TRABALHO DE GRADUAÇÃO

RISC-V SiMPLE

Arthur de Matos Beggs

Brasília, Julho de 2019

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

RISC-V SiMPLE

Arthur de Matos Beggs

Relatório submetido como requisito parcial de obtenção
de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Marcus Vinicius Lamar, CIC/UnB _____
Orientador

Prof. Ricardo Pezzuol Jacobi, CIC/UnB _____
Co-Orientador

Brasília, Julho de 2019

FICHA CATALOGRÁFICA

ARTHUR, DE MATOS BEGGS RISC-V SiMPLE, [Distrito Federal] 2019. $n°$???, ???p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2019). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia. 1. RISC-V 2. ??? I. Mecatrônica/FT/UnB II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

BEGGS, ARTHUR DE MATOS, (2019). RISC-V SiMPLE. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$???, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, ???p.

CESSÃO DE DIREITOS

AUTOR: Arthur de Matos Beggs

TÍTULO DO TRABALHO DE GRADUAÇÃO: RISC-V SiMPLE.

GRAU: Engenheiro ANO: 2019

Arthur de Matos Beggs

SHCGN 703 Bl G Nº 120, Asa Norte

70730-707 Brasília – DF – Brasil.

Dedicatória

Dedico ao pato de borracha especialista em TI que sempre me ajuda a depurar meus códigos.

Arthur de Matos Beggs

Agradecimentos

Agradecimentos!

Arthur de Matos Beggs

RESUMO

Resumo.

Palavras Chave: RISC-V

ABSTRACT

Abstract.

Keywords: RISC-V

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

BSD	Distribuição de Software de Berkeley — *Berkeley Software Distribution*
CSR	Registradores de Controle e Estado — *Control and Status Registers*
FPGA	Arranjo de Portas Programáveis em Campo — *Field Programmable Gate Array*
hart	*hardware thread*
ISA	Arquitetura do Conjunto de Instruções — *Instruction Set Architecture*
MIPS	Microprocessador sem Estágios Intertravados de *Pipeline* — *Microprocessor without Interlocked Pipeline Stages*
OAC	Organização e Arquitetura de Computadores
RISC	Computador com Conjunto de Instruções Reduzido — *Reduced Instruction Set Computer*
SiMPLE	Ambiente de Aprendizado Uniciclo, Multiciclo e *Pipeline* — *Single-cycle Multicycle Pipeline Learning Environment*
RAS	Pilha de Endereços de Retorno — *Return Address Stack*
TTL	Lógica Transistor-Transistor — *Transistor-Transistor Logic*

Capítulo 1

Introdução

1.1 Motivação

O mercado de trabalho está a cada dia mais exigente, sempre buscando profissionais que conheçam as melhores e mais recentes ferramentas disponíveis. Além disso, muitos universitários se sentem desestimulados ao estudarem assuntos desatualizados e com baixa possibilidade de aproveitamento do conteúdo no mercado de trabalho. Isso alimenta o desinteresse pelos temas abordados e, em muitos casos, leva à evasão escolar. Assim, é importante renovar as matérias com novas tecnologias e tendências de mercado sempre que possível, a fim de instigar o interesse dos discentes e formar profissionais mais capacitados e preparados para as demandas da atualidade.

Hoje, a disciplina de Organização e Arquitetura de Computadores da Universidade de Brasília é ministrada utilizando a arquitetura *MIPS32*. Apesar da arquitetura *MIPS32* ainda ter grande força no meio acadêmico (em boa parte devido a sua simplicidade e extensa bibliografia), sua aplicação na indústria tem diminuído consideravelmente na última década.

Embora a curva de aprendizagem de linguagens *Assembly* de alguns processadores *RISC* seja relativamente baixa para quem já conhece o *Assembly MIPS32*, aprender uma arquitetura atual traz o benefício de conhecer o *estado da arte* da organização e arquitetura de computadores.

Para a proposta de modernização da disciplina, foi escolhida a *ISA RISC-V* desenvolvida na Divisão de Ciência da Computação da Universidade da Califórnia, Berkeley como substituta à *ISA MIPS32*.

1.2 Por que *RISC-V*?

A *ISA RISC-V* (lê-se *"risk-five"*) é uma arquitetura *open source* com licença *BSD*, o que permite o seu livre uso para quaisquer fins, sem distinção de se o trabalho possui código-fonte aberto ou proprietário. Tal característica possibilita que grandes fabricantes utilizem a arquitetura para criar seus produtos, mantendo a proteção de propriedade intelectual sobre seus métodos de implementação e quaisquer subconjuntos de instruções não-*standard* que as empresas venham a

produzir, o que estimula investimentos em pesquisa e desenvolvimento.

