

TRABALHO DE GRADUAÇÃO

RISC-V SiMPLE:

Projeto e desenvolvimento de processadores RISC-V
com a ISA RV32IMF usando as microarquiteturas
Uniciclo, Multiciclo e Pipeline em FPGA

Arthur de Matos Beggs

Brasília, Maio de 2021



**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: Projeto e desenvolvimento de processadores RISC-V com a ISA RV32IMF usando as microarquiteturas Uniciclo, Multiciclo e Pipeline em FPGA

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 _____
Examinador Interno

Prof. Marcelo Grandi Mandelli,CIC/UnB _____
Examinador Interno

Brasília, Maio de 2021

FICHA CATALOGRÁFICA

ARTHUR, DE MATOS BEGGS

RISC-V SiMPLE,

[Distrito Federal] 2021.

???, ???p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. RISC-V

2. Verilog

3. FPGA

I. Mecatrônica/FT/UnB

II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

BEGGS, ARTHUR DE MATOS, (2021). RISC-V SiMPLE. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº???, 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: 2021

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Trabalho de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desse Trabalho de Graduação pode ser reproduzida sem autorização por escrito do autor.

Arthur de Matos Beggs

SHCGN 703 Bl G N° 120, Asa Norte

70730-707 Brasília – DF – Brasil.

Dedicatória

Arthur de Matos Beggs

Agradecimentos

Arthur de Matos Beggs

RESUMO

Desenvolvimento e documentação de uma plataforma de ensino de arquitetura de computadores em *Verilog* sintetizável em *FPGA*, com foco em um processador com arquitetura do conjunto de instruções *RISC-V* implementado em três microarquiteturas para ser utilizado como recurso de laboratório na disciplina de Organização e Arquitetura de Computadores da Universidade de Brasília. A plataforma funciona nas *FPGAs terasIC DE1-SoC* disponíveis no laboratório da Universidade, possui periféricos de depuração como *display* dos registradores do processador na saída de vídeo, além de outros periféricos como *drivers* de áudio e vídeo para uma experiência mais completa de desenvolvimento, e permite que o processador seja substituído por implementações de diversas arquiteturas de *32 bits* com certa facilidade.

Palavras Chave: RISC-V, Verilog, FPGA

ABSTRACT

Keywords: RISC-V, Verilog, FPGA

SUMÁRIO

1	Introdução.....	1
1.1	MOTIVAÇÃO	1
1.2	POR QUE RISC-V?.....	1
1.3	O PROJETO RISC-V SiMPLE	2
2	Revisão Teórica.....	3
2.1	ARQUITETURA DE COMPUTADORES	3
2.1.1	ARQUITETURA MIPS.....	5
2.1.2	ARQUITETURA ARM	5
2.1.3	ARQUITETURA X86.....	5
2.1.4	ARQUITETURA AMD64	5
2.1.5	ARQUITETURA RISC-V	5
2.1.5.1	MÓDULO INTEIRO	6
2.1.5.2	EXTENSÕES.....	6
2.1.5.3	ARQUITETURA PRIVILEGIADA	6
2.1.5.4	FORMATOS DE INSTRUÇÕES.....	7
2.1.5.5	FORMATOS DE IMEDIATOS	8
2.2	MICROARQUITETURAS	10
2.2.1	UNICICLO	10
2.2.2	MULTICICLO.....	10
2.2.3	PIPELINE	10
2.3	REPRESENTAÇÃO DE HARDWARE	10
2.3.1	VHDL	10
2.3.2	VERILOG.....	10
2.4	SÍNTESE LÓGICA	10
2.4.1	ANÁLISE E SÍNTESE	10
2.4.2	FITTING	10
2.4.3	TIMING ANALYZER.....	10
2.5	FIELD PROGRAMMABLE GATE ARRAYS.....	10
2.5.1	ARQUITETURA GENERALIZADA DE UMA FPGA	11
2.5.2	ARQUITETURA DA FPGA CYCLONE V SoC	12
2.5.2.1	ADAPTATIVE LOGIC MODULES	12
2.5.2.2	EMBEDDED MEMORY BLOCKS	13

2.6	ESTADO DA ARTE DOS PROCESSADORES RISC-V	13
3	Sistema Proposto	15
3.1	IMPLEMENTAÇÃO DOS SOFT-CORES	18
3.1.1	MICROARQUITETURA UNICICLO	18
3.1.2	MICROARQUITETURA MULTICICLO	20
3.1.3	MICROARQUITETURA <i>PIPELINE</i> DE 5 ESTÁGIOS	22
3.2	CHAMADAS DE SISTEMA	24
3.3	INTERFACE DE VÍDEO E DEPURAÇÃO	27
3.4	CONFIGURAÇÃO E SÍNTESE DO PROCESSADOR PELO QUARTUS	29
3.5	SIMULAÇÃO DO PROCESSADOR PELO QUARTUS E MODELSIM	29
3.6	SCRIPT <i>MAKE.SH</i>	31
3.7	USO DA FPGA DE1-SoC	31
4	Resultados	34
5	Conclusões	35
5.1	PERSPECTIVAS FUTURAS	35
REFERÊNCIAS BIBLIOGRÁFICAS		36
Anexos		39
I Descrição do conteúdo do CD		40
II Programas utilizados		41

LISTA DE FIGURAS

2.1	Abstração da arquitetura de um computador	4
2.2	Codificação de instruções de tamanho variável da arquitetura <i>RISC-V</i>	5
2.3	Formatos de Instruções da <i>ISA RISC-V</i>	7
2.4	Formatos de Instruções da <i>ISA MIPS32</i>	7
2.5	Formação do Imediato de tipo I	8
2.6	Formação do Imediato de tipo S	8
2.7	Formação do Imediato de tipo B	8
2.8	Formação do Imediato de tipo U	9
2.9	Formação do Imediato de tipo J	9
2.10	Formatos de Imediato da <i>ISA MIPS32</i>	9
2.11	Abstração da arquitetura de uma <i>FPGA</i>	11
2.12	Funcionamento da chave de interconexão	12
2.13	Arquitetura da <i>FPGA</i> Intel Cyclone V SoC.....	12
2.14	Diagrama de blocos de um ALM	13
3.1	Diagrama de blocos do sistema.	18
3.2	Diagrama da implementação das <i>ISAs</i> RV32I e RV32IM na microarquitetura uniciclo.	19
3.3	Diagrama da implementação da <i>ISA</i> RV32IMF na microarquitetura uniciclo.	20
3.4	Diagrama da implementação das <i>ISAs</i> RV32I e RV32IM na microarquitetura multiciclo.	21
3.5	Diagrama da implementação da <i>ISA</i> RV32IMF na microarquitetura multiciclo.....	22
3.6	Diagrama da implementação das <i>ISAs</i> RV32I e RV32IM na microarquitetura <i>pipeline</i> de 5 estágios.....	23
3.7	Diagrama da implementação da <i>ISA</i> RV32IMF na microarquitetura <i>pipeline</i> de 5 estágios.	24
3.8	Exibição do <i>frame</i> de vídeo da <i>FPGA</i>	28
3.9	<i>Menu OSD</i> exibindo os valores dos registradores do processador.	28
3.10	<i>Intel Quartus Lite v18.1</i> com a janela de configurações do projeto.	29
3.11	Janela de configuração da simulação no <i>Quartus</i>	30
3.12	O <i>script</i> <i>NativeLink</i> invoca o <i>ModelSim</i> passando o <i>script</i> <i>.do</i> com as informações de como simular o sistema.....	31
3.13	Placa de desenvolvimento <i>terasIC DE1-SoC</i>	32

LISTA DE TABELAS

3.1	Tabela de <i>syscalls</i> implementadas.	25
3.1	Tabela de <i>syscalls</i> implementadas.	26
3.1	Tabela de <i>syscalls</i> implementadas.	27

LISTA DE SÍMBOLOS

Siglas

ASIC	Circuito Integrado de Aplicação Específica — <i>Application Specific Integrated Circuit</i>
AWS	<i>Amazon Web Services</i>
BSD	Distribuição de Software de Berkeley — <i>Berkeley Software Distribution</i>
CISC	Computador com Conjunto de Instruções Complexo — <i>Complex Instruction Set Computer</i>
CSR	Registradores de Controle e Estado — <i>Control and Status Registers</i>
DSP	Processamento Digital de Sinais — <i>Digital Signal Processing</i>
FPGA	Arranjo de Portas Programáveis em Campo — <i>Field Programmable Gate Array</i>
hart	<i>hardware thread</i>
ISA	Arquitetura do Conjunto de Instruções — <i>Instruction Set Architecture</i>
MIPS	Microprocessador sem Estágios Intertravados de <i>Pipeline</i> — <i>Microprocessor without Interlocked Pipeline Stages</i>
OAC	Organização e Arquitetura de Computadores
PC	Contador de Programa — <i>Program Counter</i>
PLL	Malha de Captura de Fase — <i>Phase-Locked Loop</i>
RAS	Pilha de Endereços de Retorno — <i>Return Address Stack</i>
RISC	Computador com Conjunto de Instruções Reduzido — <i>Reduced Instruction Set Computer</i>
SBC	Computadores em Placa Única — <i>Single Board Computers</i>
SDK	Conjunto de Programas de Desenvolvimento — <i>Software Development Kit</i>
SiMPLE	Ambiente de Aprendizado Uniciclo, Multiciclo e <i>Pipeline</i> — <i>Single-cycle Multicycle Pipeline Learning Environment</i>
SoC	Sistema em um Chip — <i>System on Chip</i>
TSMC	<i>Taiwan Semiconductor Manufacturing Company</i>

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.

Até recentemente, a disciplina de Organização e Arquitetura de Computadores da Universidade de Brasília era ministrada em todas as turmas 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.

Hoje, a disciplina também é ministrada na arquitetura *ARM*, bem como na *ISA RISC-V*, desenvolvida na Divisão de Ciência da Computação da Universidade da Califórnia - Berkeley, e será o objeto de estudo desse trabalho.

1.2 Por que RISC-V?

A *ISA RISC-V* (lê-se “*risk-five*”) é uma arquitetura *open source* [1] 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, Nvidia, Samsung, 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 aprimoramento e documentação do processador com conjunto de instruções *RISC-V*, sintetizável em *FPGA* e com *hardware* descrito em *Verilog* utilizado como material de laboratório de uma das turmas da disciplina de OAC. O objetivo é ter uma plataforma de testes e simulação bem documentada e com o mínimo de *bugs* para servir de referência na disciplina. O projeto implementa três microarquiteturas que podem ser escolhidas a tempo de síntese: uniciclo, multiciclo e *pipeline*, todas as três com um *hart* e caminho de dados de 32 bits.

Os processadores 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, os processadores desenvolvidos são do tipo *RV32IMF*.

O projeto também contempla *traps*, interrupções, exceções, *CSRs*, chamadas de sistema e outras funcionalidades de nível privilegiado da arquitetura [2].

Capítulo 2

Revisão Teórica

2.1 Arquitetura de Computadores

Para nos comunicarmos, necessitamos de uma linguagem, e no caso dos brasileiros, essa linguagem é o português. Como toda linguagem, o português possui sua gramática e dicionário que lhe dá estrutura e sentido. Línguas humanas como o português, inglês e espanhol são chamadas de linguagens naturais, e evoluíram naturalmente a partir do uso e repetição. [3]

Por causa da excelente capacidade de interpretação e adaptação da mente humana, somos capazes de criar e entender novos dialetos que não seguem as regras formais das linguagens naturais que conhecemos. Porém, fora da comunicação casual é importante e às vezes obrigatório que nos expressemos sem ambiguidade. Línguas artificiais como a notação matemática e linguagens de programação possuem semântica e sintaxe mais rígidas para garantir que a mensagem transmitida seja interpretada da maneira correta. Sem essa rigidez, os computadores de hoje não seriam capazes de entender nossos comandos.

