

Escola Politécnica da Universidade de São Paulo
PCS3732 - Laboratório de Processadores

Lucas Pavan Garieri

11806614

Pedro Henrique Rodrigues de Viveiros

11804035

Pedro Vitor Bacic

11806934

Sophia Lie Asakura

11806656

Documentação do Projeto
Simulador ARM

15/08/2023

SUMÁRIO

1. INTRODUÇÃO E ESTRUTURA DO PROJETO	2
a. Escopo e Implementação	2
b. Organização do Projeto	3
c. Execução do Projeto	3
2. FRONTEND	5
a. Telas de Simulação das Instruções	6
b. Tela do Quiz	9
3. BACKEND	11
a. Modelagem do Sistema	11
b. Decisões de Projeto	12
4. PRÓXIMOS PASSOS	13

1. INTRODUÇÃO E ESTRUTURA DO PROJETO

a. Escopo e Implementação

Dado o contexto atual da tecnologia e a grande proliferação de aparelhos como celulares, computadores e sistemas embarcados, por exemplo, é inegável a importância e a presença da arquitetura ARM na atualidade. ARM, ou "Advanced RISC Machine", é um conjunto de instruções focado em flexibilidade, eficiência energética e em seu design mais compacto. Dessa forma, a proposta desse projeto é desenvolver um simulador de instruções ARM que possibilite o usuário aprender mais sobre algumas categorias de instruções e sobre o funcionamento do ARM como um todo, enquanto pode testar seu conhecimento simulando as instruções aprendidas.

O projeto foi desenvolvido em JavaScript e Python, usando React e Django. A parte visual do projeto e a interface com a qual o usuário interage foi implementada usando React e é o que possibilita que o projeto seja disponibilizado no formato de uma aplicação Web. Já o tratamento dos dados, processamento e simulação das instruções é feito no backend da aplicação, implementado em Django, permitindo que uma lógica que simula as principais funcionalidades do ARM fosse implementada.

Desse modo, foi implementado um conjunto de instruções, que foram selecionadas visando uma apreensão gradual dos conceitos. Inicialmente, foi abordado operações aritméticas como **ADD** (adição), **SUB** (subtração), **RSB** (subtração reversa) e **MUL** (multiplicação). Em seguida, operações lógicas **AND**, **ORR** (ou) e **EOR** (ou exclusivo) foram abordadas para promover a compreensão da lógica de bits. Após isso, foram implementadas instruções de desvio condicional, **B** (desvio), **BL** (desvio com link), **BX** (desvio e troca) e **BLX** (desvio com link e troca), com uma simplificação na qual o label não é utilizado.

Ademais, a movimentação de dados é aborda com as instruções **MOV** (mover) e **MVN** (mover negação). Além disso, **CLZ** (contagem de zeros à esquerda) foi incluída para oferecer uma perspectiva adicional sobre manipulação de dados. A instrução **BIC** (bit clear) foi implementada para ilustrar a clara operação de limpeza de bits específicos. Por fim, a inclusão das condições e flags fornece uma compreensão prática da execução condicional das instruções. Em suma, essa abordagem gradual confere uma base sólida para compreensão das operações e controle dentro do contexto da arquitetura ARM.

b. Organização do Projeto

A divisão de telas do projeto foi feita a partir de cada categoria de instrução e ser simulada e outras funcionalidades da aplicação. Dessa forma, temos uma tela dedicada para a descrição e simulação de cada tipo de instrução citada anteriormente, sendo uma para aritméticas, uma para desvios e uma para os demais detalhes da arquitetura. Além disso, temos uma tela principal, uma tela contendo informações sobre os integrantes do grupo e uma tela de quiz, na qual o usuário pode responder uma série de perguntas sobre a arquitetura ARM e receber o resultado do seu desempenho.

c. Execução do Projeto

Este projeto foi disponibilizado [neste repositório](#) do GitHub e, para executá-lo, é necessário clonar o repositório na sua máquina local. Após clonar o projeto, é preciso instalar o NPM, um gerenciador de pacotes do Node.js, antes de instalar as demais dependências do projeto e executá-lo.