Empresas como Google, IBM, AMD, Nvidia, Hewlett Packard, Microsoft, Qualcomm e Western Digital são algumas das fundadoras e investidoras da *RISC-V Foundation*, órgão responsável pela governança da arquitetura. Isso demonstra o interesse das gigantes do mercado no sucesso e disseminação da arquitetura.

A licença também permite que qualquer indivíduo produza, distribua e até mesmo comercialize sua própria implementação da arquitetura sem ter que arcar com *royalties*, sendo ideal para pesquisas acadêmicas, *startups* e até mesmo *hobbyistas*.

O conjunto de instruções foi desenvolvido tendo em mente seu uso em diversas escalas: sistemas embarcados, *smartphones*, computadores pessoais, servidores e supercomputadores, o que permitirá maior reuso de *software* e maior integração de *hardware*.

Outro fator que estimula o uso do *RISC-V* é a modernização dos livros didáticos. A nova versão do livro utilizado em OAC, Organização e Projeto de Computadores, de David Patterson e John Hennessy, utiliza a *ISA RISC-V*.

Além disso, com a promessa de se tornar uma das arquiteturas mais utilizadas nos próximos anos, utilizar o *RISC-V* como arquitetura da disciplina de OAC se mostra a escolha ideal no momento.

1.3 O Projeto *RISC-V SiMPLE*

O projeto *RISC-V SiMPLE (Single-cycle Multicycle Pipeline Learning Environment)* consiste no desenvolvimento de um processador com conjunto de instruções *RISC-V*, sintetizável em *FPGA* e com *hardware* descrito em *Verilog*. A microarquitetura implementada nesse trabalho é uniciclo, escalar, em ordem, com um único *hart* e com caminho de dados de 64 bits. Trabalhos futuros poderão utilizar a estrutura altamente configurável e modularizada do projeto para desenvolver as versões em microarquiteturas multiciclo e *pipeline*.

O processador contém o conjunto de instruções I (para operações com inteiros, sendo o único módulo com implementação mandatória pela arquitetura) e as extensões *standard* M (para multiplicação e divisão de inteiros) e F (para ponto flutuante com precisão simples conforme o padrão IEEE 754 com revisão de 2008). O projeto não implementa as extensões D (ponto flutuante de precisão dupla) e A (operações atômicas de sincronização), e com isso o *soft core* desenvolvido não pode ser definido como de propósito geral, G (que deve conter os módulos I, M, A, F e D). Assim, pela nomenclatura da arquitetura, o processador desenvolvido é um *RV64IMF*.

O projeto contempla *traps*, interrupções, exceções, *CSRs*, chamadas de sistema e outras funcionalidades de nível privilegiado da arquitetura.

O *soft core* possui barramento Avalon para se comunicar com os periféricos das plataformas de desenvolvimento. O projeto foi desenvolvido utilizando a placa DE2–115 com *FPGA Altera Cyclone* e permite a fácil adaptação para outras placas da Altera.

Capítulo 2

A *ISA RISC-V*

2.1 Visão Geral da Arquitetura

A *ISA RISC-V* é uma arquitetura modular, sendo o módulo base de operações com inteiros mandatório em qualquer implementação. Os demais módulos são extensões de uso opcional. A arquitetura não suporta *branch delay slots* e aceita instruções de tamanho variável. A codificação das instruções de tamanho variável é mostrada na Figura 2.1. As instruções presentes no módulo base correspondem ao mínimo necessário para emular por *software* as demais extensões (com exceção das operações atômicas).

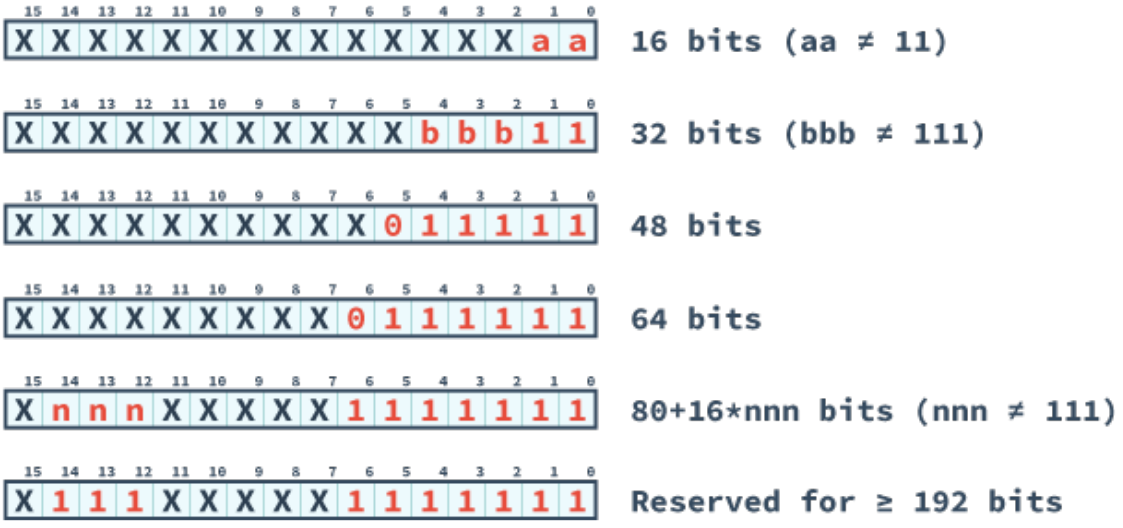

Figura 2.1: Codificação de instruções de tamanho variável da arquitetura *RISC-V*