Para a comunicação com o processador de um computador, utilizamos mensagens chamadas de instruções, e o conjunto dessas instruções é chamado de Arquitetura do Conjunto de Instruções (*ISA*). Um processador só é capaz de entender as mensagens que obedecem as regras semânticas e sintáticas de sua *ISA*, e qualquer instrução que fuja das suas regras causará um erro de execução ou realizará uma tarefa diferente da pretendida. A linguagem de máquina é considerada de baixo nível pois apresenta pouca ou nenhuma abstração em relação à arquitetura.

As instruções são passadas para o processador na forma de código de máquina, sequências de dígitos binários que correspondem aos níveis lógicos do circuito. Para melhorar o entendimento do código e facilitar o desenvolvimento, uma outra representação é utilizada, o *assembly*. Um código *assembly* é transformado em código de máquina por um programa montador, (*assembler*), e o processo inverso é realizado por um *disassembler*. As linguagens *assembly*, dependendo do *assembler* utilizado, permitem o uso de macros de substituição e pseudo instruções (determinadas instruções que não existem na *ISA* que são expandidas em instruções válidas pelo montador) e são totalmente dependentes da arquitetura do processador, o que normalmente impede que o mesmo código seja executado em arquiteturas diferentes.

A Figura 2.1 é uma representação simplificada de um processador. A unidade de controle lê uma instrução da memória e a decodifica; o circuito de lógica combinacional lê os dados dos registradores, entrada e memória conforme necessário, executa a instrução decodificada e escreve no banco de registradores, na memória de dados ou na saída se for preciso; a unidade de controle lê uma nova instrução e o ciclo se repete até o fim do programa. A posição de memória da instrução que está sendo executada fica armazenada em um registrador especial chamado de Contador de Programa (*PC*). Algumas instruções modificam o *PC* condicionalmente ou diretamente, criando a estrutura para saltos, laços e chamada/retorno de funções.

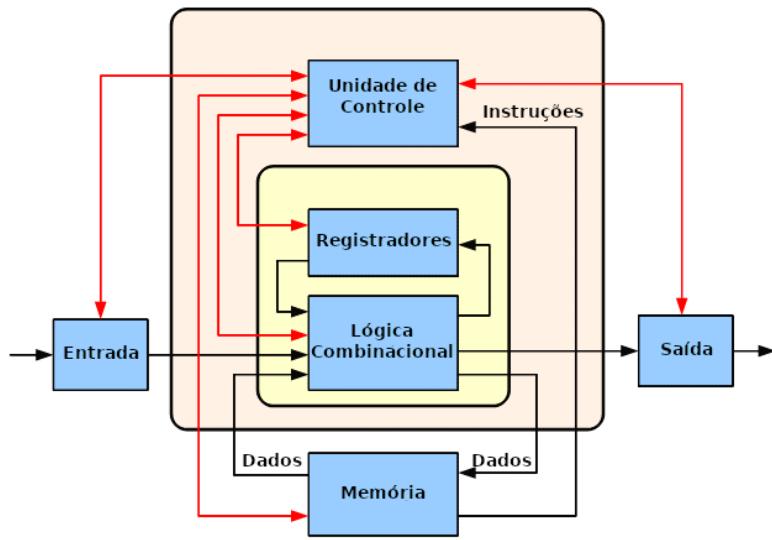


Figura 2.1: Abstração da arquitetura de um computador. Fonte: [4]

Historicamente, as arquiteturas são divididas em *ISAs RISC* e *CISC*. Na atualidade, a diferença entre elas é que as *ISAs RISC* acessam a memória por instruções de *load/store*, enquanto as *CISC* podem acessar a memória diretamente em uma instrução de operação lógica ou aritmética.

Algumas arquiteturas *RISC* notáveis são a *RISC-V*, objeto de estudo desse trabalho, a *ARM* e a *MIPS*. Quanto às *CISC*, a *x86* e sua extensão de 64 bits, a *AMD64*, são as mais conhecidas.

2.1.1 Arquitetura MIPS

2.1.2 Arquitetura ARM

2.1.3 Arquitetura x86

2.1.4 Arquitetura AMD64

2.1.5 Arquitetura RISC-V

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 2.2. As instruções presentes no módulo base correspondem ao mínimo necessário para emular por *software* as demais extensões (com exceção das operações atômicas).

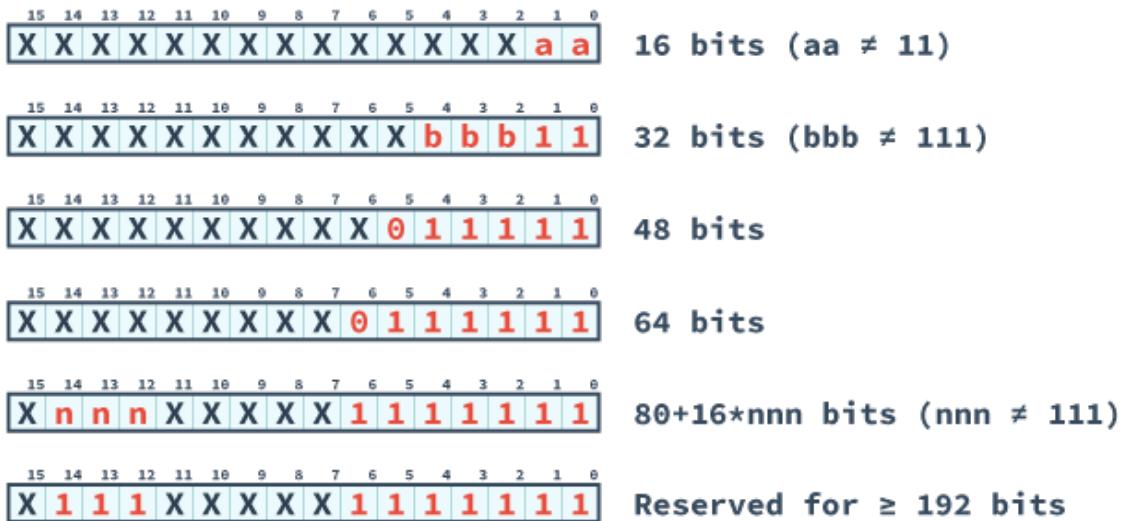


Figura 2.2: Codificação de instruções de tamanho variável da arquitetura *RISC-V*

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 32 bits somente com o módulo base de Inteiros é denominado “RV32I”.

2.1.5.1 Módulo Inteiro

O módulo Inteiro é o módulo base da arquitetura. O *design* de sua especificação visa reduzir o *hardware* necessário 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 uma vez que necessitam de circuito especializado e por isso encaram 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.

2.1.5.2 Extensões

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

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

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

2.1.5.2.4 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.

2.1.5.2.5 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 customizadas podem ser adicionadas.

2.1.5.3 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).

2.1.5.4 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 2.3 apresenta os formatos das instruções do módulo I da *ISA RISC-V*, e, para efeitos de comparação, a Figura 2.4 mostra os formatos de instruções equivalentes na arquitetura MIPS32.

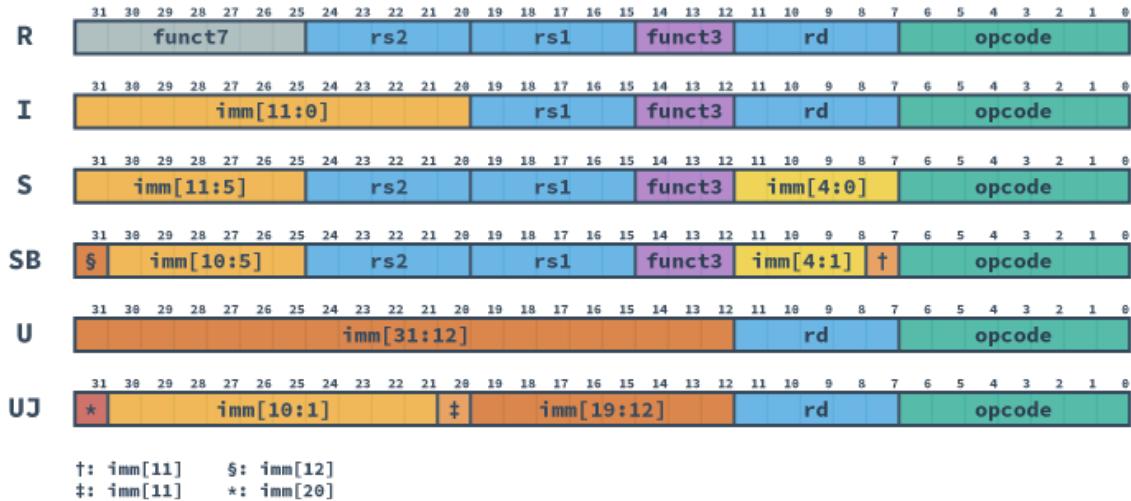


Figura 2.3: Formatos de Instruções da *ISA RISC-V*

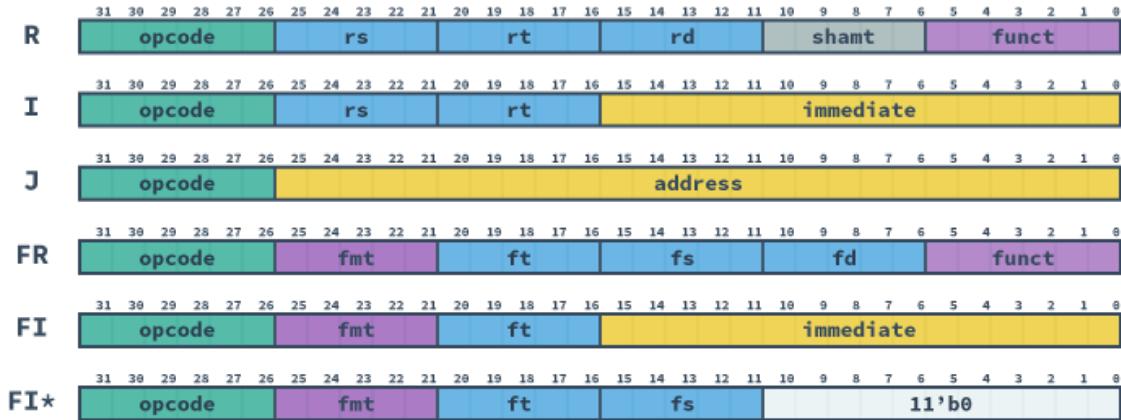


Figura 2.4: Formatos de Instruções da *ISA MIPS32*

2.1.5.5 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 2.3.

