Universidade Estadual do Oeste do Paraná - UNIOESTE

Centro de Engenharias e Ciências Exatas - CECE

Disciplina de Organização e Arquitetura de Computadores

Docente: Camile Frazao Bordini

Isabela Pimentel Loebel

Kevin Willian de Oliveira

Gustavo Juliano Borges

Trabalho 2 e 3

Relatório

Foz do Iguaçu

2024

# Introdução

# Problema Proposto:

Implementar um simulador RISC-V Monociclo em uma linguagem de programação de sua escolha. O simulador deve conseguir executar o seguinte conjunto de instruções: *add, sub, and, or, addi, lw, sw, beq* e *bne*.

O programa deve interpretar corretamente as instruções fornecidas nos 3 arquivos de testes.

Observam-se as operações:

* *addi* (formato I: *opcode*=0010011; *funct3*=000)
* *bne* (formato S: *opcode*=1100011; *funct3*=001)

# Desenvolvimento

Para obtenção de uma solução satisfatória, buscou-se entender a lógica de funcionamento do processamento da entrada binária, sua tradução e por fim, o que de fato ela deveria executar. Como todo sistema, a solução proposta primeiramente verifica a validade do valor inserido como entrada, ou seja, o bloco de binários a ser analisado.

Para os passos seguintes, utilizamos como material de apoio, o quadro de constantes das instruções RISC-V disponibilizado nos materiais de aula, podendo ser visualizado na figura 1 abaixo.

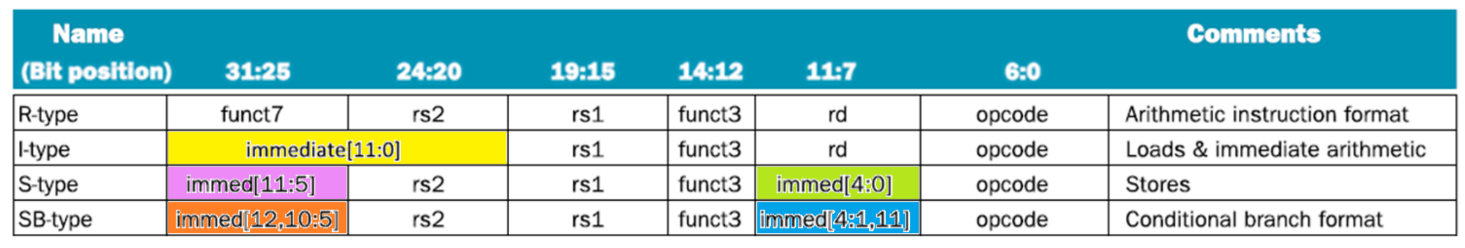


Figura 1 - Quadro de constantes.

Também se mostrou necessário o entendimento da maneira que o gerador de imediato constrói as constantes e estende o último bit, para isso, visualize a figura 2.

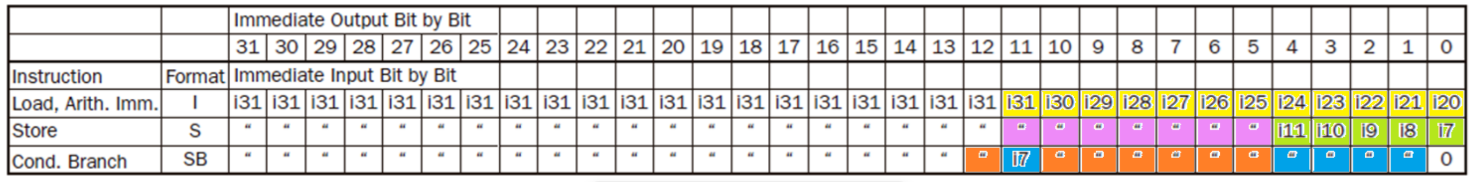


Figura 2 - Gerador de imediato.

Com isso esclarecido, inicia-se o processo de tradução, visto que o sistema deverá entender a entrada, e após isso, realizar sua operação. Usaremos de exemplo um bloco de binários, podendo ser verificado na tabela 1.

|  |  |
| --- | --- |
| **Binário bruto** | 00000000110001011000011100110011 |

Tabela 1 – Bloco binário de exemplo.

A estratégia de tradução é feita de forma que o primeiro artefato a ser analisado é o *opcode*, sendo os 7 últimos números do binário como mostrado na figura 1, o qual indica, qual o tipo da instrução, podendo ser visualizado na tabela 2 os possíveis valores e seu tipo significante.

|  |  |
| --- | --- |
| ***opcode*** | **tipo da instrução** |
| 0110011 | Tipo R |
| 0100011 | Tipo S |
| 1100011 | Tipo B |
| 0010011 | Tipo I |
| 0000011 | Tipo I |

Tabela 2 – *Opcode* e sua instrução correspondente.

Após obtermos o tipo da instrução, é encaminhado para o seu processamento, onde será feito o restante das traduções, visto que, assim como mostrado na figura 1, cada tipo de instrução obtém seus valores de formas diferentes.

Seguindo o exemplo da tabela 1, ao iniciar o tratamento do restante das informações contidas no binário da instrução, é feita a separação do binário, onde cada variável (*rs1, rs2, rd, funct3* e *funct7*) receberão seus valores separando as partes relevantes aos mesmos da instrução de entrada.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Binário bruto** | 0000000011000101100001110 (sem *opcode*) | | | | |
| **Binário particionado** | 0000000 | 01100 | 01011 | 000 | 01110 |
| **Constante tipo R** | *funct7* | *rs2* | *rs1* | *funct3* | *rd* |

Tabela 3 – Bloco binário de exemplo separado e variáveis correspondentes.

Separadas as variáveis, verifica-se o *funct7*, no caso da instrução do tipo R, podendo ser os valores indicados na tabela 4. Nota-se que a segunda opção de *funct7* oferece mais que um tipo de operação resultante, isso deve-se ao fato que a percepção final de qual será a operação é obtida na verificação do *funct3*, como mostrado na tabela 5.

|  |  |
| --- | --- |
| ***funct7*** | **operação** |
| 0100000 | sub |
| 0000000 | add **OU** or **OU** and |

Tabela 4 – Funct7 e operação correspondente.

|  |  |
| --- | --- |
| ***funct3*** | **operação** |
| 000 | add |
| 110 | or |
| 111 | and |

Tabela 5 – Funct3 e operação correspondente.

Com isso traduzido, é feito o processamento de fato da instrução, e por fim, serão exibidos os sinais de controle de cada operação, seguindo a tabela 6.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Sinais de Controle** | | | | | | |
| **ALUOp** | **Branch** | **ALUScr** | **MemToReg** | **ReqWrite** | **MemRead** | **MemWrite** |
| addi | 00 | 0 | 1 | 0 | 1 | 0 | 0 |
| add, sub, and e or | 10 | 0 | 0 | 0 | 1 | 0 | 0 |
| sw | 00 | 0 | 1 | - | 0 | 0 | 1 |
| lw | 00 | 0 | 1 | 1 | 1 | 1 | 0 |
| beq | 01 | 1 | 0 | - | 0 | 0 | 0 |
| bne | 11 | 1 | 0 | 0 | 0 | 0 | 0 |

Tabela 6 – Sinais de controle referente à instrução analisada.

Por fim, é listado os registradores e a pilha da memória também para serem exibidas em tela para visualização passo a passo das instruções.

# Conclusão