Após instalar o NPM, basta abrir um terminal na pasta raiz do projeto (PCS3732_LabProc) e executar os comandos mostrados abaixo.

- Primeiro entrar na pasta referente à aplicação:

```
cd ./frontend
```

- Após isso, instalar as dependências do projeto e executá-lo localmente:

```
npm install  
npm start
```

A partir desses comandos é possível executar a interface, no entanto, para apresentar a interação com o banco de dados e realizar as operações de fato, é necessário instalar e configurar o backend. Para tal, é necessário executar as seguintes instruções:

```
cd backend  
py -m pip install pipenv  
pipenv install --python 3.10  
pipenv run python manage.py migrate
```

Documentação do Projeto - Simulador ARM

Com o ambiente local configurado, é possível executar o servidor via comando `pipenv run python manage.py runserver`.

Para facilitar a configuração de ambos os ambientes, foi elaborado um script que realiza a configuração de forma automática chamado de `test_setup.bat` com as seguintes dependências:

- Python 3.10
- NVM

Em caso de erros durante a execução do script, basta seguir o passo a passo que consta no README.

2. FRONTEND

Como descrito anteriormente, o frontend da aplicação foi desenvolvido usando JavaScript e React. O projeto conta com seis telas no total, sendo três para a simulação das instruções aritméticas, desvios e demais funcionalidades, uma tela para o quiz e as telas principal e “Quem Somos Nós?”. Uma descrição mais detalhada de cada tela, juntamente com as imagens de cada uma, pode ser visto a seguir:

Figura 1: Tela Principal da Aplicação do Projeto



Figuras 2 e 3: Tela Quem Somos Nós?



Documentação do Projeto - Simulador ARM



Lucas Pavan Garieri

NUSP 11886614. Bom, todo mundo me chama de Luquinhas, ou alguma variação como Lu, e, nesse projeto, fiquei responsável pelo frontend e algumas das telas. Gosto muito de música, tanto ouvir, quanto tocar, e estou tentando aprender teclado atualmente.

Pedro Henrique Viveiros

NUSP 11884835. Oi, tudo bom? Também me chamam de Pedrinho e nesse projeto atuei como desenvolvedor backend. Sou muito fã de Jô e tô tentando retomar a academia pós quadris.



Pedro Vitor Bacic

NUSP 11886934. Todo mundo me chama de Bacic/Basic, mas dizem as fics que meu nome é Pedro. Nesse projeto, me envolvi com o backend. Amo música pop/funk nacional, amo reality shows e Minecraft e Devour é o que mais jogo no momento.

Sophia Lie Asakura

NUSP 11886656. A maioria das pessoas da Poli me chamam de Lie com exceção do Luquinhas. Nesse projeto fiquei responsável pelo frontend. Gosto muito de dançar e queria voltar a praticar no futuro.




a. Telas de Simulação das Instruções

Para a implementação das telas de instrução, foram separadas em três telas principais: instruções aritméticas, instruções de desvio e demais instruções, que englobam instruções lógicas e instruções básicas para o funcionamento do ARM. Cada uma das telas possui os mesmos componentes principais, onde o usuário é capaz de selecionar qual instrução ele quer simular, quais registradores são utilizados e quais os valores armazenados. Além disso, para uma simulação mais completa e para um maior entendimento, o usuário também pode selecionar a utilização ou não de um imediato no lugar do segundo operando e também é feita uma explicação breve sobre a instrução sendo simulada.

Para um melhor entendimento do usuário que está utilizando o projeto com fins de aprender mais sobre as instruções, também foram feitas telas de explicação onde é possível selecionar a instrução e ver uma explicação mais completa sobre ela. As telas de instruções implementadas podem ser vistas nas imagens a seguir:

Figuras 4 e 5: Telas de Instrução Aritmética

 PCS3732

Aritméticas

Desvio

Quiz

Outras Instruções

Quem somos nós?

