4 bits.
- **Base register (Rb):** usado pelos formatos de instruções que acessam à memória como registrador base. É estendido para 4 bits.
- **Registrador de segundo operando (Rn):** utilizado pelo formato 2 para somas e subtrações com três registradores distintos. É estendido para 4 bits.
- **Offset register (Ro)**: usado pelos formatos de instruções que acessam à memória como deslocamento a partir do registrador base. É estendido para 4 bits.
- **Register list (Rlist):** especifica os registradores “baixos” (low) que devem ser salvos, ou restaurados, pelas instruções de memória.
- **Offset3:** campo de 3 bits, utilizado como segundo operando no formato 2. Não precisa ser alterado.
- **Offset5**: campo de 5 bits, utilizado de duas maneiras.
    - No formato 1, utiliza para deslocar o registrador fonte. Não precisa ser alterado.
    - Nos formatos 9 e 10, é utilizado para deslocar o acesso à memória com relação ao registrador base. Não precisa ser alterado.

- **Offset8**: campo de 8 bits utilizado pelo formato 3 como segundo operando de operações de processamento de dados. Não precisa ser alterado.
- **Soffset8:** campo de 8 bits utilizado pelas instruções de desvio condicional. Deve ser estendido para 24 bits, levando-se em conta a representação em complemento de 2, e deslocado uma posição para a direita, uma vez que os desvios em THUMB e ARM32 têm comprimentos diferentes. Cabe ressaltar que caso seu bit menos significativo seja 1, não é possível alcançar o endereço a partir de uma instrução ARM32 de desvio condicional.
- **Offset11:** campo de 11 bits utilizado pelas instruções de desvio incondicional. As mesmas regras de transformação do campo “Soffset8” se aplicam a este campo.
- **Offset**: campo do formato 19, que não pode ser traduzido como uma única instrução ARM32. Ignorado.
- **Sword7:** campo de 7 bits codificado em sinal magnitude. Para a tradução, deve ser deslocado de dois bits para a esquerda. Deve ser deslocado dois bits à esquerda.
- **Word8:** campo de 8 bits usado pelos formatos 11 e 12 para operações relativas ao contador de programa e ao ponteiro de pilha. Deve ser deslocado dois bits à esquerda.
- **Value8**: campo de 8 bits passado para a instrução de “software interrupt”. Não precisa ser alterado.
- **Cond:** campo usado para determinar a condição de execução de um branch. Não precisa ser alterado
- **Flag para registradores H1/H2**: usadas no formato 5 para determinar qual registrador deve ser utilizado (“high”). Se configurados como “1”, alteram os respectivos registradores da instrução.
- **Flag para imediato (I):** presente no formato 2 para decidir se o terceiro operando é um registrador ou um imediato.
- **Flag para tipo de acesso a memória (L):** indica quando um acesso a memória é para armazenar, ou recuperar, um valor.
- **Flag para halfword (H):** indica quando um acesso a memória é para armazenar, ou recuperar, uma halfword.
- **Flag para byte (B):** indica quando um acesso a memória é para armazenar, ou recuperar, um byte.
- **Flag para sinal estendido (S):** indica quando um acesso a memória é para armazenar, ou recuperar, operandos com sinal estendido. Também pode ser usado para configurar o tipo de instrução (ADD ou SUB) a ser realizado no formato 13.
- **Flag para configuração de Rs (SP):** usada para indicar se o source register será o contador de programa ou o ponteiro de pilha no formato 12.
- **Flag para armazenamento condicional (R):** usada para armazenar ou o registrador 14, ou o contador de programa nas operações PUSH e POP.

Montagem da instrução

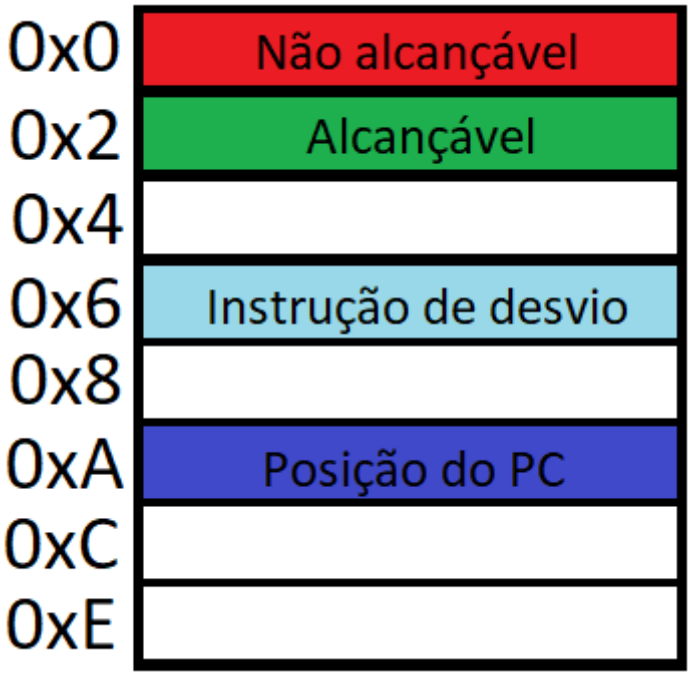
Após os campos serem capturados via máscara de bits, passamo-los como parâmetros para as funções responsáveis por montar a instrução ARM32.