A nomenclatura do conjunto de instruções implementado segue a seguinte estrutura:

- As letras “RV”;
- A largura dos registradores do módulo Inteiro;
- A letra “I” representando a base Inteira. Caso o subconjunto Embarcado (*Embedded*) seja implementado, substitui-se pela letra “E”;
- Demais letras identificadoras de módulos opcionais.

Assim, uma implementação com registradores de 64 bits somente com o módulo base de Inteiros é denominado “RV64I”.

2.2 Módulo Inteiro

O módulo Inteiro é o módulo base da arquiterura. O *design* de sua especificação visa reduzir o *hardware* necesário para uma implementação mínima, bem como ser um alvo de compilação satisfatório.

Diferente de outras arquiteturas como a *ARM*, as instruções de multiplicação e divisão não fazem parte do conjunto básico uam vez que necessitam de circuito especializado e por isso encarecem o desenvolvimento e produção dos processadores.

Para sistemas embarcados com restrições mais severas de tamanho, custo, potência, etc o módulo base I pode ser substituído por um *subset*, o módulo E. Porém, nenhuma das demais extensões pode ser usada em conjunto com o módulo E.

2.3 Extensões

2.3.1 Extensão M

A extensão M implementa as operações de multiplicação e divisão de números inteiros.

2.3.2 Extensão A

A extensão A implementa instruções de acesso atômico a memória. Instruções atômicas mantém a coerência da memória em sistemas preemptivos e paralelos.

2.3.3 Extensão F

A extensão F implementa as isntruções de ponto flutuante IEEE 754 de precisão simples, bem como o banco de registradores especializado para operações com ponto flutuante.

2.3.4 Extensão D

A extensão D implementa as instruções de ponto flutuante IEEE 754 de precisão dupla. Ela é um incremento à extensão F, sendo esta de implementação obrigatória para se poder implementar a extensão D.

2.3.5 Outras Extensões

Outras extensões são previstas na especificação da arquitetura, e.g. a extensão C para instruções comprimidas (16 bits).

A arquitetura prevê a expansão de extensões, com alguns *opcodes* sendo reservados para essa finalidade. Desse modo, instruções proprietárias e/ou customisadas podem ser adicionadas.

2.4 Arquitetura Privilegiada

Para a *ISA RISC-V*, existem quatro níveis de privilégio de acesso, sendo eles o de usuário (módulo I e extensões), de máquina (*syscalls*) de supervisor (sistema operacional) e hipervisor (virtualização).

2.5 Formatos de Instruções

As instruções da arquitetura podem ser separadas em subgrupos de acordo com os operadores necessários para o processador interpretá-la. A Figura 2.2 apresenta os formatos das instruções do módulo I da *ISA RISC-V*, e, para efeitos de comparação, a Figura 2.3 mostra os formatos de instruções equivalentes na arquitetura MIPS32.