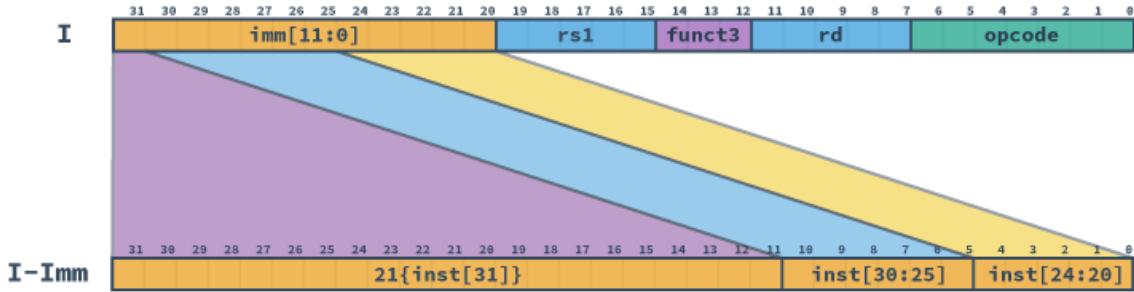


Figura 2.5: Formação do Imediato de tipo I

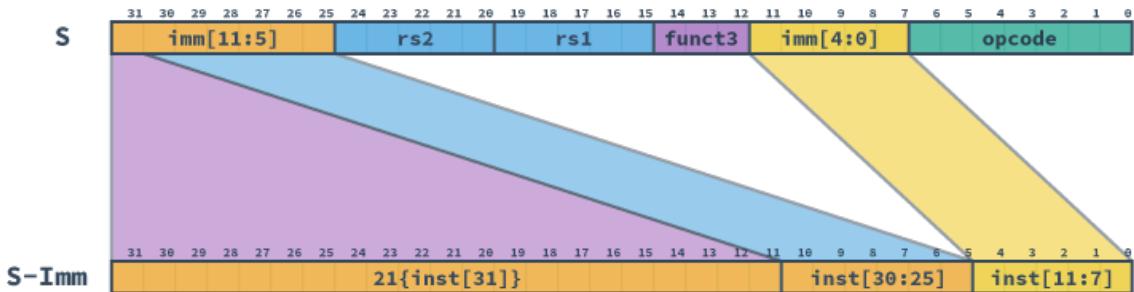


Figura 2.6: Formação do Imediato de tipo S

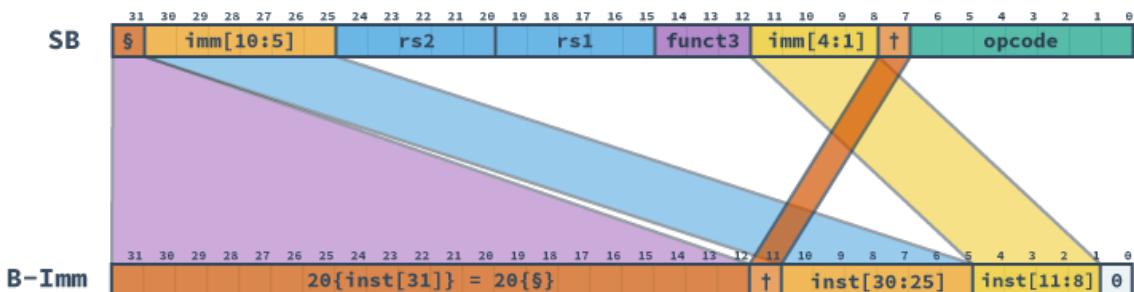


Figura 2.7: Formação do Imediato de tipo B

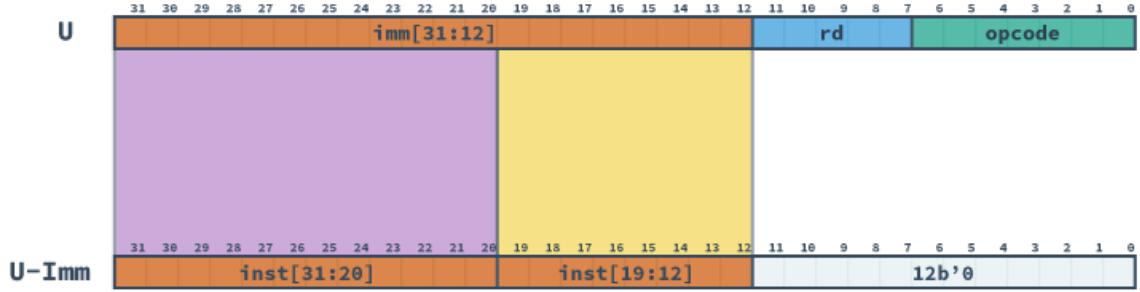


Figura 2.8: Formação do Imediato de tipo U

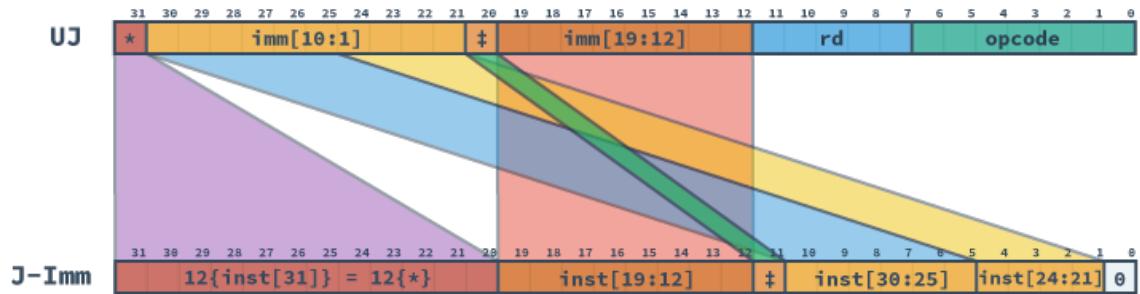


Figura 2.9: Formação do Imediato de tipo J

Para efeitos comparativos, a Figura 2.10 mostra a formação de imediatos na arquitetura MIPS32.



Figura 2.10: Formatos de Imediato da *ISA MIPS32*

2.2 Microarquiteturas

2.2.1 Uniciclo

2.2.2 Multiciclo

2.2.3 Pipeline

2.3 Representação de Hardware

2.3.1 VHDL

2.3.2 Verilog

2.4 Síntese Lógica

2.4.1 Análise e Síntese

2.4.2 Fitting

2.4.3 Timing Analyzer

2.5 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) são circuitos integrados que permitem o desenvolvimento de circuitos lógicos reconfiguráveis. Por serem reprogramáveis, as *FPGAs* geram uma grande economia em tempo de desenvolvimento e em custos como os de prototipagem, validação e manufatura do projeto em relação aos circuitos de aplicações específicas, os *ASICs*, e aos projetos *full-custom*. As *FPGAs* podem ser tanto o passo intermediário no projeto de um *ASIC* ou *full-custom* quanto o meio final do projeto quando a reconfigurabilidade e os preços muito mais acessíveis forem fatores importantes.

Cada fabricante de *FPGAs* possui seus *softwares* de desenvolvimento, ou *SDKs*. A indústria de *hardware* é extremamente protecionista com sua propriedade intelectual, sendo a maioria dessas ferramentas de código proprietário. Para a Intel Altera®, essa plataforma é o Quartus Prime®.

FPGAs mais modernas possuem, além do arranjo de portas lógicas, blocos de memória, *PLLs*, *DSPs* e *SoCs*. Os blocos de memória internos funcionam como a memória *cache* de um microprocessador, armazenando os dados próximo ao seu local de processamento para diminuir a latência. Os *PLLs* permitem criar sinais de *clock* com diversas frequências a partir de um relógio de referência, e podem ser reconfigurados a tempo de execução. *DSPs* são responsáveis pelo processamento de sinais analógicos discretizados, e podem ser utilizados como multiplicadores de baixa latência. Já os *SoCs* são microprocessadores como os ARM® presentes em celulares, e são capazes de

executar sistemas operacionais como o Linux.

Além de disponíveis na forma de *chips* para a integração com placas de circuito impresso customizadas, as *FPGAs* possuem *kits* de desenvolvimento com diversos periféricos para auxiliar no processo de criação de soluções. Esses *kits* são a principal ferramenta de aprendizagem no universo dos circuitos reconfiguráveis. No Laboratório de Informática da UnB, as placas *terasic DE1-SoC* com a *FPGA Intel® Cyclone V SoC* estão disponíveis para os alunos de OAC desenvolverem seus projetos.

2.5.1 Arquitetura Generalizada de uma FPGA

De forma genérica, uma *FPGA* possui blocos lógicos, chaves de interconexão, blocos de conexão direta e portas de entrada e saída, conforme apresentado na Figura 2.11.

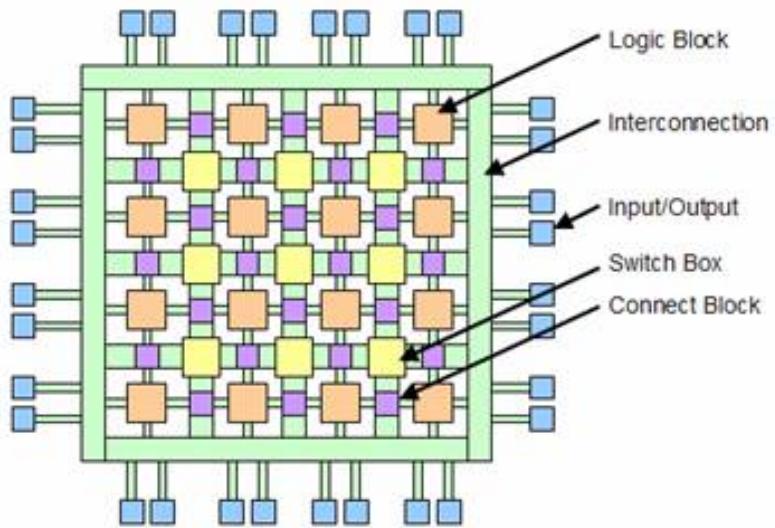


Figura 2.11: Abstração da arquitetura de uma FPGA. Fonte: [5]

Os blocos lógicos possuem *lookup tables*, registradores, somadores e multiplexadores. É neles que a lógica reconfigurável é implementada.

Já as chaves de interconexão são responsáveis por conectar os diversos blocos da *FPGA*. A Figura 2.12 exemplifica como é feito o roteamento da malha de interconexão. Os blocos de conexão direta são um tipo especial de chave de interconexão, e sua função é ligar blocos lógicos adjacentes.

Por fim, as portas de entrada e saída conectam a *FPGA* ao “mundo externo” e.g. *drivers* de áudio e vídeo.

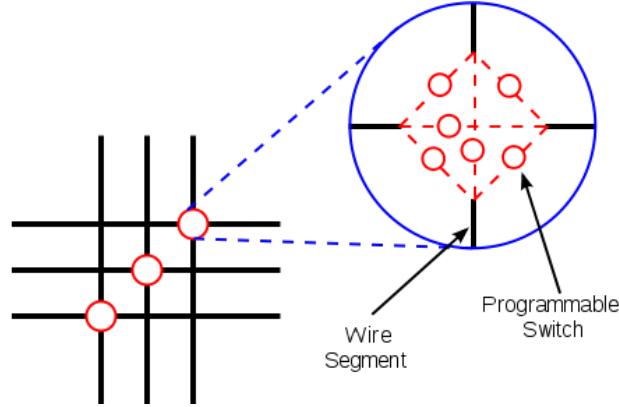


Figura 2.12: Funcionamento da chave de interconexão. Fonte: [6]

2.5.2 Arquitetura da FPGA Cyclone V SoC

A Figura 2.13 apresenta a arquitetura da *FPGA Cyclone V SoC*. O *chip* possui um processador *ARM* integrado, blocos de memória embutidos, *DSPs* para acelerar operações como multiplicação de números ou processamento de sinais genéricos, diversos pinos para integrar o *chip* a um projeto de circuito mais complexo, *PLLs* para gerar diversos sinais de *clock*, entre outras funcionalidades.