- function armSWI(comment_field)
- function armBranchExchange(Rn)
- function armBranch(exec_condition, link, offset, offsetsize)
- function armLoadStore(imm_flag, byte_flag, ldr_str, Rn, Rd, offset)
- function armHSLoadStore(imm_flag, signed_flag, half_flag, ldr_str, Rn, Rd, op2_high, op2_low)
- function armMultiplication(Rd, Rn)
- function armRegTransfer(pre_pos, up_down, ldr_str, Rn, Rlist)
- function armDataProcessing(immediate_flag, opcode, set_condition_codes, Rn, Rd, operand2)

A particularidade dos desvios

Tanto os desvios condicionais quanto os desvios incondicionais merecem atenção especial, pois nem todos os endereços especificados em uma instrução THUMB é alcançável por uma instrução ARM32. Isto acontece porque as instruções dos conjuntos tem tamanhos diferentes (16 bits THUMB, 32 bits ARM32).

A regra prática é que, quando o bit menos significativo da instrução for igual a 1, o endereço não é alcançável por ARM32. Abaixo ilustramos uma memória para fins demonstrativos.

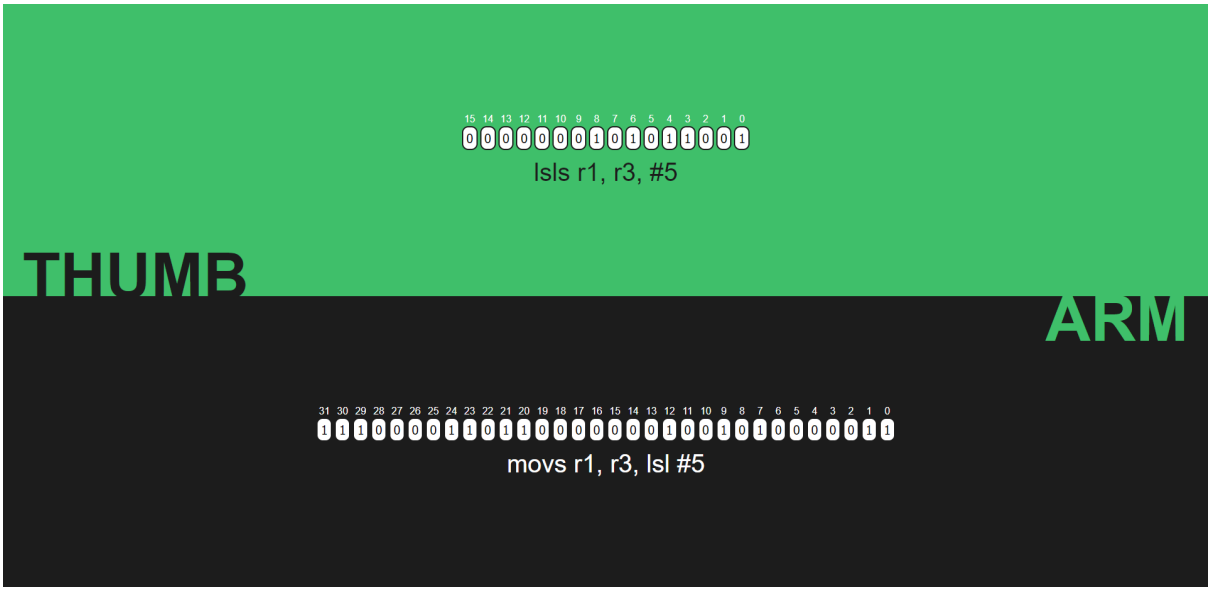


	Alcançável	**Não alcançável**
Endereço relativo ao PC	0xA (000_0000_1010)	0x8(000_0000_1000)
Armazenamento em Offset11	0x7	0x4(000_0000_0100)