Instruções Aritméticas

Instruções aritméticas na arquitetura ARM desempenham um papel crucial na realização de operações matemáticas fundamentais, como adição, subtração, multiplicação e acumulação. Essas instruções permitem que um processador ARM execute cálculos numéricos essenciais, tanto para aplicações simples quanto para tarefas complexas. Ao possibilitar a manipulação de dados numéricos em nível de máquina, as instruções aritméticas formam a base para o processamento de informações em diversos domínios, incluindo computação científica, jogos, processamento de sinais e muito mais. Sua importância reside na capacidade de realizar cálculos de forma eficiente e precisa, permitindo que dispositivos ARM executem operações matemáticas vitais para uma ampla gama de aplicações, tornando-os versáteis e essenciais em muitos contextos tecnológicos.

Emulador


A instrução ADD (Addition) na arquitetura ARM é utilizada para realizar operações de adição. Ela permite somar o valor de dois registradores, um registrador e um valor imediato, ou um registrador e um valor de memória. O resultado é armazenado no registrador de destino.

A sintaxe geral da instrução ADD é:
ADD{cond} Rd, Rn, Operando2

{cond}: Condição opcional para executar a instrução. (Não é simulado neste projeto)
Rd: Registrador de destino, onde o resultado será armazenado.
Rn: Registrador que contém o primeiro operando.
Operando2: Valor a ser somado, que pode ser um registrador, um valor imediato ou um valor deslocado.

Exemplos de uso:
ADD R1, R2, R3 ; R1 = R2 + R3
ADD R4, R5, #10 ; R4 = R5 + 10

ADD | ▾

 PCS3732

Aritméticas

Desvio

Quiz

Outras Instruções

Quem somos nós?

Instruções Aritméticas

Imediato ☐

SUB | ▾

R2 | ▾

R3 | ▾

R2 | ▾

1r

Informações sobre a instrução acima

No caso da instrução de subtração (SUB) selecionada, a ALU pega os valores armazenados nos registradores R3 e R2 subtrai os seus valores na mesma ordem (primeiro registrador menos o segundo registrador) e armazena em R2. É possível observar os resultados a partir da tabela de reistradores ao lado ao executar a instrução.

R0	0b binário, 0x hexa
R1	0b binário, 0x hexa
R2	0b binário, 0x hexa
R3	0b binário, 0x hexa
R4	0b binário, 0x hexa
R5	0b binário, 0x hexa
R6	0b binário, 0x hexa
R7	0b binário, 0x hexa
R8	0b binário, 0x hexa
R9	0b binário, 0x hexa
R10	0b binário, 0x hexa
R11	0b binário, 0x hexa
R12	0b binário, 0x hexa
R13	0b binário, 0x hexa
R14	0b binário, 0x hexa
R15	0b binário, 0x hexa

7

Documentação do Projeto - Simulador ARM

Figuras 6 e 7: Telas de Instrução de Desvio

PCS3732

AritméticasDesvioQuizOutras InstruçõesQuem somos nós?

Instruções de Desvio

Instruções de desvio na arquitetura ARM desempenham um papel essencial no controle do fluxo de execução de um programa. Elas permitem que um programa altere sequencialmente a ordem das instruções, desviando para diferentes partes do código com base em condições ou necessidades específicas. As instruções de desvio, como B, BL, BX e BLX, capacitam os programadores a criar estruturas de decisão, loops e chamadas de função, criando fluxos de execução dinâmicos e adaptativos. A importância dessas instruções reside na capacidade de criar lógica complexa, permitindo que um programa tome decisões e execute diferentes caminhos de código, aumentando a versatilidade e a funcionalidade dos sistemas baseados em ARM. Em essência, as instruções de desvio são os pilares do controle de fluxo em linguagem de máquina ARM, possibilitando a criação de programas mais interativos, flexíveis e eficientes.

Emulador

A instrução BL (Branch with Link) na arquitetura ARM é utilizada para realizar desvios condicionais ou incondicionais no fluxo de execução de um programa, assim como a instrução B. No entanto, a instrução BL também armazena o endereço de retorno, permitindo que o programa volte ao local de onde o desvio ocorreu.

A sintaxe geral da instrução BL é:
BL{cond} label

{cond}: Condição opcional para executar o desvio apenas se uma determinada condição for satisfeita. (Não é simulado neste projeto)
label: Rótulo que representa o endereço de memória de destino para o qual o programa será desviado.

