UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉ LUIZ TINASSI D´AMATO

Toolchain MRISC16t: um sistema de auxílio ao desenvolvimento de aplicações para sistemas embarcados

Maringá

2012

ANDRÉ LUIZ TINASSI D´AMATO

Toolchain MRISC16t: um sistema de auxílio ao desenvolvimento de aplicações para sistemas embarcados

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientadora: Profª. Drª. Linnyer Beatrys Ruiz Aylon
Coorientador: Prof. Dr. Anderson Faustino da Silva

Maringá
2012

"Dados Internacionais de Catalogação-na-Publicação (CIP)"
(Biblioteca Setorial - UEM. Nupélia, Maringá, PR, Brasil)

D155t

D´Amato, André Luiz Tinassi, 1983-
Toolchain MRISC16t: um sistema de auxílio ao desenvolvimento de aplicações para sistemas embarcados / André Luiz Tinassi D´Amato. -- Maringá, 2012.
104 f. : il.(algumas color.).

Dissertação (mestrado em Ciência da Computação)--Universidade Estadual de Maringá, Dep. de Informática, 2012.
Orientadora: Profa. Dra. Linnyer Beatrys Ruiz Aylon.
Coorientador: Prof. Dr. Anderson Faustino da Silva.

1. Toolchain MRISC16t (Ferramenta operacional para desenvolvimento e análise de softwares). 2. Processador UNB-RISC16. 3. Consumo de energia. - Estimativa – Ferramenta Operacional. I. Universidade Estadual de Maringá. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação.

CDD 22. ed. -005.12
NBR/CIP - 12899 AACR/2

Maria Salete Ribelatto Arita CRB 9/858

FOLHA DE APROVAÇÃO

ANDRÉ LUIZ TINASSI D'AMATO

Toolchain MRISC16t: um sistema de auxílio ao desenvolvimento de aplicações para sistemas embarcados

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Prof. Dr. Anderson Faustino da Silva
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Ronaldo Augusto de Lara Gonçalves
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Joaquim Celestino Junior
Universidade Estadual do Ceará – MACC/UECE

Aprovada em: 03 de setembro de 2012.
Local da defesa: Sala 102, Bloco C56, *campus* da Universidade Estadual de Maringá.

Toolchain MRISC16t: um sistema de auxílio ao desenvolvimento de aplicações para sistemas embarcados

RESUMO

Sistemas embarcados apresentam severas restrições quanto a quantidade de memória, energia disponível e capacidade de processamento. Sendo assim, um software desenvolvido para esse tipo de sistema deve apresentar todas as suas funcionalidades com o mínimo possível de quantidade de código. Isso pode ser conseguido a partir do uso de um compilador que favoreça estas características. Este trabalho de dissertação lida com o desafio de propor um *toolchain* para auxílio ao desenvolvimento de software para o processador `UNB-RISC16`, que será utilizado em sistemas embarcados. O conjunto de componentes utilizados para definir o *toolchain* proposto, inclui a ferramenta `GCC` para geração de código otimizado, um montador para geração do código objeto, um viculador para produção de código binário executável, e a ferramenta R16-EPROF para para simulação e estimativa do consumo de energia despendida por uma aplicação executada pelo processador `UNB-RISC16`.

Palavras-chave: Geração de código. *Toolchain*. Estimativa de consumo. Energia. UNB-RISC16.

Toolchain MRISC16t: a support system to applications development for embedded systems

ABSTRACT

Embedded systems have severe restrictions on the amount of memory, available energy and processing capacity. Thus, a software developed for this type of system should provide all its functionality with the least possible amount of code. This can be achieved through the use of a compiler that fosters these characteristics. This dissertation deals with the challenge of proposing a toolchain to aid the development of software for the `UNB-RISC16` processor, which will be used in embedded systems. The set of components used to set the proposed toolchain includes the GCC tool for generating optimized code, an assembler to create the object code, a linker to produce a binary executable code, and the R16-EPROF tool for simulation and estimation of energy expended by an application executed by the `UNB-RISC16` processor.

Keywords: Code generation. Toolchain. Estimated consumption. Energy. `UNB-RISC16`.