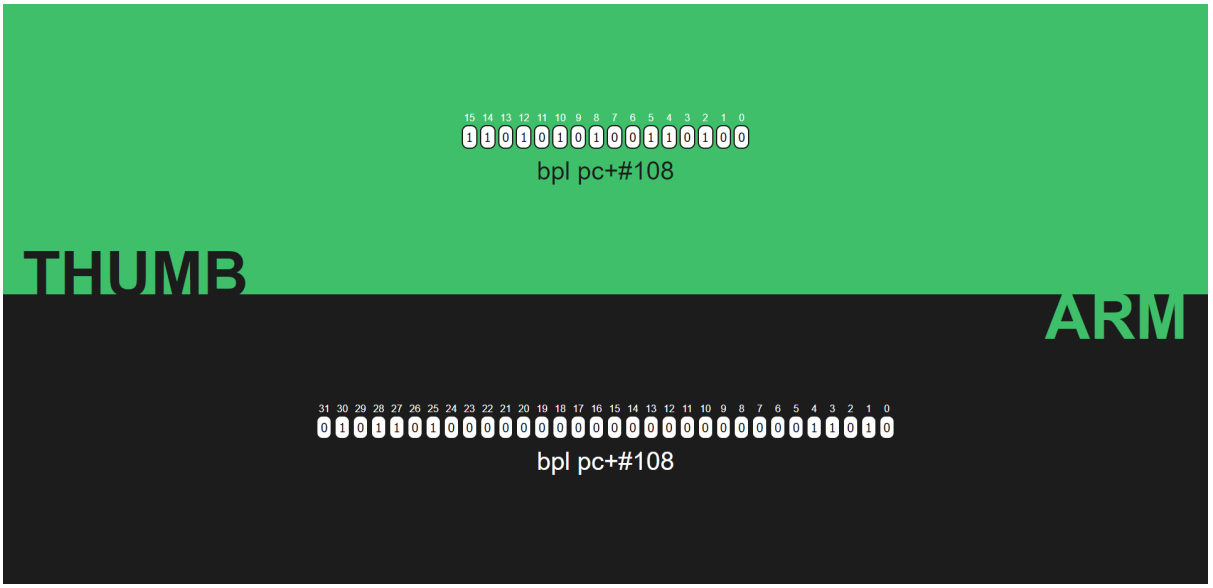
	Alcançável	Não alcançável
	(000_0000_0101)	

3.3 Exemplos

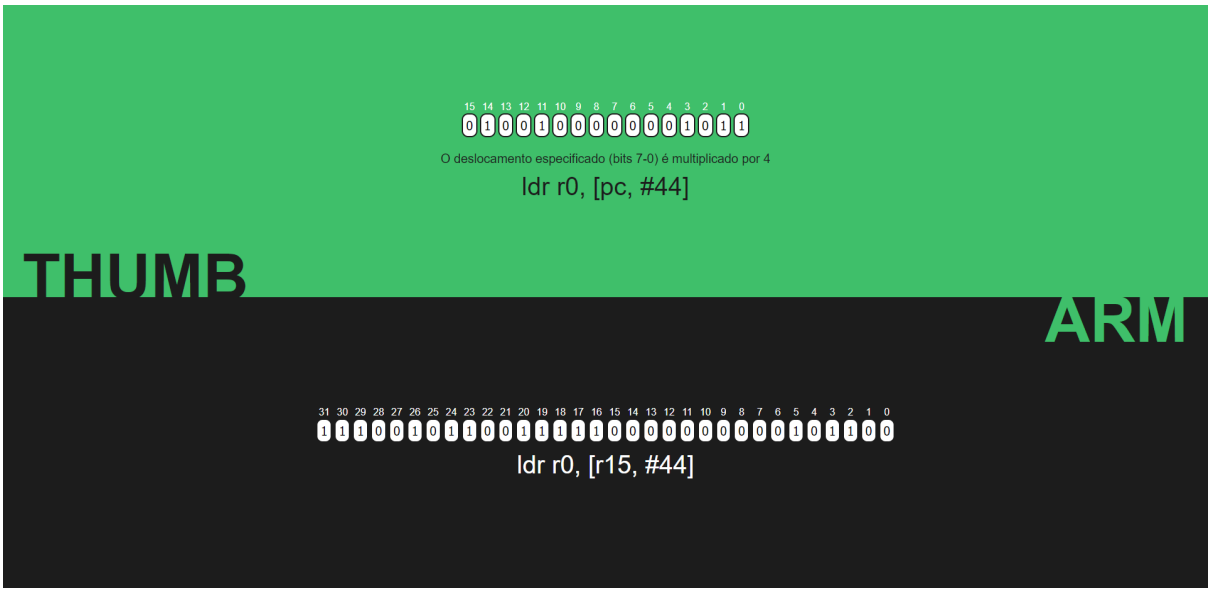
- Operação de deslocamento do conteúdo de um registrador



- Operação de desvio condicional, endereço atingível



- Operação "Load" relativo ao contador de programa



4. Testes

Esta seção é dedicada a uma série de testes para validar o comportamento do tradutor. Na tabela de testes que segue logo abaixo, temos a instrução a ser traduzida e a instrução resultante, ambas na visualização ASCII e hexadecimal – considerou-se a representação binária demasiadamente extensa.