Exemplos de uso:
BL _fim
BL #10
BL 0xa
BL 0b1010

BL

PCS3732

AritméticasDesvioQuizOutras InstruçõesQuem somos nós?

Instruções de Desvio

Imediato ☐

BX

R2

Ir

Informações sobre a instrução acima

No caso da instrução acima selecionada de branch and exchange (BX), pegando como exemplo uma sequência de instruções sendo executada, ao encontrar com a instrução branch, o programa será desviado para o local armazenado no imediato, nesse caso sendo R2, porém sem armazenar o local onde foi realizado o desvio, assim como é feito no branch (B). A diferença é que é apenas um registrador no local de desvio. É possível observar os resultados a partir da tabela de registradores ao lado ao executar a instrução.

R0	0b binário, 0x hexa	R8	0b binário, 0x hexa
R1	0b binário, 0x hexa	R9	0b binário, 0x hexa
R2	0b binário, 0x hexa	R10	0b binário, 0x hexa
R3	0b binário, 0x hexa	R11	0b binário, 0x hexa
R4	0b binário, 0x hexa	R12	0b binário, 0x hexa

Figuras 8 e 9: Telas de Demais Instruções

PCS3732

Aritméticas
Desvio
Quiz
Outras Instruções
Quem somos nós?

Outras instruções: lógicas e outros operandos

As instruções abordadas nesta seção da arquitetura ARM são operações fundamentais que envolvem a manipulação direta de dados a nível de bits. Elas permitem a realização de operações lógicas, movimentação de dados, inversão de bits e contagem de zeros, todas operando bit a bit. Essas instruções desempenham um papel crucial na programação de baixo nível, onde o controle granular sobre os bits e registradores é essencial. A importância dessas instruções reside em sua capacidade de efetuar operações complexas e eficientes em nível de bits, fundamentais para várias áreas, como criptografia, otimização de algoritmos, manipulação de dados em hardware e operações de máscara. Elas oferecem um nível avançado de controle sobre os dados e permitem que os programadores realizem tarefas intrincadas, como combinar, inverter ou contar bits de forma precisa e rápida. Essas operações, embora possam parecer primitivas, são a base para a construção de algoritmos sofisticados e eficientes, tornando-se ferramentas essenciais para a implementação de sistemas de software e hardware robustos, rápidos e seguros na arquitetura ARM.

A instrução BIC (Bit Clear) na arquitetura ARM é usada para realizar operações de "bitwise AND" (E lógico) entre os bits de um registrador e o complemento dos bits do segundo operando. Essa instrução permite que bits específicos em um registrador sejam limpos (definidos como 0), enquanto os outros bits permanecem inalterados.

A sintaxe geral da instrução BIC é: BIC{cond} Rd, Rn, Operando2

{cond}: Condição opcional para executar a instrução. (Não é simulado neste projeto)

Rd: Registrador de destino, onde o resultado da operação será armazenado.

Rn: Registrador que contém um dos operandos.

Operando2: Segundo operando, que pode ser um registrador, um valor imediato ou um valor deslocado.

Exemplos de uso:

```

BIC R1, R2, #0x03 ; R1 = R2 AND NOT 0x03
BIC R3, R4, #0b100 ; R3 = R4 AND NOT 0b100
BIC R5, R6, R7, LSR #5 ; R5 = R6 AND NOT (R7 >> 5)
                
```

BIC

▼

PCS3732

Aritméticas
Desvio
Quiz
Outras Instruções
Quem somos nós?

Demais Instruções

Instrução Lógica (4 operandos) ☒ Imediato ☐

MVN

▼

R2

▼

R3

▼

R3

▼

Ir

Informações sobre a instrução acima

No caso da instrução move not (MVN) selecionada acima, são utilizados os valores armazenados nos registradores R2 e R3. Pela operação lógica MVN, o valor do segundo operando é armazenado no primeiro operando após realizar o NOT no valor registrado, armazenando em: R2. É possível observar os resultados a partir da tabela de registradores ao lado ao executar a instrução.