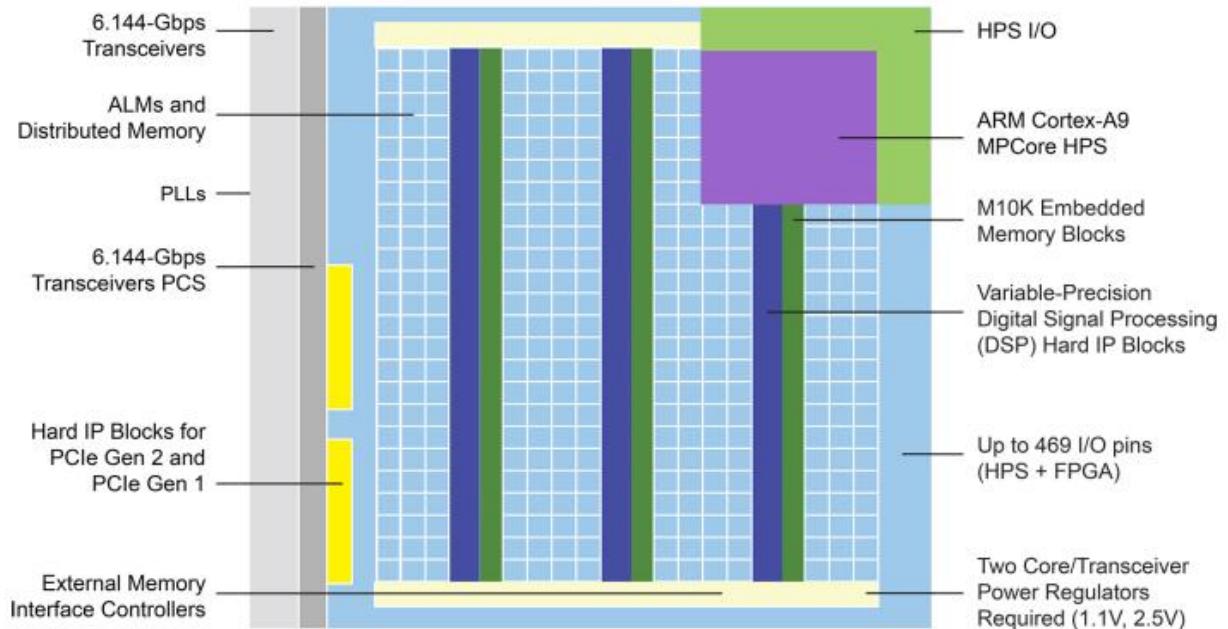


Figura 2.13: Arquitetura da *FPGA Altera Cyclone V SoC*. Fonte: [7]

2.5.2.1 Adaptative Logic Modules

Os blocos lógicos, como mostrados na abstração da Figura 2.11 são implementados na *FPGA Cyclone V SoC* como *Adaptative Logic Modules*, conforme a Figura 2.14. Como os *ALMs* são

blocos genéricos, há um *trade-off* entre configurabilidade e performance.

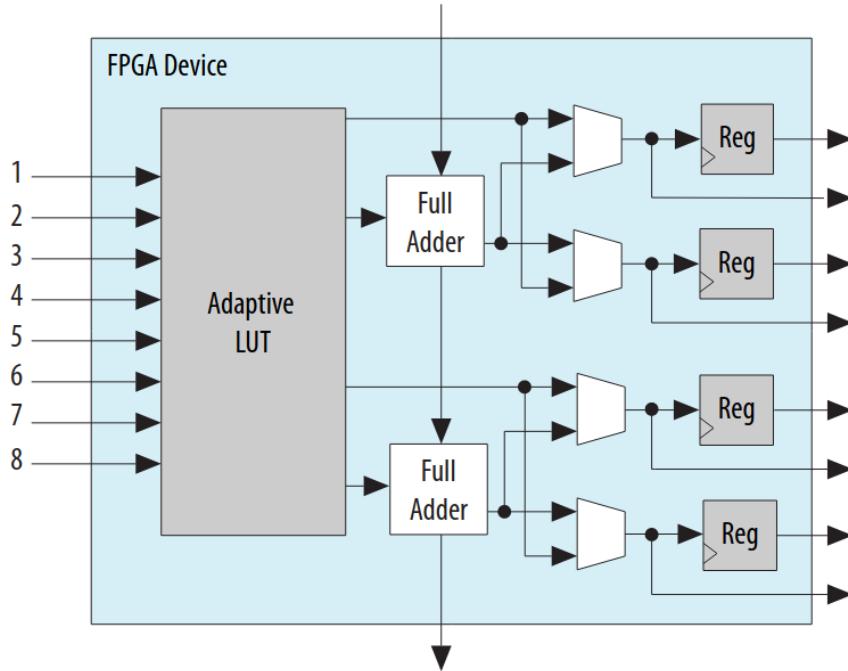


Figura 2.14: Diagrama de blocos de um ALM. Fonte: [8]

2.5.2.2 Embedded Memory Blocks

Como mostrado na Figura 2.13, existem blocos de memória embutidos na *FPGA*. Esses blocos são o equivalente a uma memória *cache L1*, sendo a camada de memória mais próxima dos registradores. Para utilizá-los no *design* do circuito, blocos do *IP* de memória são configurados e instanciados pelo programa de desenvolvimento para gerar um módulo integrável no resto do *design*.

As placas de desenvolvimento podem possuir outros tipos de memória, como as *SRAM* e *DRAM*. Apesar de possuírem capacidade de armazenamento bem maiores que os blocos embutidos, seus módulos controladores são mais complexos e apresentam latência maior para leitura e escrita de dados. Para seu uso eficiente, é necessário implementar camadas de *caching* para que as operações de *input* e *output* (*IO*) não se tornem um gargalo que comprometa o resto do *design*.

2.6 Estado da Arte dos processadores RISC-V

Por alguns anos, processadores com arquitetura *RISC-V* só podiam ser utilizados por meio de emuladores como o *qemu* [9] ou em *FPGAs*, o objeto desse trabalho. Algumas fabricantes já divulgaram planos para começar a usar microcontroladores com a arquitetura em seus produtos, como é o caso dos controladores de discos rígidos e *SSDs* da *Western Digital* [10] e da *Nvidia* como o substituto dos controladores *Falcon* de suas placas de vídeo [11]. Porém, ainda não se sabe se

as empresas já utilizam os controladores em seus *hardwares*, se a adoção ainda está em fase de projeto ou se a ideia foi abandonada.

Porém, começam a surgir no mercado microcontroladores e *Single Board Computers (SBCs)* com preços acessíveis. Placas como a linha *Sipeed* da *Seed* se equiparam aos *MCUs ESP32* [12], e outras como a *SiFive HiFive1* se assemelham aos *arduinoss* [13]. Também é possível utilizar processadores de alto desempenho como o *BOOM* em instâncias *EC2 F1* da *AWS* [14].

Há uma expectativa de *SBCs* mais robustos, capazes de rodar um sistema operacional de uso geral, como um *Raspberry Pi*. Existem alguns pré-lançamentos de placas para atender essa demanda, como a *SiFive HiFive Unmatched* [15] e a *BeagleV* [16].

A empresa *SiFive*, liderada pelos criadores da arquitetura, produzirá em parceria com a *TSMC* (*Taiwan Semiconductor Manufacturing Company*) o primeiro processador *RISC-V* de 32 bits em tecnologia de 5nm [17]. A *TSMC* é a *foundry* líder em manufatura de circuitos integrados no mundo.

Atualmente, compilar códigos em *C/C++* para *targets RISC-V* não envolve mais a instalação de *toolchains* complicadas e frágeis. Tanto o *gcc* [18] quanto o *clang* [19] já oferecem suporte para o *RISC-V*, eliminando assim uma barreira para a adoção da arquitetura.

Uma outra característica essencial para o uso do *RISC-V* em sistemas de uso geral é a existência de sistemas operacionais que funcionem na plataforma. Desde a versão 4.15, o *kernel* do *linux* oferece suporte para a arquitetura [20]. *Distros* como *Fedoras* [21], e *Alpine* [22] já possuem suporte experimental. A chinesa *Alibaba* fez o *port* do *OS Android* para um de seus *SoCs RISC-V* [23]. Alguns ecossistemas mais robustos possuem *ports* completos, como é o caso do *Haiku-OS* [24] e do *microkernel seL4* [25], possibilitando o uso em ambientes industriais e áreas que exigem maior robustez do sistema operacional.

Uma das surpresas na adoção da arquitetura *RISC-V* nos seus *designs* veio da *MIPS Technologies*, detentora das patentes das arquiteturas *MIPS*. Em 2013, a empresa foi adquirida pela *Imagination Technologies* [26], e lançou alguns *development kits* voltados a visão computacional e microcontroladores, mas não conseguiu dar tração aos projetos. Em 2017 a companhia foi novamente vendida para a *Tailwood Venture Capital* [27], que tentou capitalizar em cima dos *royalties* da arquitetura. Porém, em 2018 a companhia foi vendida novamente para a *Wave Computing* [28], companhia voltada para aplicações de inteligência artificial. Em 2020, a *Wave Computing* declara falência [29], demitindo todos os seus funcionários. Em março desse ano, a empresa conseguiu se recuperar da falência, mudou o nome da companhia para *MIPS* e anunciou que seus novos *designs* serão baseados na arquitetura *RISC-V* [30]. Atualmente, a empresa *MIPS* integra a *RISC-V Foundation* como Membro Estratégico.

Com o recente sucesso dos processadores *ARM M1* lançados pela *Apple*, e com os processadores *ARM Graviton* disponíveis no serviço de servidores em nuvem da *Amazon*, é uma possibilidade forte que o desenvolvimento de plataformas *RISC-V* para uso geral desacelere.

Capítulo 3

Sistema Proposto

O sistema proposto consiste em um *soft-core* da *ISA RISC-V* de 32 bits com as extensões **I**, **M** e **F**, podendo ser sintetizado nas versões *RV32I*, *RV32IM* ou *RV32IMF*. A extensão *Zicsr* com os Registradores de Controle e Estado (*CSR*) é parcialmente implementada em todas as três configurações.

Cada uma das combinações da *ISA* pode ser realizada em três microarquiteturas diferentes: uniciclo, multiciclo ou *pipeline* de cinco estágios. Assim, o processador pode ser sintetizado em nove combinações diferentes.

O projeto utiliza a placa de desenvolvimento *terasIC DE1-SoC* contendo diversos periféricos e um *SoC Intel Altera Cyclone-V*. A maioria dos periféricos presentes na plataforma tem controladores implementados com Entradas e Saídas Mapeadas em Memória (*MMIO*) para que o *soft-core* possa utilizá-los. A síntese dos controladores dos periféricos, como a saída de vídeo, entrada de teclado e barramento *RS-232*, é opcional.

O projeto é organizado seguindo o seguinte arranjo de pastas:

core	(arquivos que implementam o soft-core)
clock	(arquivos de interface e controle de sinais de clock do processador)
memory	(arquivos de interface/controle de memória)
misc	(módulos como somador e multiplexador de largura definidas por parâmetros)
peripherals	(interfaces e controladores para os periféricos da FPGA)
risc_v	(projeto do processador RISC-V)
CPU.v	(arquivo top-level do processador)
Control_*.v	(módulos de controle de cada microarquitetura)
Datapath_*.v	(módulos do caminho de dados de cada parch)
...	(demais módulos do processador)
config.v	(arquivo de configuração de versão do

```

processador a implementar, seus
periféricos e endereçamento de memória
das interfaces MMIO)
└── default_data.mif      (arquivo de inicialização de memória de
                           dados usado na síntese do projeto)
└── default_framebuffer.mif (arquivo de inicialização de memória de
                           vídeo usada na síntese do projeto)
└── default_text.mif       (arquivo de inicialização de memória de
                           texto usado na síntese do projeto)
└── fpga_top.sdc          (restrições desejadas de temporização do
                           sistema sintetizado)
└── fpga_top.v             (interface verilog entre o soft-core e a
                           placa de desenvolvimento)
├── doc                   (documentação e guias do projeto)
└── project               (arquivos de projeto do Quartus)
    ├── de1_soc
    │   ├── db                (arquivos de saída intermediários do
                               Quartus; pasta ignorada pelo git)
    │   ├── incremental_db    (arquivos de saída intermediários do
                               Quartus; pasta ignorada pelo git)
    │   ├── output_files       (arquivos de saída do Quartus; os logs de
                               síntese gerados pelo script "make.sh"
                               ficam aqui, bem como o .sof da última
                               síntese completa; ignorada pelo git)
    │   ├── fpga_top.qpf       (arquivo de projeto do Quartus indicando
                               a versão do projeto)
    │   ├── fpga_top.qsf       (arquivo de projeto do Quartus contendo
                               as configurações do projeto)
    │   └── ...
    └── ...
├── system                 (códigos em assembly RISC-V implementando
                           as chamadas de sistema e macros)
└── test
    ├── assembly_testbench  (códigos em assembly RISC-V para testar o
                           funcionamento correto das instruções
                           do processador)
    ├── gtkwave              (formas de onda predefinidas para visualizar
                           os arquivos .vcd gerados pelo ModelSim
                           usando o GTKwave)
    └── mif_library          (testbenches assembly compilados para o
                           formato .mif para gravação na memória
                           da FPGA)