Teste	Thumb ASCII	Thumb Hex	ARM32 ASCII	ARM32 Hex
1	asrs r2, r1, #3	0x10ca	movs r2, r1, asr #3	0xe1b0_21c1
2	subs r2, r1, #5	0x1f4a	subs r2, r1, #0x5	0xe251_2005
3	cmp r5, #0x74	0x2d4a	cmp r5, #0x4a	0xe355_504a
4	muls r2, r1	0x434a	muls r2, r1, r2	0xe012_0291
5	negs r2, r1	0x424a	rsbs r2, r1, #0x0	0xe271_2000
6	add r14, r9	0x44ce	add r14, r14, r9	0xe08e_e009
7	ldr r3, [pc, #136]	0x4b22	ldr r3, [r15, #136]	0xe59f_3088
8	strb r3, [r6, r2]	0x54b3	strb r3, [r6, r2]	0xe7c6_3002
9	ldrh r3, [r6, r2]	0x5ab3	ldrh r3, [r6, r2]	0xe196_30b2

10	ldrh r7, [r6, #20]	0x8ab7	ldrh r3, [r6, #20]	0xe1d6_71b4
11	add r5, SP, #540	0xad87	add r5, r13, #0x21c	0xe28d_5f87
12	add SP, #-0	0xb0ff	sub r13, r13, #0x0	0xe24d_d000
13	PUSH {r0-r5, LR}	0xb53f	stmdb r13!, {r0-r5,r14}	0xe92d_403f
14	ldmia r3!, {r0-r2,r4}	0xcb17	ldmia r3!, {r0-r2,r4}	0xe8b3_0017
15**	bhi pc+#80	0xd826	bhi pc+#80	0x8a00_0013
16	b pc+#1084	0xe21c	b pc+#1084	0xea00_010e
17	swi #155	0xdf9b	swi #155	0xef00_009b

** Os desvios inalcançáveis não foram incluídos na tabela, retornam uma instrução nula 0x0

5. Próximos passos

Relativo a proposta do projeto, destacam-se dois pontos principais de extensão do projeto para melhor usabilidade do projeto:

- **Sugestão de instruções:** para o caso em que não é possível traduzir diretamente as instruções Thumb em ARM32 – isto é, a tradução não envolve apenas uma instrução – convém sugerir instruções substitutas ao usuário que gerem um resultado equivalente em termos de execução.
    - Um exemplo é a tradução de uma instrução de desvio (condicional, ou não) cujo campo de deslocamento (offset) tenha o primeiro bit igual a 1. Uma opção seria o usuário carregar o endereço em um registrador e usar o desvio com troca (branch exchange).
- **Tradução ARM32 para Thumb:** Outra possibilidade de extensão é a tradução de instruções ARM32 para Thumb. Como o conjunto ARM32 é muito mais amplo em termos de quantidade e versatilidade de instruções, uma implementação desse tipo envolveria um estudo cuidadoso dos tipos de instruções que possuem “equivalente” em Thumb. Tal extensão faria uso muito mais intensivo de sugestão de instruções.
- **Integração com o site do professor:** O professor da disciplina já tem um site que fornece o assembly equivalente dado uma instrução em linguagem de máquina. Ele funciona tanto para ARM32 quanto para Thumb. O passo seguinte seria, portanto, fazer um pull request desse projeto para, quando o usuário clicar no botão que chaveia de Thumb para ARM, ser feita a tradução da instrução em linguagem de máquina.

6. Conclusão

A implementação do projeto se deu de maneira muito objetiva e controlada, visto que os resultados esperados estavam pré-definidos pela documentação do conjunto de instruções Thumb. Com o uso do desenvolvimento guiados a testes (TDD), foi possível

reduzir consideravelmente o tempo de implementação, e, assim, aumentar a confiabilidade do resultado.

O projeto foi inspirado na aplicação desenvolvida pelo Professor Bruno Basseto, na qual é possível visualizar, em codificação ASCII, as instruções Thumb e ARM32 configuradas em bits. Espera-se, da mesma maneira, que o projeto B LEARN seja útil às demais gerações de estudantes e entusiastas dos conjuntos de instruções ARM, e que eles possam utilizá-lo – e estendê-lo – devidamente.

7. Referências

[1] DWEDIT. ARM7TDMI Technical Reference Manual. Disponível em: https://www.dwedit.org/files/ARM7TDMI.pdf. Acesso em: 14 ago. 2023.

[2] BRU4BAS. asmclick repository. GitHub. Disponível em: https://github.com/bru4bas/asmclick. Acesso em: 14 ago. 2023.