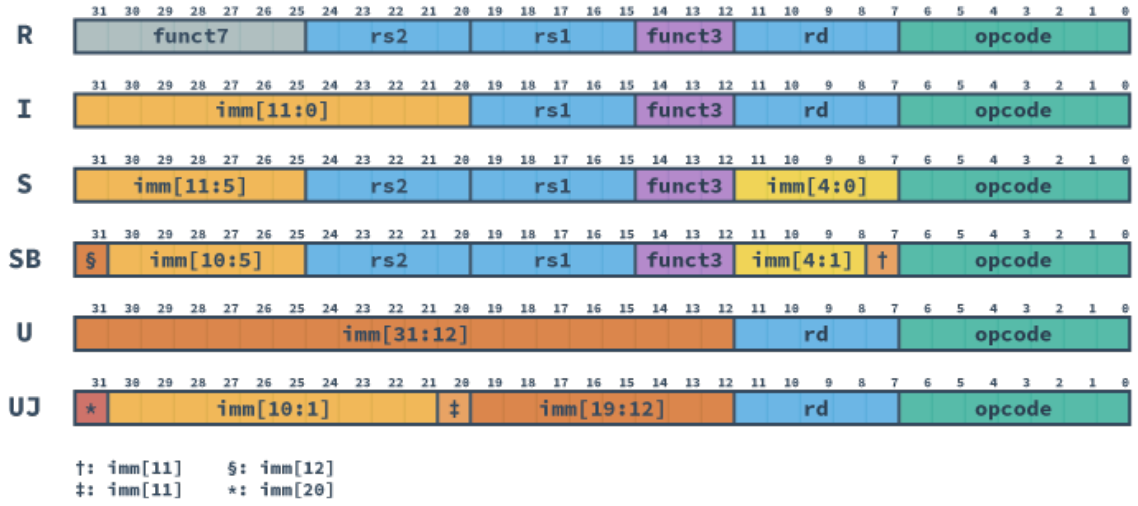

Figura 2.2: Formatos de Instruções da*ISA RISC-V*

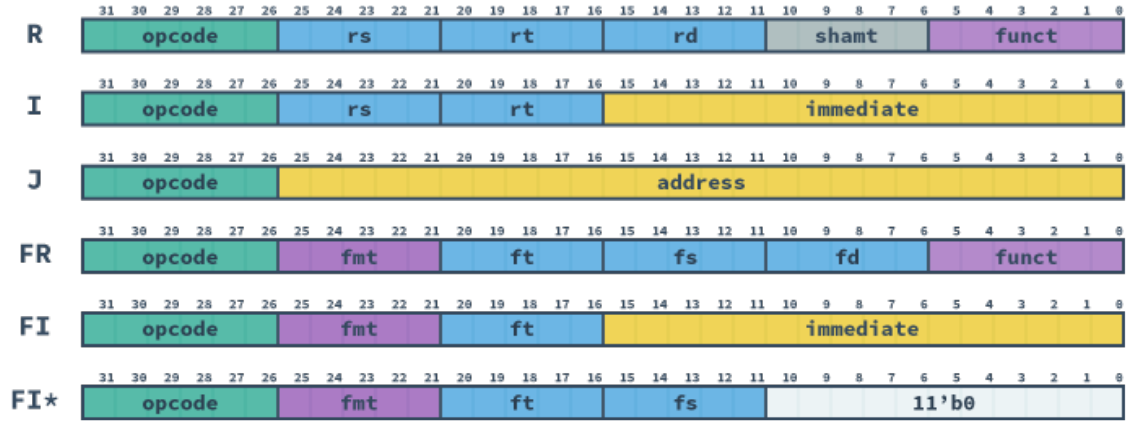

Figura 2.3: Formatos de Instruções da *ISA MIPS32*

2.6 Formatos de Imediatos

Os imediatos são operandos descritos na própria instrução em vez de estar contido em um registrador. Como os operandos necessitam ter a mesma largura que o banco de registradores, algumas regras são utilizadas para gerar os operandos imediatos. As figuras a seguir mostram a formação de cada tipo de imediato dos formatos da Figura 2.2.