LISTA DE FIGURAS

Figura - 2.1 Arquitetura do MIMOLA 17
Figura - 2.2 Arquitetura do *toolchain* SPAM 19

Figura - 3.1 Formato das instruções do processador UNB-RISC16 28
Figura - 3.2 Arquitetura do processador UNB-RISC16 30
Figura - 3.3 Estrutura de um compilador 35
Figura - 3.4 Arquitetura do GCC 38

Figura - 4.1 Formato de arquivo objeto relocável do processador UNB-RISC16 . 47

Figura - 5.1 Entradas e saídas do R16-EPROF 55

Figura - 6.1 Consumo de energia por aplicação 68
Figura - 6.2 Consumo de energia por função e regiões de código da aplicação Jfd otimizada 70
Figura - 6.3 Tamanho de código das aplicações 71
Figura - 6.4 Quantidade de acessos a memória pela aplicação Jfd 72

LISTA DE TABELAS

Tabela - 2.1 Recursos dos *Toolchains* 21
Tabela - 2.2 Recursos dos Simuladores 24

Tabela - 3.1 Instruções do processador `UNB-RISC16` 27
Tabela - 3.2 Banco de registradores do processador `UNB-RISC16` 28

Tabela - 4.1 Ferramentas que compõem o *toolchain* MRISC16T 42
Tabela - 4.2 Campos da seção de cabeçalho do formato `UNB-RISC16` 47
Tabela - 4.3 Campos da seção extern do formato `UNB-RISC16` 48
Tabela - 4.4 Campos da seção de símbolos realocáveis do formato `UNB-RISC16` 48
Tabela - 4.5 Campos da tabela de símbolos do formato `UNB-RISC16` 48

Tabela - 6.1 Conjunto de aplicações utilizadas 67

LISTA DE SIGLAS E ABREVIATURAS

RSSF: Redes de Sensores Sem Fio
RF: Rádio Freqüência
UCP: Unidade Central de Processamento
UNB: Universidade de Brasília
UCP: Unidade Central de Processamento
GCC: *GNU Compiler Collection*
RI: Representação Intermediária
RISC: *Reduced Instruction Set Computer*
MIPS: *Microprocessor with Interlocked Pipeline Stages*
ULA: Unidade Lógica Aritmética
RTL: *Register Transfer Language*
ACE: *Associated Computer Experts*
BFD: *Binary File Description*
SSA: *Static Single Assignment*
LLVM: *Low Level Virtual Machine*
RISC: *Reduced instruction set computing*
SRAM: *Static Random Access Memory*
ROM: *Read Only Memory*
CMOS: *Complementary Metal Oxide Semiconductor*
PC: *Program Counter*
GP: *Global Pointer*
SP: *Stack Pointer*
RA: *Return Address*
A/D: Analógico/Digital
MIPS: *Microprocessor without Interlocked Pipeline Stages*
CRC: *Cyclic Redundancy Check*
FFT: *Fast Fourier Transform*
LUD: Decomposição LU
DCT: *Discrete Cosine Transform*

SUMÁRIO

1 INTRODUÇÃO **12**

1.1 MOTIVAÇÃO 13

1.2 OBJETIVOS E CONTRIBUIÇÕES 14

1.3 ORGANIZAÇÃO DO TEXTO 14

2 *TOOLCHAINS* E SIMULADORES **16**

2.1 *TOOLCHAINS* PARA MICROPROCESSADORES 16

2.1.1 Mimola 16

2.1.2 Lcc 18

2.1.3 Spam 18

2.1.4 Cosy 19

2.1.5 Lance 20

2.1.6 Marion 20

2.2 SIMULADORES 22

2.2.1 Power Tossim 22

2.2.2 Avrora 22

2.2.3 Atemu 23

3 REFERENCIAL TEÓRICO **25**

3.1 SISTEMAS DE RSSF 25

3.2 O PROCESSADOR UNB-RISC16 26

3.2.1 Descrição da Implementação do Processador UNB-RISC16 29

3.3 COMPILADORES 33

3.3.1 A Estrutura de um Compilador 34

3.3.2 Compiladores Redirecionáveis 35

3.4 GCC: GNU *COMPILER COLLECTION* 36

3.4.1 A Arquitetura do GCC 37

3.4.2 Portando O GCC 39

4 O *TOOLCHAIN* MRISC16t **42**

4.1 FERRAMENTAS DE TRADUÇÃO 43

4.1.1 A Ferramenta R16-AS 43

4.1.2 A Ferramenta R16-LD 44

4.1.3 A Ferramenta R16-CPP 44

4.2 FERRAMENTAS DE ANÁLISE 44

4.2.1 A Ferramenta R16-OBJDUMP 44
4.2.2 A Ferramenta R16-GPROF 45
4.2.3 A Ferramenta R16-EPROF 45
4.3 FORMATO DE ARQUIVO OBJETO UNB-RISC16 46
4.3.1 Segmentos do Formato UNB-RISC16 46
4.4 RESTRIÇÕES DO PROCESSADOR UNB-RISC16 PARA GERAÇÃO DE CÓDIGO 49
4.4.1 Restrições Relativa ao Conjunto de Instruções 50