```

```

    └── simulation           (arquivos de saída da simulação pelo
                                ModelSim; pasta ignorada pelo git)
    └── simulation_scripts  (scripts .do para que o ModelSim simule o
                                sistema corretamente)
    └── sof_library          (arquivos .sof das versões do processador
                                prontos para gravação na FPGA)
    └── verilog_testbench    (testbench usado para simular as entradas
                                da FPGA, inicializá-la e definir o
                                tempo de simulação)

    └── tools
        └── bitmap_converter   (conversor de imagens para uso na FPGA)
        └── rars                (montador e simulador de assembly RISC-V)

    └── vendor
        └── ...
            (licenças dos softwares utilizados)

    └── LICENSE              (licença do sistema implementado)

    └── make.sh               (script para síntese e simulação de todas
                                as variantes do processador)

    └── README.md             (README sobre o que é o projeto e como
                                utilizá-lo)

```

O trabalho também é organizado de forma a facilitar a migração para placas de desenvolvimento diferentes da *DE1-SoC* ou trocar o *soft-core* desenvolvido por outra implementação, independente da sua *ISA*. O *soft-core* implementado se encontra no caminho `core/risc_v`. Assim, os demais módulos presentes na pasta `core` não dependem da arquitetura do processador, exceto a tela de *debug* presente na interface de vídeo. No entanto, a tela de *debug* foi projetada de modo a ser relativamente fácil customizá-la para uso em outra arquitetura.

O arquivo `core/config.v` possui todas as opções de configuração, definição de parâmetros e endereçamento de memória dos módulos, facilitando escolher as extensões, microarquitetura e periféricos sintetizados.



Figura 3.1: Diagrama de blocos do sistema.

3.1 Implementação dos *soft-cores*

Todos os *soft-cores* implementados possuem execução em ordem, sem *branch prediction*, sem *caching* de memória e sem *Return Address Stack*. O processador é escalar e possui um único *hart*. Como a implementação atual só utiliza blocos de memória presentes no chip da FPGA, sem utilizar as memórias *SRAM* e *DRAM* externas presentes na placa de desenvolvimento, e também não faz uso de memória secundária, as operações de *load* e *store* transferem dados diretamente entre os registradores e os blocos de memória *M10K*.

3.1.1 Microarquitetura Uniciclo

Os processadores uniciclo com extensões I e IM são implementados conforme o diagrama da Figura 3.2. O módulo de controle é implementado somente com lógica combinacional, e a frequência máxima de operação é limitada pela instrução mais lenta do processador.

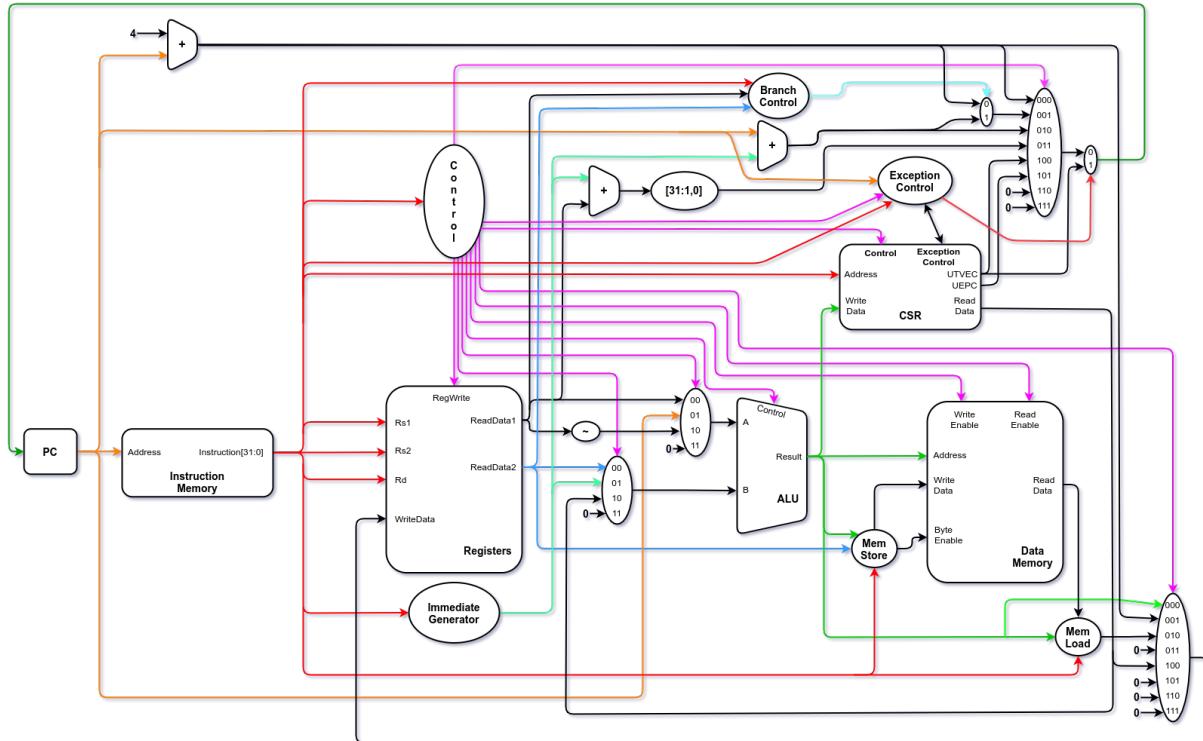


Figura 3.2: Diagrama da implementação das *ISAs* RV32I e RV32IM na microarquitetura uniciclo.

O processador uniciclo com extensão IMF é implementado conforme o diagrama da Figura 3.3. A unidade lógica e aritmética de ponto flutuante utiliza uma frequência de *clock* maior que a do resto do processador, e é o único módulo da implementação uniciclo que utiliza mais de um ciclo de relógio para realizar sua operação. A frequência máxima de operação do *clock* principal do processador continua limitada pela operação mais lenta.

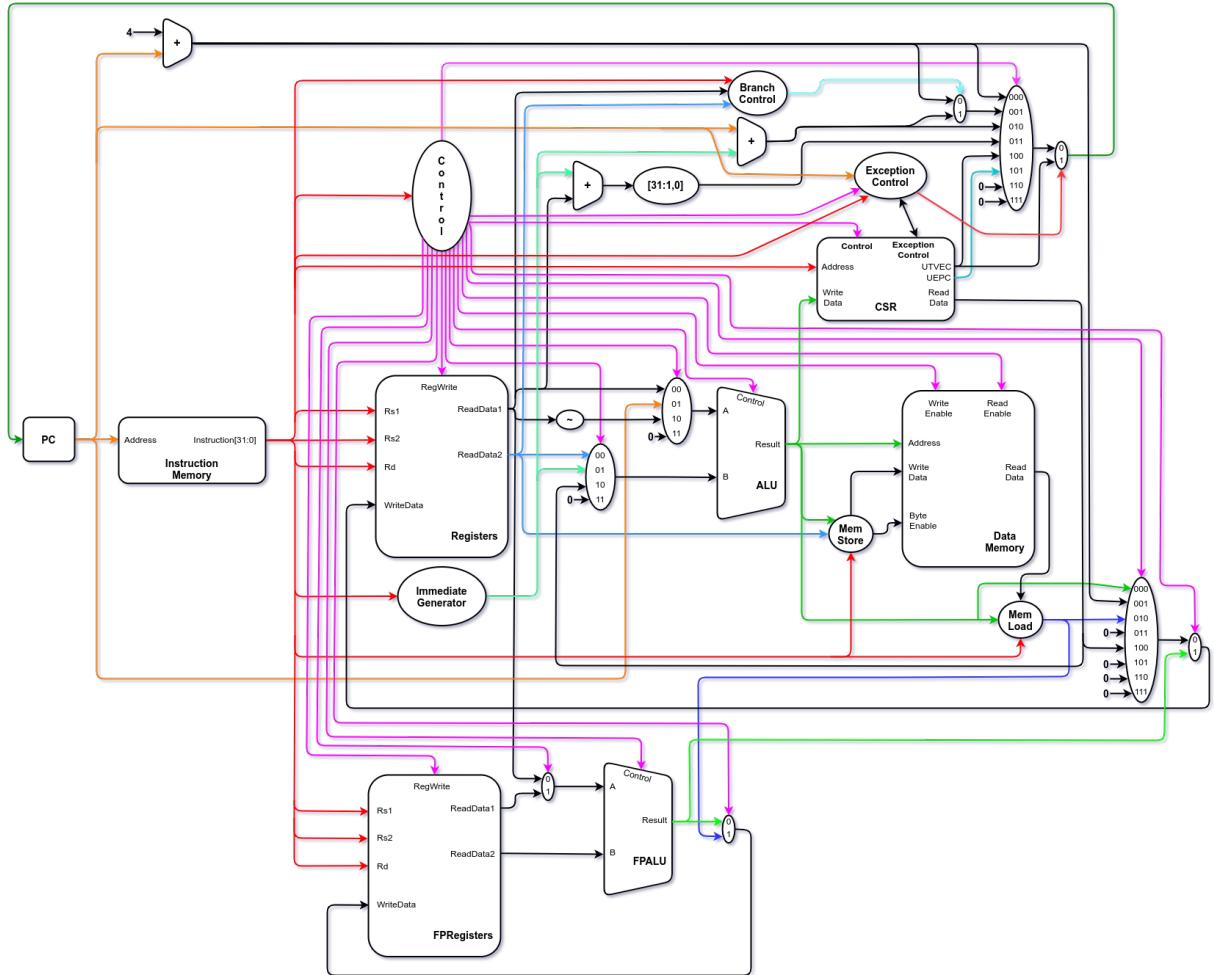


Figura 3.3: Diagrama da implementação da *ISA RV32IMF* na microarquitetura uniciclo.

3.1.2 Microarquitetura Multiciclo

Os processadores multiciclo com extensões I e IM são implementados conforme o diagrama da Figura 3.4. A unidade de controle é implementada utilizando microcódigo para executar as instruções. Com isso, a frequência de operação do processador depende da operação mais lenta do microcódigo, e não da execução da instrução completa.

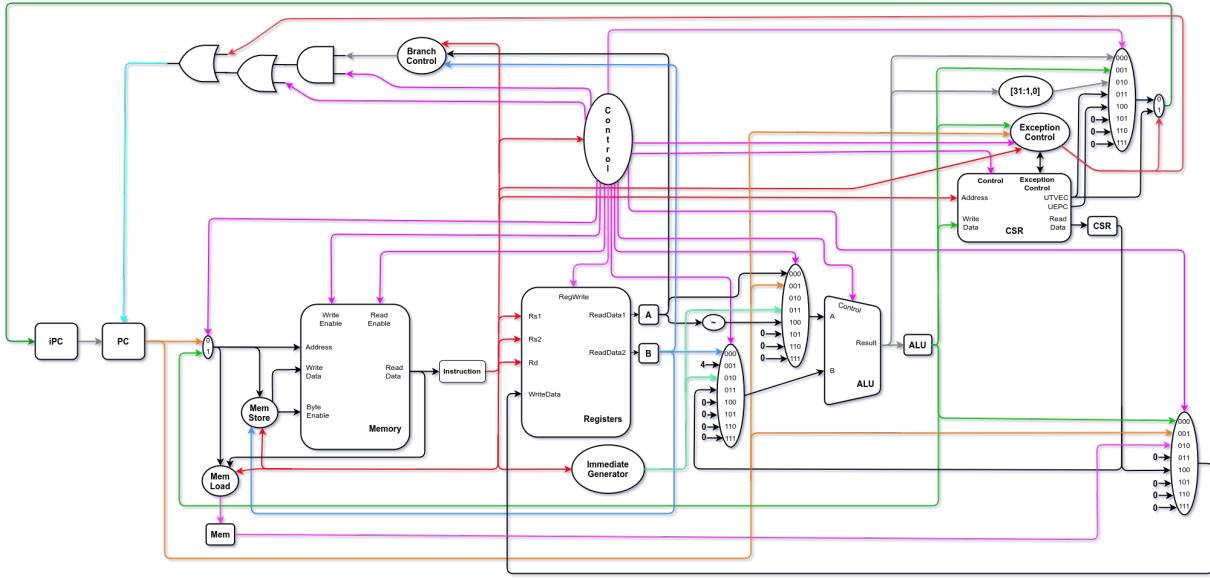


Figura 3.4: Diagrama da implementação das *ISAs* RV32I e RV32IM na microarquitetura multiciclo.