R0	0b binário, 0x hexa	R8	0b binário, 0x hexa
R1	0b binário, 0x hexa	R9	0b binário, 0x hexa
R2	0b binário, 0x hexa	R10	0b binário, 0x hexa
R3	0b binário, 0x hexa	R11	0b binário, 0x hexa
R4	0b binário, 0x hexa	R12	0b binário, 0x hexa
R5	0b binário, 0x hexa	R13	0b binário, 0x hexa
R6	0b binário, 0x hexa	R14	0b binário, 0x hexa
R7	0b binário, 0x hexa	R15	0b binário, 0x hexa

b. Tela do Quiz

Para a implementação do quiz, foi feito um pequeno card no meio da tela que, dinamicamente, carrega uma sequência de perguntas e 4 opções de múltipla escolha para serem escolhidas. Por não necessitar de muito processamento, optou-se por deixar o tratamento das respostas do quiz no próprio frontend da aplicação.

Documentação do Projeto - Simulador ARM

Uma sequência de perguntas, suas opções e a resposta certa estão mapeadas em um arquivo JavaScript dentro do projeto. O componente quiz do projeto, então, faz o controle das perguntas a partir do estado em que o usuário se encontra, renderizando o card com a pergunta na sequência adequada. Além disso, é mantida a quantidade de acertos que o usuário fez e, esta, mostrada ao final do quiz. O quiz pode ser acessado de qualquer tela e a qualquer momento, sendo que, ao final, o usuário é levado de volta para a tela principal da aplicação.

Figuras 10 e 11: Tela de Quiz e Resultado

PCS3732 Aritméticas Desvio Quiz Saiba mais Quem somos nós?

Qual das seguintes opções melhor descreve a sigla 'ARM'?

Advanced Resource Machine

Advanced Reduced Memory

Acorn RISC Machine

Acquired Reduced Microprocessor

1/12 Voltar

Fim do Quiz!

Você acertou 2 perguntas de 12!

Voltar

3. BACKEND

a. Modelagem do Sistema

Toda lógica realizada pela arquitetura ARM está contida no backend desenvolvido no framework Django (Python).

Primeiramente, foi definido o modelo que guiaria as operações, nesse caso, os registradores. Sendo assim, utilizando do princípio de orientação a objetos, foi feita a classe *Register* que contém, além do valor armazenado no registrador, também o “nome” do registrador, nesse caso variando de R0 a R15, além do CPSR. Esse modelo foi utilizado em todas as operações que envolviam registradores de fato.

Com essa base, passou-se para a execução das operações em si. Elas foram implementadas por meio de views do Django, isto é, as funções que permitem a comunicação do sistema com o usuário, recebendo as suas requisições via métodos HTTP. No caso das operações, todas são direcionadas para um mesmo endpoint, cujo corpo da requisição POST pode ser parametrizável como ao lado:

```
{
  "operation": "ADD",
  "registerDestination": {"label": "R0"},
  "firstOperand": {
    "label": "R1",
    "value": "2"
  },
  "secondOperand": {
    "label": "R2",
    "value": "1"
  }
}
```

O campo *operation* contém o mnemônico da instrução a ser executada em modo de string. O *registerDestination* contém o nome dado ao registrador que armazenará o resultado da instrução. Por fim, temos os campos *firstOperand* e *secondOperand* que podem representar tanto registradores quanto imediatos, no caso deste, apenas o campo *value* está presente, já quando o campo representa um registrador, ele apresenta o campo *label* juntamente com o campo *value*.

O backend recebe tais informações e, caso a operação seja bem sucedida, envia o código HTTP 200, indicando que ocorreu tudo como devido. Caso contrário, o código 400 (Bad Request) é enviado, indicando algum erro no que tange à requisição feita pelo usuário. Toda a lógica implementada nas operações está contida no arquivo *utils.py*.

Além desse endpoint, dois outros foram implementados que são importantes para o funcionamento do sistema: o de listagem de registradores e atualização dos registradores.

O de listagem implementa um método do tipo GET para obter o valor de todos os registradores presentes no backend juntamente com seus nomes. Este será utilizado para atualizar a tabela de registradores a fim de fornecer um feedback ao usuário das operações.