5 A FERRAMENTA R16-EPROF 54
5.1 A ARQUITETURA DA FERRAMENTA R16-EPROF 54
5.2 O MODELO UTILIZADO PARA ESTIMAR O CONSUMO DE ENERGIA 56
5.3 O MODELO DE ENERGIA DO PROCESSADOR UNB-RISC16 58
5.4 Modos da Ferramenta R16-EPROF 59
5.4.1 Estático 59
5.4.2 Dinâmico 59
5.5 OS RECURSOS DO *ENERGY PROFILER* 61
5.6 AS OPÇÕES DISPONÍVEIS NA FERRAMENTA R16-EPROF 62
5.6.1 Opções da Classe *Simulation* 62
5.7 OPÇÕES DA CLASSE *Monitors* 63
5.8 OPÇÕES DA CLASSE *INCODE* 65

6 RESULTADOS 66
6.1 METODOLOGIA 67
6.2 RESULTADOS OBTIDOS EM RELAÇÃO AO CONSUMO DE ENERGIA 67
6.2.1 Análise de Consumo de Energia com as Opções ENERGY-LOG e ENERGY-PROFILER 69
6.3 RESULTADOS OBTIDOS EM RELAÇÃO AO TAMANHO DE CÓDIGO, E ACESSOS A MEMÓRIA 70
6.3.1 Análise de Consumo de Memória Utilizando a Opção MEMORY-ANALYSIS 71
6.4 CONSIDERAÇÕES FINAIS DO EXPERIMENTO 72

7 CONCLUSÃO E TRABALHOS FUTUROS 74

REFERÊNCIAS 76

A APÊNDICE A **83**
A.1 Programa Dijkstra Escrito em Linguagem de Programação `C` 83
A.2 Código `Assembly` do Programa Dijkstra 84
A.3 Arquivo de Configuração Utilizado no Modo Dinâmico 89

B APÊNDICE B **90**
B.1 Saída Da Opção TIME-FLAGS com os *Breakpoints* `.dijkstra e .main` . . 90
B.2 Saída da Opção ENERGY-LOG 90
B.3 Saída da Opção EXEC 91
B.4 Saída da Opção INSTRUCTIONS-SHOW + STACK-SIZE 92
B.5 Saída do Modo Dinâmico 93
B.6 Saída da Opção INSTRUCTIONS-PROFILER 94
B.7 Saída da Opção MEMORY 96
B.8 Saída da Opção ENERGY-PROFILER 97

1 INTRODUÇÃO

Toolchain é um termo utilizado na computação para referenciar um conjunto de ferramentas de desenvolvimento de aplicações (Al Saad et al., 2008; Luk et al., 2009). Os *toolchains* são utilizados tanto no desenvolvimento de aplicações para sistemas não integrados quanto para sistemas embarcados. No entanto, o desenvolvimento de aplicações para sistemas embarcados é fundamentalmente diferente de sistemas não integrados (Graaf et al., 2003), pois a tecnologia utilizada no desenvolvimento de software para sistemas embarcados devem lidar com restrições que não estão presentes em sistemas não integrados. As restrições encontradas nos sistemas embarcados estão relacionadas a pouca quantidade de memória e energia disponíveis. Portanto é importante que os *toolchains* para sistemas embarcados possibilitem ao desenvolvedor gerar aplicações que possam ser armazenadas dentro da memória disponível, e que gastem a menor quantidade de energia possível.

Os nós de uma Rede de Sensores Sem Fio (RSSFs) (Choi et al., 2010; Fummi et al., 2012; Nabi et al., 2009) são exemplos de sistemas embarcados que apresentam as restrições mencionadas. Portanto é fundamental que os compiladores utilizados no desenvolvimento de aplicações para RSSFs sejam projetados visando a otimização do tamanho de código final. No entanto o tamanho de código de uma aplicação nem sempre influencia no consumo de processamento, no consumo de energia, e na quantidade de acessos a memória. Para obter informações sobre o efeito da execução de um programa de modo confiável é necessário executar as aplicações monitorando os ciclos de processamento despendidos, como também, a quantidade de leituras e escritas realizadas na memória. Os simuladores são programas de computador que permitem a execução virtual de aplicações reproduzindo de forma exata o comportamento de um processador.

Os simuladores são importantes ferramentas em ambientes de desenvolvimento de aplicações para RSSF, pois promovem uma plataforma coerente de teste, e depuração de código (Jeremiassen, 2000). A simulação de uma aplicação possibilita identificar também a quantidade de ciclos de instruções necessários para executar um programa, tal qual a quantidade de memória necessária para as operações de leitura e escrita. As ferramentas `Power Tossim` (Perla et al., 2008; Shnayder et al., 2004), `Avrora` (de Paz Alberola e Pesch, 2008; Titzer et al., 2005) e `Atemu` (Polley et al., 2004) são exemplos de ferramentas para simular a execução e o consumo de energia de uma aplicação para uma RSSF.