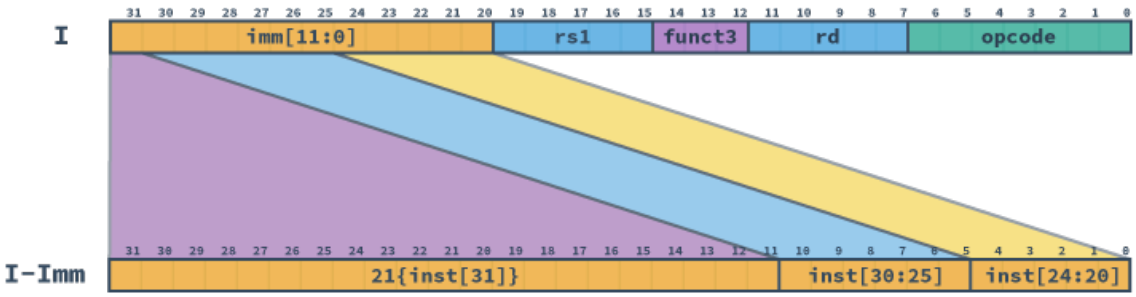

Figura 2.4: Formação do Imediato de tipo I

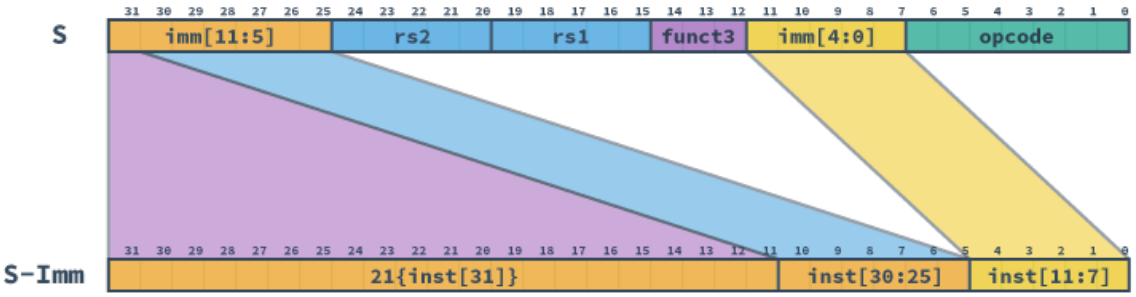

Figura 2.5: Formação do Imediato de tipo S

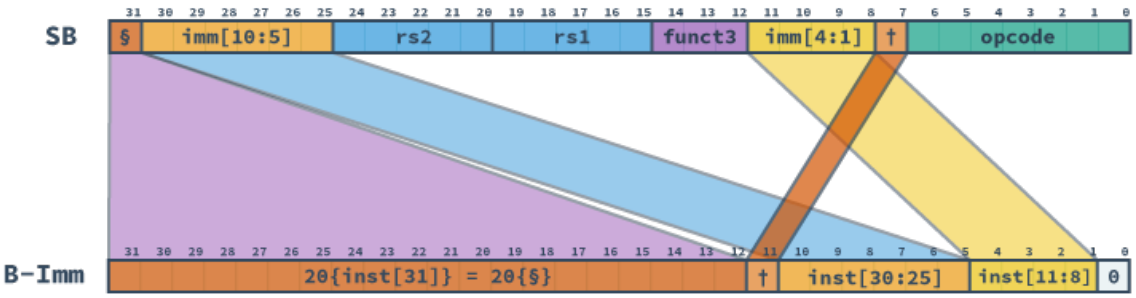

Figura 2.6: Formação do Imediato de tipo B

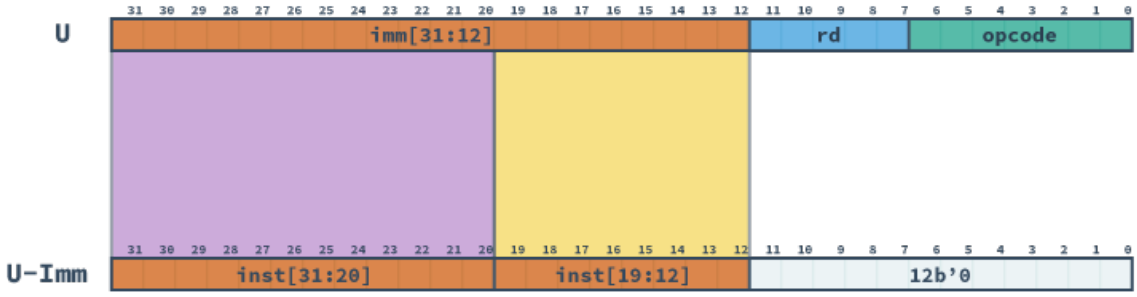

Figura 2.7: Formação do Imediato de tipo U

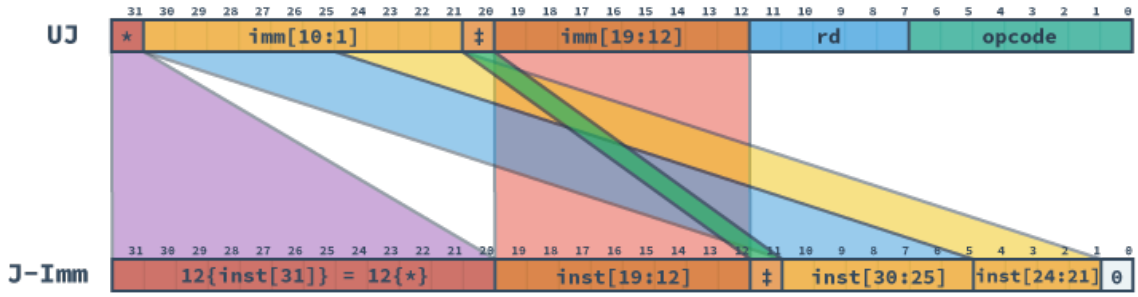

Figura 2.8: Formação do Imediato de tipo J

Para efeitos comparativos, a Figura 2.9 mostra a formação de imediatos na arquitetura MIPS32.

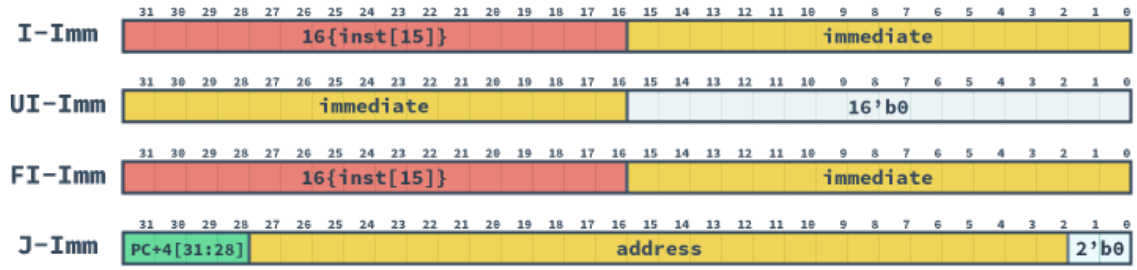

Figura 2.9: Formatos de Imediato da *ISA MIPS32*

Capítulo 3

Implementação

3.1 Caminho de Dados

O caminho de dados projetado para a implementação da microarquitetura uniciclo é apresentado na Figura 3.1.