O processador multiciclo com extensões IMF é implementado conforme o diagrama da Figura 3.5. A unidade lógica e aritmética de ponto flutuante utiliza uma frequência de *clock* mais alta que a do resto do processador, e possui um sinal de *ready* que causa o *stall* do clock principal do processador enquanto a operação de ponto flutuante não completa. Assim, a frequência do *clock* do processador é variável, já que em operações de ponto flutuante o ciclo do relógio é mais longo que em outras operações.

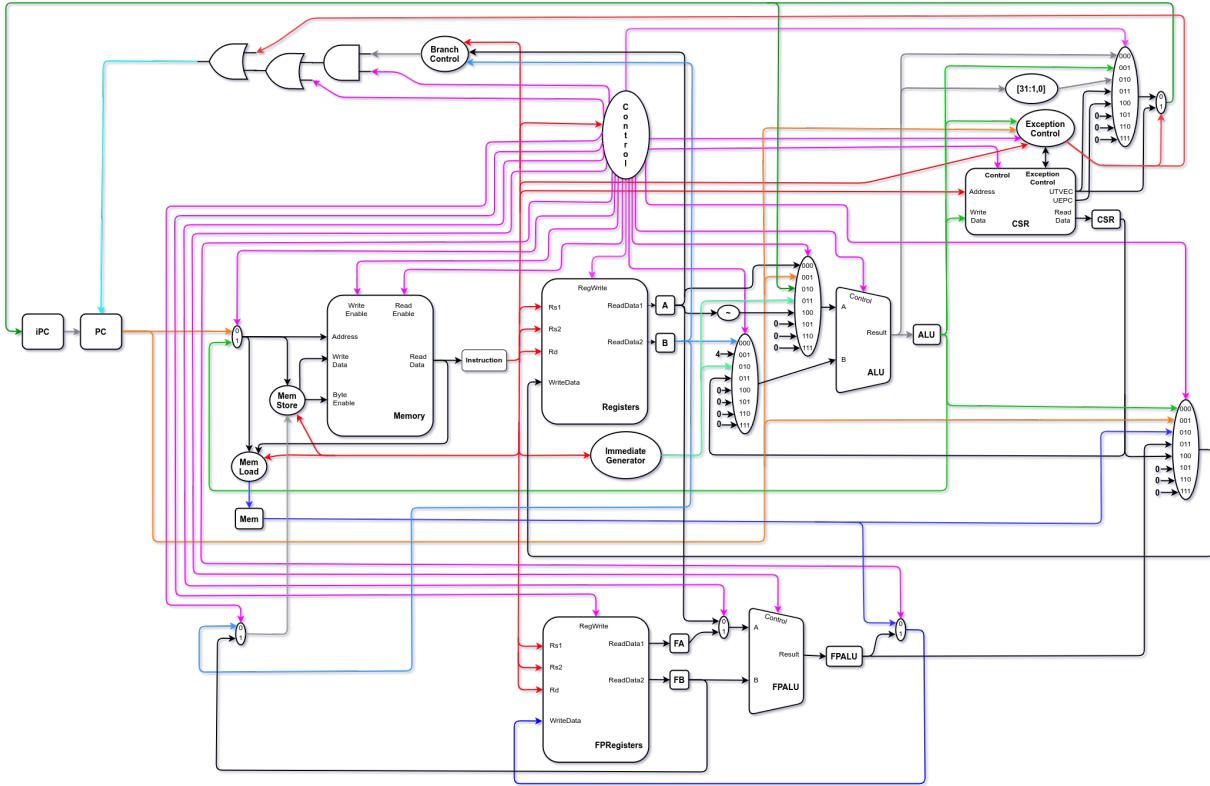


Figura 3.5: Diagrama da implementação da ISA RV32IMF na microarquitetura multiciclo.

3.1.3 Microarquitetura *Pipeline* de 5 Estágios

Os processadores *pipeline* com extensões I e IM são implementados conforme o diagrama da Figura 3.6. Seus cinco estágios são:

1. *Instruction Fetch*
2. *Instruction Decode*
3. *Execution*
4. *Memory Stage*
5. *Write Back*

A frequência máxima do seu *clock* é limitada pela operação mais lenta da unidade lógica e aritmética na terceira etapa do *pipeline*.



Figura 3.6: Diagrama da implementação das *ISAs* RV32I e RV32IM na microarquitetura *pipeline* de 5 estágios.

O processador *pipeline* com extensões IMF é implementado conforme o diagrama da Figura 3.7.



Figura 3.7: Diagrama da implementação da *ISA* RV32IMF na microarquitetura *pipeline* de 5 estágios.

3.2 Chamadas de sistema

A pasta `system` contém a implementação das chamadas de sistema do processador. O código *assembly* deve incluir o arquivo `system/MACROv21.s` no início do programa e o arquivo `system/SYSTEMv21.s` ao final do programa.

```
# Inicio do programa
.include "MACROv21.s"

# Dados do programa
.data
    ...

# Instrucoes do programa
.text
    ...

# Chamadas de sistema
.include "SYSTEMv21.s"
```

O arquivo `MACROSV21.s` insere macros para testar se o programa está sendo executado no Rars ou na FPGA ou no ModelSim para decidir o uso de determinadas *syscalls*, e também fornece a implementação por *software* de algumas instruções caso a extensão necessária não esteja implementada no processador.

Os endereços de memória dos periféricos acessados por *MMIO* também estão presentes como definições `.eqv` a fim de facilitar a implementação do programa. Por fim, o endereço inicial do nível privilegiado do sistema é gravado no *CSR UTVEC* e as interrupções são ativadas. Como o processador implementado não possui memória reservada para o *kernel*, a posição inicial de memória varia de acordo com o tamanho do programa implementado.

Já o arquivo `SYSTEMV21.s` implementa o *kernel* do sistema, tratando exceções e executando as *syscalls*. As chamadas de sistema implementadas são apresentadas na Tabela 3.1.

Tabela 3.1: Tabela de *syscalls* implementadas.

<i>syscall</i>	a7	Argumentos	Operação
Print Integer	1 ou 101	a0 = inteiro a1 = coluna a2 = linha a3 = cores a4 = frame	Imprime no <i>frame</i> a4 o número inteiro a0 (complemento de 2) na posição (a1,a2) com as cores a3[7:0] de <i>foreground</i> e a3[15:8] de <i>background</i> .
Print Float	2 ou 102	fa0 = float a1 = coluna a2 = linha a3 = cores a4 = frame	Imprime no <i>frame</i> a4 o número de ponto flutuante fa0 na posição (a1,a2) com as cores a3[7:0] de <i>foreground</i> e a3[15:8] de <i>background</i> .
Print String	4 ou 104	a0 = endereço da string a1 = coluna a2 = linha a3 = cores a4 = frame	Imprime no <i>frame</i> a4 a <i>string</i> iniciada no endereço a0 e terminada em <i>NULL</i> na posição (a1,a2) com as cores a3[7:0] de <i>foreground</i> e a3[15:8] de <i>background</i> .
Read Int	5 ou 105		Retorna em a0 o valor do inteiro em complemento de 2 lido do teclado.
Read Float	6 ou 106		Retorna em a0 o valor do <i>float</i> com precisão simples lido do teclado.
Read String	8 ou 108	a0 = endereço do buffer a1 = número máximo de caracteres	Escreve no <i>buffer</i> iniciado em a0 os caracteres lidos, terminando com um caractere <i>NULL</i> .

Tabela 3.1: Tabela de *syscalls* implementadas.

<i>syscall</i>	a7	Argumentos	Operação
Exit	10 ou 110		Retorna ao sistema operacional. Na <i>DE1-SoC</i> , trava o processador.
Print Char	11 ou 111	a0 = char ASCII a1 = coluna a2 = linha a3 = cores a4 = frame	Imprime no <i>frame</i> a4 o caracter a0 na posição (a1,a2) com as cores a3[7:0] de <i>foreground</i> e a3[15:8] de <i>background</i> .
Read Char	12 ou 112		Retorna em a0 o valor ASCII do caracter lido do teclado.
Time	30 ou 130		Retorna o tempo do sistema em <i>ms</i> , com os 32 <i>bits</i> menos significativos em a0 e os 32 <i>bits</i> mais significativos em a1.
MIDI Out Assíncrono	31 ou 131	a0 = pitch a1 = duração (<i>ms</i>) a2 = instrumento a3 = volume	Gera o som definido e retorna imediatamente.
Sleep	32 ou 132	a0 = duração (<i>ms</i>)	Coloca o processador em <i>sleep</i> por a1 <i>ms</i> .
MIDI Out Síncrono	33 ou 133	a0 = pitch a1 = duração (<i>ms</i>) a2 = instrumento a3 = volume	Gera o som definido e retorna após o término da execução da nota.
Print Integer	34 ou 134	a0 = inteiro a1 = coluna a2 = linha a3 = cores a4 = frame	Imprime no <i>frame</i> a4 o número inteiro a0 em formato hexadecimal na posição (a1,a2) com as cores a3[7:0] de <i>foreground</i> e a3[15:8] de <i>background</i> .
Print Integer Unsigned	36 ou 136	a0 = inteiro a1 = coluna a2 = linha a3 = cores a4 = frame	Imprime no <i>frame</i> a4 o número inteiro a0 sem sinal na posição (a1,a2) com as cores a3[7:0] de <i>foreground</i> e a3[15:8] de <i>background</i> .
Rand	41 ou 141		Retorna um número pseudorandômico de 32 <i>bits</i> em a0.

Tabela 3.1: Tabela de *syscalls* implementadas.

<i>syscall</i>	a7	Argumentos	Operação
Draw Line	47 ou 147	a0 = x_0 a1 = y_0 a2 = x_1 a3 = y_1 a4 = cor a5 = frame	Desenha no <i>frame</i> a5 uma linha reta do ponto (a0,a1) até o ponto (a2,a3) com as cores a3[7:0] de <i>foreground</i> e a3[15:8] de <i>background</i> .
Read Char	48 ou 148	a0 = cor a1 = frame	Preenche o <i>frame</i> a1 com a cor a0.

As *ecalls* 1XX são utilizadas no *Rars* pelas ferramentas *Bitmap Display Tool* e *Keyboard Display MMIO Tool*, que foram customizadas para funcionar de maneira idêntica quando o programa é executado na *FPGA*.

3.3 Interface de vídeo e depuração

A interface de vídeo possui resolução de 320x240 *pixels* com 8 *bits* de cor para cada pixel. Efectivamente, a interface de vídeo possui 255 cores diferentes e uma cor utilizada como transparência, o magenta 0xC7. Ela também conta com dois *framebuffers*, permitindo renderizar duas imagens diferentes e alternar entre elas, ou se aplicado em um jogo, permite a transição de *frames* sem *flickering*: enquanto um *frame* é exibido, o outro *framebuffer* é construído com as imagens do próximo *frame*, e quando pronto, a tela é atualizada com o novo *frame* completamente renderizado.