Já o de atualização, implementada via método POST, é utilizado para que, após edição do próprio usuário, a tabela do frontend seja enviada para o backend para que, antes de realizar as operações, os registradores estejam devidamente atualizados e siga as especificações do usuário.

Ambos os endpoints levam em conta a estrutura de dados ao lado para o tratamento.

b. Decisões de Projeto

Quanto à escolha do banco de dados para armazenamento de dados, como o projeto foi feito para fins de teste, usou-se o SQLite, uma vez que atende às necessidades de poucos dados e é compatível com o Django por ser seu padrão, facilitando a configuração inicial do projeto.

Quanto à organização e legibilidade do código, foi implementado um linter, isto é, um formatador de código que, além de verificar possíveis erros antes da compilação, também deixa o código uniforme em questões de design, facilitando a leitura do código e, conseqüentemente, a expansão futura do projeto.

Para testagem isolada do backend durante o desenvolvimento do projeto foi utilizado o software Postman, o qual permite que sejam feitas requisições a um link especificado alterando o corpo da requisição e mostrando a resposta no próprio aplicativo. Para conferência, é possível importar as requisições criadas para teste contidas na pasta *collection* do projeto.

```
[
  {
    "label": "R0",
    "value": "0"
  },
  {
    "label": "R1",
    "value": "0"
  },
  {
    "label": "R2",
    "value": "0"
  },
  {
    "label": "R3",
    "value": "0"
  },
  {
    "label": "R4",
    "value": "0"
  },
  {
    "label": "R5",
    "value": "0"
  },
  {
    "label": "R6",
    "value": "0"
  },
  {
    "label": "R7",
    "value": "0"
  },
  {
    "label": "R8",
    "value": "0"
  },
  {
    "label": "R9",
    "value": "0"
  },
  {
    "label": "R10",
    "value": "0"
  },
  {
    "label": "R11",
    "value": "0"
  },
  {
    "label": "R12",
    "value": "0"
  },
  {
    "label": "R13",
    "value": "0"
  },
  {
    "label": "R14",
    "value": "0"
  },
  {
    "label": "R15",
    "value": "0"
  },
  {
    "label": "CPSR",
    "value": "0"
  }
]
```

4. PRÓXIMOS PASSOS

Por fim, foram mapeados alguns pontos de melhoria ou objetivos que não foram cumpridos, por motivos de tempo, e levantados como possíveis próximos passos para desenvolver e melhorar ainda mais o projeto.

Nesse sentido tem-se, primeiramente, a possibilidade de simular mais que uma instrução simultaneamente e a criação de programas mais complexos. Isso implicaria em melhorar a forma como o processamento das instruções é feito, adicionar a simulação de uma memória e demais registradores e unificar as telas de instruções. Isso, no entanto, exigiria a replicação de uma maneira mais fiel e mais complexa da arquitetura ARM no backend da aplicação.

Além disso, tem-se a criação de uma área didática sobre a arquitetura ARM na aplicação e a melhoria do quiz para torná-lo ainda mais dinâmico. Nesse sentido, a aplicação contaria com uma tela dividida em diferentes assuntos sobre o ARM, na qual o usuário poderia ler e aprender um pouco mais sobre a arquitetura. E, por fim, um banco de perguntas maiores para possibilitar que, cada vez que o usuário iniciasse o quiz, ele respondesse questões diferentes e, assim, tornar o quiz menos previsível.

Quanto ao backend, em específico, percebe-se que uma reformulação para um outro framework ou linguagem de programação seria apropriado, pois Python, por ser fracamente tipado, pode apresentar inconveniências para tratamento de dados em binário, especialmente na simulação dessas instruções em baixo nível. Por familiaridade dos integrantes do grupo, o Django foi escolhido, mas com mais tempo para desenvolvimento seria interessante essa transição para obter um código mais organizado.

Ademais, algumas particularidades das instruções implementadas não foram consideradas, como o uso de flags e instruções condicionais. Sendo assim, implementar esse mecanismo é importante para explorar toda a capacidade do processador ARM e seu conjunto de instruções.