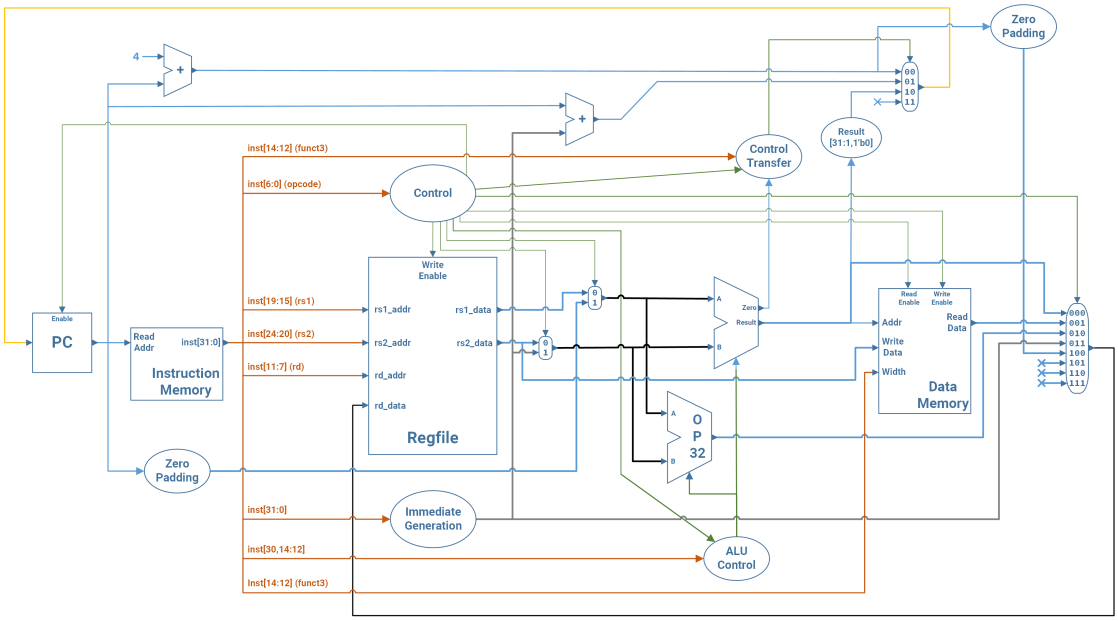

Figura 3.1: Caminho de Dados implementado para o módulo I

O *datapath* possui um banco de 32 registradores de uso geral de 64 bits cada. A memória possui arquitetura Harvard, sendo a memória de instruções (*text*) *read-only* e a memória de dados (*data*) *read-write*. São implementadas 49 instruções, sendo elas:

- LUI: Load Upper Intermediate;

- AUIPC: Add Upper Intermediate to Program Counter;
- JAL: Jump And Link;
- JALR: Jump And Link Register;
- BEQ: Branch if EQual;
- BNE: Branch if Not Equal;
- BLT: Branch if Less Than;
- BGE: Branch if Greater or Equal;
- BLTU: Branch if Less Than Unsigned;
- BGEU: Branch if Greater or Equal Unsigned;
- LB: Load Byte;
- LH: Load Halfword;
- LW: Load Word;
- LBU: Load Byte Unsigned;
- LHU: Load Halfword Unsigned;
- SB: Store Byte;
- SH: Store Halfword;
- SW: Store Word;
- ADDI: ADD Immediate;
- SLTI: Set on Less Than;
- SLTIU: Set on Less Than Unsigned;
- XORI: XOR Immediate;
- ORI: OR Immediate;
- ANDI: AND Immediate;
- SLLI: Shift Left Logical Immedate;
- SRLI: Shift Right Logical Immediate;
- SRAI: Shift Right Arithmetic Immediate;
- ADD: ADD;
- SUB: SUB;

- SLL: Shift Left Logical;
- SLT: Set on Less Than;
- SLTU: Set on Less Than Unsigned;
- XOR: XOR;
- SRL: Shift Right Logical;
- SRA: Shift Right Arithmetic;
- OR: OR;
- AND: AND;
- LWU: Load Word Unsigned;
- LD: Load Double;
- SD: Store Double;
- ADDIW: ADD Immediate Word-size;
- SLLIW: Shift Left Logical Immedate Word-size;
- SRLIW: Shift Right Logical Immediate Word-size;
- SRAIW: Shift Right Arithmetic Immediate Word-size;
- ADDW: ADD Word-size;
- SUBW: SUB Word-size;
- SLLW: Shift Left Logical Word-size;
- SRLW: Shift Right Logical Word-size;
- SRAW: Shift Right Arithmetic Word-size;

Para que o processador seja completamente compatível com a especificação da *ISA*, falta implementar tratamentos de exceções, interrupções e *traps*, Registradores *CSR*, instruções de chamada ao ambiente (ECALL/EBREAK), instruções de *fencing* de memória, suporte ao acesso desalinhado à memória de dados e pilha de endereço de retorno (RAS).