A conexão do vídeo do sistema é feita por interface VGA, podendo se conectar a qualquer monitor com entrada VGA. A resolução real da interface é de 640x480 *pixels* com taxa de atualização de 59 Hz por questões de compatibilidade com os monitores. Cada *pixel* da interface de vídeo representa uma célula de 4 *pixels* na saída de vídeo real. A saída de vídeo VGA também possui 24 *bits* de cor, pois o controlador faz a conversão das cores em 8 *bits* para três canais de 8 *bits*, um verde, um vermelho e um azul.



Figura 3.8: Exibição do *frame* de vídeo da *FPGA*.

Acionando um *switch* da *FPGA*, é mostrado por cima do *frame* um *menu On Screen Display* que mostra o valor atual contido nos bancos de registradores do processador, incluindo os *CSRs* e, caso a extensão F esteja implementada, outro *switch* permite alternar entre a visualização dos registradores de ponto flutuante e os de ponto fixo.

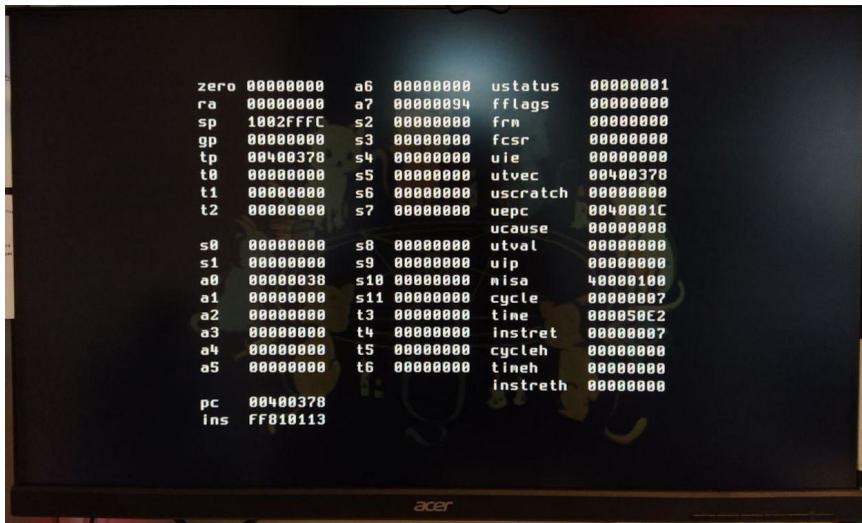


Figura 3.9: *Menu OSD* exibindo os valores dos registradores do processador.

O *menu OSD* é implementado como uma matriz de 52x24 caracteres monoespaçados. Na matriz, os caracteres que não mudam com o tempo, como é o caso do nome dos registradores, são representados com um parâmetro correspondente ao próprio caractere. Já os valores que se alteram, como o valor dos registradores, são representados por um parâmetro *placeholder* e o valor a ser mostrado na tela é obtido usando uma tabela de *lookup*. O projeto do *menu OSD* foi pensado de forma que possa ser modificado para expansão ou utilização em outras arquiteturas de processadores de maneira simples.

3.4 Configuração e síntese do processador pelo Quartus

O software utilizado para a síntese do processador, fornecimento de *IPs* como as de memória e operações de ponto flutuante, gravação do *soft-core*, dentre outras funcionalidades é o *Intel Quartus Lite v18.1*. Versões superiores são compatíveis com menos sistemas operacionais e/ou não possuem todos os *IPs* necessários para a síntese do processador.

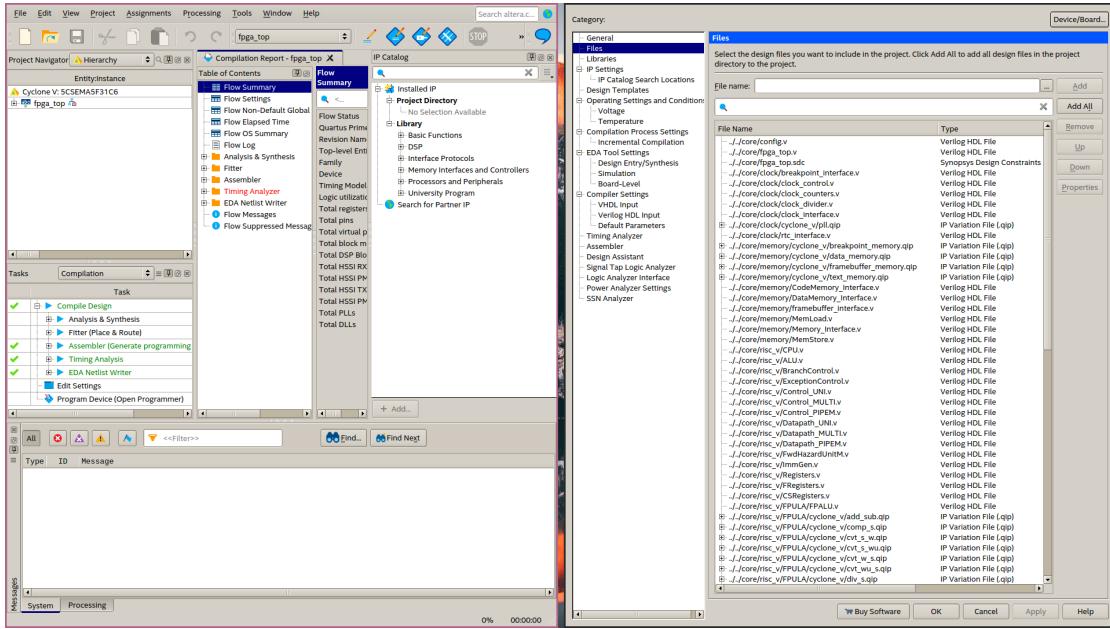


Figura 3.10: *Intel Quartus Lite v18.1* com a janela de configurações do projeto.

Todas as configurações do projeto também podem ser alteradas manualmente no arquivo `project/de1_soc/fpga_top.qsf`. Para ativar ou desativar os pinos do *chip* da *FPGA* que conectam os periféricos da placa, é preferível que a edição seja feita diretamente no arquivo de configuração, comentando ou descomentando a declaração dos pinos.

Para realizar a síntese completa do processador para poder utilizá-lo na *FPGA*, basta acessar o menu **Processing > Start Compilation** ou utilizar o atalho **Ctrl + L** ou clicar no ícone de "Play" na barra de tarefas do programa. Assim, as etapas de Análise e Síntese, *Placing* e *Routing*, *Assembler* e *Timing Analysis* serão realizadas, e no caso de não ocorrerem erros durante o processo, o *soft-core* estará pronto para ser gravado na *FPGA* utilizando o arquivo `.mif` gerado.

3.5 Simulação do processador pelo Quartus e ModelSim

O projeto possui um *testbench* em *Verilog* para simular as entradas e saídas da *FPGA* que o usuário operaria, como os botões e *switches*. Ele configura a rotina de reinicialização da placa após o *power-up* e define por quanto tempo a simulação será executada.

O script `test/simulation_scripts/de1_soc_rtl.do` é necessário para realizar a simulação de forma correta. O script gerado automaticamente pelo *Quartus* apresenta problemas que im-

pedem que a simulação seja executada corretamente, além de não gerar o arquivo de saída da simulação no formato desejado.

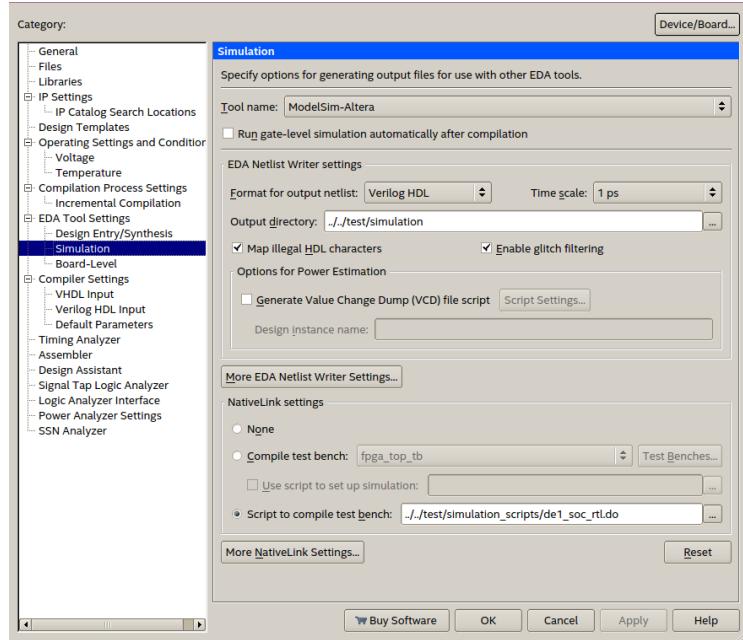


Figura 3.11: Janela de configuração da simulação no *Quartus*.

Por limitações do *Quartus*, não é possível simular *FPGAs Cyclone V* a nível de portas lógicas, somente sendo possível fazer a simulação *RTL*. Por outras limitações no programa, o *script .do* produzido manualmente só é executado usando o menu *Tools > Run Simulation Tool > Gate Level Simulation...*, que apesar do nome, executará uma simulação *RTL*. A opção *Tools > Run Simulation Tool > RTL Simulation*, utiliza o script gerado automaticamente e não funciona.

Ao executar a simulação, os *scripts NativeLink* iniciarão o programa *ModelSim*, carregando o *script .do* customizado, conforme mostrado na Figura 3.12. Ao finalizar a simulação, um arquivo *.vcd* será gerado e poderá ser analisado em *softwares* de visualização de formatos de onda, como o *GTKWave* ou o próprio *ModelSim*.

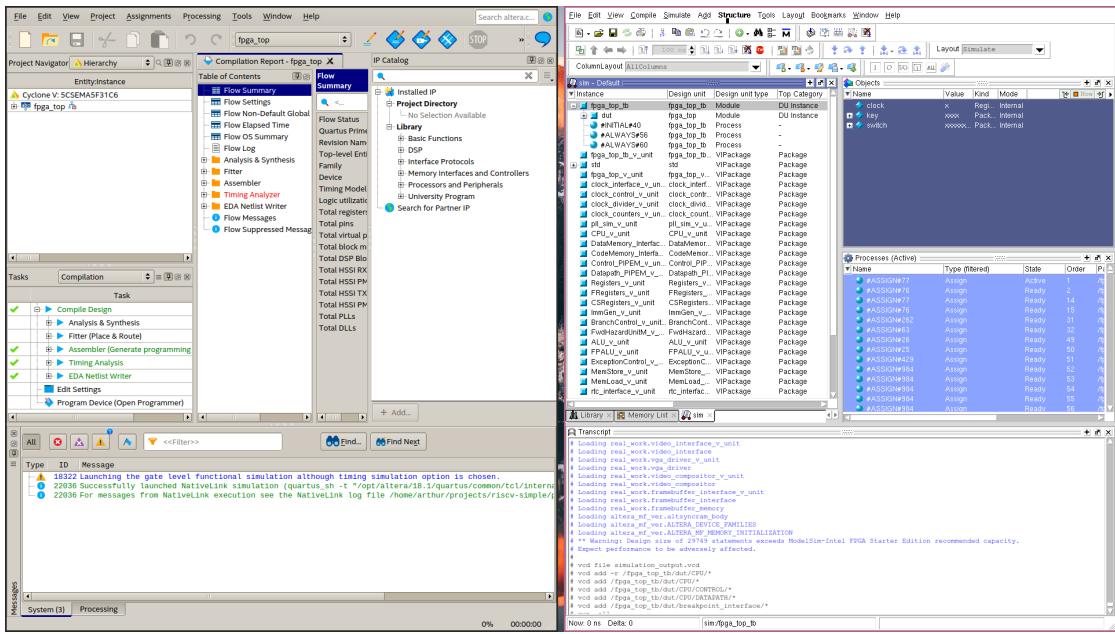


Figura 3.12: O *script NativeLink* invoca o *ModelSim* passando o *script .do* com as informações de como simular o sistema.