3.2 *Hardware Description Language*

Linguagens de Descrição de Hardware, ou *HDL's* são linguagens de programação que permitem a descrição em alto nível de um circuito lógico.

Diferente de diagramas de blocos onde se descreve o circuito a nível de portas lógicas, em uma *HDL* o comportamento do circuito é descrito por funções, e um sintetizador de *hardware* (programa similar a um compilador) transforma as funções em circuitos.

O uso de *HDL's* possui muitas vantagens em relação aos diagramas de blocos. Por se tratar de uma linguagem com maior nível de abstração, o entendimento das lógicas de funcionamento se um circuito são mais fáceis de se entender, alem de permitir o uso de sistemas de versionamento de código e.g. git para manter um histórico de todas as alterações feitas.

O código a seguir foi retirado do módulo de transferência de controle do processador e exemplifica a estrutura de um programa em Verilog, a linguagem de descrição de *hardware* utilizada no projeto:

```
1  module control_transfer_singlecycle (
2      input  branch_en,
3      input  jal_en,
4      input  jalr_en,
5      input  result_eq_zero,
6      input  [2:0] inst_funct3,
7
8      output reg [1:0] pc_sel
9  );
10
11 always @ ( * ) begin
12     if (branch_en) begin
13         case (inst_funct3)
14             `BRANCH_EQ:
15                 pc_sel = result_eq_zero ? 2'b01 : 2'b00;
16
17             `BRANCH_NE:
18                 pc_sel = result_eq_zero ? 2'b00 : 2'b01;
19
20             `BRANCH_LT:
21                 pc_sel = result_eq_zero ? 2'b00 : 2'b01;
22
23             `BRANCH_GE:
24                 pc_sel = result_eq_zero ? 2'b01 : 2'b00;
25
26             `BRANCH_LTU:
27                 pc_sel = result_eq_zero ? 2'b00 : 2'b01;
28
29             `BRANCH_GEU:
```

```
30                  pc_sel = result_eq_zero ? 2'b01 : 2'b00;
31
32              default:
33                  pc_sel = 2'b00;
34          endcase
35      end
36
37      else if (jal_en) begin
38          pc_sel = 2'b01;
39      end
40
41      else if (jalr_en) begin
42          pc_sel = 2'b10;
43      end
44
45      else begin
46          pc_sel = 2'b00;
47      end
48  end
49
50  endmodule
```

3.3 *Field Programmable Gate Array*

As *FPGA's* são circuitos integrados que possuem uma matriz de blocos lógicos configuráveis. Um diagrama de blocos ou uma *HDL*, após passar pelo processo de síntese (transformação em uma linguagem intermediária que representa o circuito a nível *TTL*) passa pelos processos de mapeamento e alocação (*mapping* e *fitting*) que conectam os elementos da matriz. Assim, o código ou diagrama são "traduzidos" em um circuito.

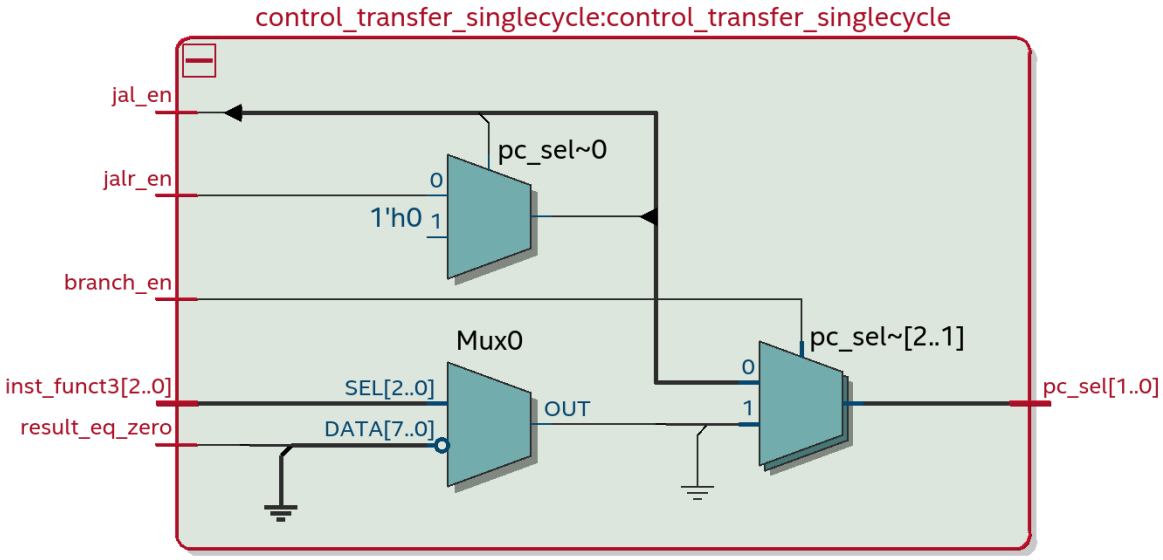

Figura 3.2: Módulo de Transferência de Controle após ser sintetizado

Kits de desenvolvimento possuem, além da *FPGA*, diversos periféricos conectados a ela, permitindo interfacear o circuito sinteitzado com saídas de vídeo, portas USB, memórias FLASH, entre outros componentes. A Figura 3.3 apresenta a placa de desenvolvimento DE2-115 produzida pela empresa *terasIC*.

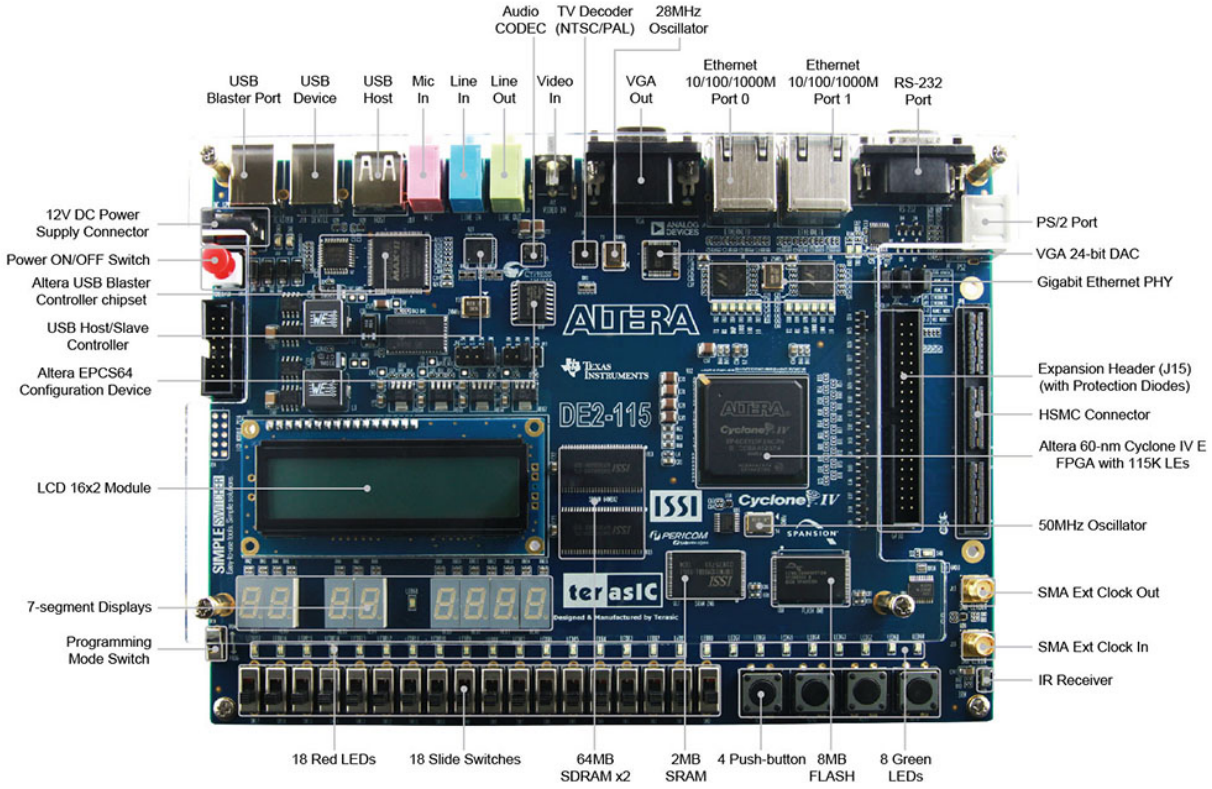

Figura 3.3: Placa de desenvolvimento DE2-115 da terasIC

Capítulo 4

Resultados

Capítulo 5

Conclusões

5.1 Perspectivas Futuras

ANEXOS

I. DESCRIÇÃO DO CONTEÚDO DO CD

Descrever CD.

II. PROGRAMAS UTILIZADOS

Quais programas foram utilizados?