3.6 Script make.sh

A pasta `test/sof_library` contém os arquivos `.sof` com as nove variações da última versão do processador, prontos para serem gravados na *FPGA*. Para facilitar a geração das nove variantes, o *bash script* `make.sh` foi criado para automatizar a síntese e salvar a nova versão na pasta `test/sof_library`. O *script* também permite realizar somente a etapa de análise das variantes para confirmar que alterações feitas no código não introduziram erros de compilação, uma vez que é um processo muito mais ágil que realizar a síntese completa.

O *script* também faz a simulação *RTL* de todas as variantes do processador, salvando os *logs* e arquivos `.vcd` na pasta `test/simulation`, que é ignorada pelo *git*.

3.7 Uso da FPGA DE1-SoC

A placa de desenvolvimento *terasIC DE1-SoC* utilizada no projeto é mostrada na Figura 3.13.

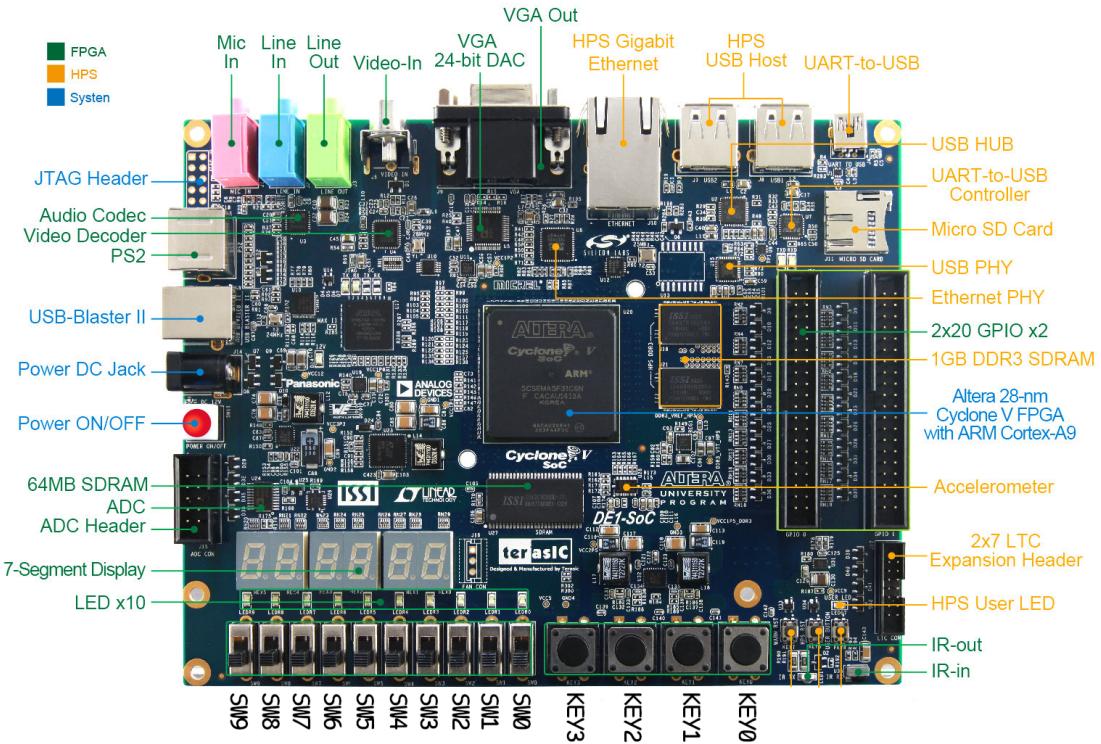


Figura 3.13: Placa de desenvolvimento *terasIC DE1-SoC*. Fonte: [31]

Os botões e *switches* mostrados na Figura 3.13 são utilizados para controlar as características do *clock* do processador, fazer seu *reset* e controlar o *menu OSD* de depuração. A função de cada *input* é:

- KEY0: Reset do processador;
- KEY1: Seletor de divisor de *clock* lento ou rápido;
- KEY2: Seletor de *clock* manual ou automático;
- KEY3: Gerador de *clock* manual;
- SW0: Bit [0] do divisor de *clock*;
- SW1: Bit [1] do divisor de *clock*;
- SW2: Bit [2] do divisor de *clock*;
- SW3: Bit [3] do divisor de *clock*;
- SW4: Bit [4] do divisor de *clock*;
- SW5: Temporizador para *stall* do processador;
- SW6: Seletor de *framebuffer* a ser exibido;
- SW7: Seletor de banco de registradores no *menu OSD*;

- SW8: Sem função;
- SW9: Habilita o *menu OSD*;

O procedimento recomendado para inicialização do processador após o *Programmer* do *Quartus* programar a *FPGA* com um novo arquivo *.mif* é: Pressionar e soltar a KEY2 para ativar o *clock* automático; pressionar e soltar a KEY0 para dar o *reset* dos estados do processador e, caso queira uma execução mais rápida, pressionar e soltar a KEY1 para mudar para um divisor de *clock* mais veloz. A frequência máxima de operação do *clock* é de 50MHz, e ocorre na opção de divisor rápido com o divisor 5'b1.

Utilizando a saída de vídeo, o sistema pode executar programas gráficos como jogos, ou pode ser usado simplesmente como ferramenta de *debug*. Ativando o *menu OSD* e utilizando o *clock* manual, é possível ver a progressão dos registradores do processador instrução por instrução.

O processador também pode receber *inputs* do usuário utilizando um teclado *PS/2*. A leitura do teclado é realizada por meio de *polling* do endereço do *buffer* do teclado.

A interface *RS-232* também pode ser utilizada para enviar e receber dados provenientes de outro computador, permitindo contornar a limitação de pouca memória disponível na *FPGA*, enviando novos dados e instruções à medida em que forem necessários e/ou requisitados pelo processador.

Capítulo 4

Resultados

Capítulo 5

Conclusões

5.1 Perspectivas Futuras

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] WATERMAN, A. et al. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA Version 20191213*. 2019.
- [2] WATERMAN, A. et al. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture Version 20190608*. 2019.
- [3] LYONS, J. *Natural language and universal grammar*. Cambridge England New York: Cambridge University Press, 1991. ISBN 9780521246965.
- [4] LAMBTRON. *Block diagram of a basic computer with uniprocessor CPU. (CC BY-SA 4.0)*. 2015. Disponível em: <<https://en.wikipedia.org/wiki/File:ABasicComputer.gif>>.
- [5] COLLEGE, A.; DOYLE, C.; RYNNING, A. *FPGA Flexible Architecture - Olin College of Engineering*. Disponível em: <http://ca.olin.edu/2005/fpga_dsp/images/fpga001.jpg>.
- [6] STANNERED. *Switch Box*. Disponível em: <https://en.wikipedia.org/wiki/File:Switch_box.svg>.
- [7] INTEL. *Cyclone V SoC FPGA Architecture*. Disponível em: <<https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/v.html>>.
- [8] INTEL. *ALM High-Level Block Diagram for Cyclone V Devices*. Disponível em: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_5v2.pdf>.
- [9] DOCUMENTATION/PLATFORMS/RISCV. Disponível em: <<https://wiki.qemu.org/Documentation/Platforms/RISCV>>.
- [10] WESTERN Digital to Use RISC-V for Controllers, Processors, Purpose-Built Platforms. Disponível em: <<https://www.anandtech.com/show/12133/western-digital-to-develop-and-use-risc-v-for-controllers>>.
- [11] RISC-V in Nvidia. Disponível em: <<https://riscv.org/wp-content/uploads/2017/05/Tue1345pm-NVIDIA-Sijstermans.pdf>>.
- [12] NEW Part Day: A RISC-V CPU For Eight Dollars. Disponível em: <<https://hackaday.com/2019/02/14/new-part-day-a-risc-v-cpu-for-eight-dollars/>>.
- [13] HIFIVE1 Rev B. Disponível em: <<https://www.sifive.com/boards/hifive1-rev-b>>.

- [14] BOOM Open Source RISC-V Core Runs on Amazon EC2 F1 Instances. Disponível em: <<https://www.cnx-software.com/2018/12/13/boom-risc-v-core-amazon-ec2-f1>>.
- [15] HIFIVE Unmatched. Disponível em: <<https://www.sifive.com/boards/hifive-unmatched>>.
- [16] BEAGLEV The First Affordable RISC-V Computer Designed to Run Linux. Disponível em: <<https://beaglev.seeed.cc/>>.
- [17] SIFIVE Tapes Out First 5nm TSMC 32-bit RISC-V Chip with 7.2 Gbps HBM3. Disponível em: <<https://www.tomshardware.com/news/openfive-tapes-out-5nm-risc-v-soc>>.
- [18] GCC RISC-V Options. Disponível em: <<https://gcc.gnu.org/onlinedocs/gcc/RISC-V-Options.html>>.
- [19] LLVM Clang RISC-V Now Supports LTO. Disponível em: <<https://riscv.org/news/2019/10/llvm-clang-risc-v-now-supports-lto-michael-larabel-phoronix>>.
- [20] RISC-V Port Merged to Linux. Disponível em: <<https://groups.google.com/a/groups.riscv.org/g/sw-dev/c/2-u-c3kyZlc>>.
- [21] FEDORA - Architectures/RISC-V. Disponível em: <<https://fedoraproject.org/wiki/Architectures/RISC-V>>.
- [22] PORTING Alpine Linux to RISC-V. Disponível em: <<https://drewdevault.com/2018/12/20/Porting-Alpine-Linux-to-RISC-V.html>>.
- [23] ANDROID 10 ported to homegrown multi-core RISC-V system-on-chip by Alibaba biz, source code released. Disponível em: <https://www.theregister.com/2021/01/21/android_riscv_port>.
- [24] MY Haiku RISC-V port progress. Disponível em: <<https://discuss.haiku-os.org/t/my-haiku-risc-v-port-progress/10663/85>>.
- [25] SEL4 is verified on RISC-V. Disponível em: <<https://microkerneldude.wordpress.com/2020/06/09/sel4-is-verified-on-risc-v/>>.
- [26] TECHNOLOGIES, I. *Acquisition of MIPS Technologies completed*. Disponível em: <<https://web.archive.org/web/20131002214436/http://www.imgtec.com/news/Release/index.asp?NewsID=10663>>.
- [27] BLOOMBERG. *Imagination Technologies Agrees to Takeover by Canyon Bridge*. Disponível em: <<https://www.bloomberg.com/news/articles/2017-09-22/imagination-technologies-agrees-to-takeover-by-canyon-bridge>>.
- [28] FELDMAN, M. *MIPS Acquired by AI Startup Wave Computing*. Disponível em: <<https://www.top500.org/news/mips-acquired-by-ai-startup-wave-computing>>.
- [29] GIANFAGNA, M. *Wave Computing and MIPS Wave Goodbye*. Disponível em: <<https://semikiwi.com/ip/284876-wave-computing-and-mips-waves-goodbye>>.

- [30] TURLEY, J. *Wait, What? MIPS Becomes RISC-V.* Disponível em: <<https://www.eejournal.com/article/wait-what-mips-becomes-risc-v>>.
- [31] TERASIC. *DE1-SoC Board.* Disponível em: <<https://www.terasic.com.tw/attachment/archive/836/image>>.

ANEXOS

I. DESCRIÇÃO DO CONTEÚDO DO CD

Descrever CD.

II. PROGRAMAS UTILIZADOS

Quais programas foram utilizados?