O grupo de engenharia elétrica da UNB (Universidade de Brasília) propôs o desenvolvimento do processador `UNB-RISC16` (Costa, 2004), um processador RISC de 16 bits projetado para uso em nós de RSSF. Para simplificar o desenvolvimento de *softwares* para o processador `UNB-RISC16` é necessário desenvolver um conjunto de ferramentas, ou *toolchain*, para geração e análise de código para este processador. Sendo assim a proposta deste trabalho é o desenvolvimento do conjunto de ferramentas MRISC16T (Manna RISC 16 bits *Toolchain*) (DAmato et al., 2011b), com o objetivo fornecer suporte ao desenvolvimento de *software* para o processador `UNB-RISC16`.

A contribuição do *toolchain* MRISC16T é fornecer tanto ferramentas de desenvolvimento, como também fornecer uma plataforma de análise e simulação das aplicações desenvolvidas, com o objetivo principal de permitir estimativa de consumo de energia. O *toolchain* MRISC16T possui como ferramenta de simulação o programa R16-EPROF (DAmato et al., 2011a). Essa ferramenta pode ser considerada inovadora, pois as funcionalidades encontradas no R16-EPROF não estão disponíveis completamente em outros simuladores de RSSF. O *toolchain* MRISC16T possui também uma versão do compilador GCC para o processador `UNB-RISC16`, o montador R16-AS, o vinculador R16-LD, e o desmontador de código R16-OBJDUMP. O compilador `GCC` foi selecionado para compor o MRISC16T devido a sua capacidade de otimização de código.

1.1 MOTIVAÇÃO

Considerando o contexto da criação do processador `UNB-RISC16` que será utilizado em nós sensores para RSSF, é necessário o desenvolvimento de ferramentas para construção de aplicações para esse processador. No entanto, os sistemas de RSSFs possuem restrições quanto a quantidade de energia e memória disponível. Sendo assim, as ferramentas de desenvolvimento de aplicações devem favorecer economia de energia e memória. Portanto a motivação deste trabalho é o desenvolvimento do *toolchain* MRISC16T, com

o objetivo de fornecer suporte ao desenvolvimento de aplicações com foco no processador `UNB-RISC16`.

1.2 OBJETIVOS E CONTRIBUIÇÕES

Este tópico aborda os recursos existentes no *toolchain* MRISC16T para o desenvolvimento de aplicações, considerando o contexto das restrições presentes no processador `UNB-RISC16`. O objetivo geral do *toolchain* MRISC16T é fornecer um ambiente de desenvolvimento de aplicações para o processador `UNB-RISC16`, sendo esse ambiente composto pelos seguintes tipos de ferramentas:

- Compilador para a linguagem de programação C, com suporte a otimização de código para o processador `UNB-RISC16`;
- Montador para gerar código objeto para o processador `UNB-RISC16`;
- Vinculador para realizar a ligação de módulos de programa;
- Desmontador de código para transformar código objeto em instruções do `UNB-RISC16`; e
- Simulador para executar aplicações com suporte a análise de execução.

Sendo assim, as contribuições desse trabalho de dissertação são as seguintes:

- Um *toolchain* para o desenvolvimento de aplicações para o processador `UNB-RISC16`;
- Um formato objeto compacto;
- Um simulador com suporte a *profile* de execução; e
- Uma ferramenta para estimar o consumo de energia despendido por uma aplicação.

1.3 ORGANIZAÇÃO DO TEXTO

A organização desta dissertação é a seguinte: O tópico 2 aborda os trabalhos relacionados utilizados na pesquisa para composição da base teórica. O tópico 3 aborda a base teórica utilizada no desenvolvimento do trabalho. As seções 4 e 5 abordam os recursos do*toolchain* MRISC16T juntamente com o formato de arquivo objeto proposto para o processador `UNB-RISC16`, e as soluções para geração de código considerando as restrições

do processador. O tópico 6 apresenta os resultados obtidos, e a tópico 7 apresenta as conclusões e os trabalhos futuros.

2

TOOLCHAINS E SIMULADORES

Esta parte do trabalho descreve algumas ferramentas de desenvolvimento de *software* para sistemas embarcados. A partir da pesquisa realizada com estes trabalhos, foi possível identificar as bases teóricas relacionadas com o desenvolvimento de ambientes de programação para sistemas embarcados.

A segunda parte sintetiza alguns simuladores de RSSF. Neste caso estes trabalhos foram fundamentais para constituir a base teórica para o desenvolvimento da ferramenta R16-EPROF. Isto devido ao fato do contexto de aplicação dos processadores utilizados em RSSF ser semelhante ao do processador `UNB-RISC16`, visto que as restrições energéticas e computacionais são as mesmas.

2.1 TOOLCHAINS PARA MICROPROCESSADORES

2.1.1 Mimola

O sistema `MIMOLA` (Nowak e Marwedel, 1989) (Leupers e Marwedel, 1998) é um conjunto de ferramentas colaborativas para desenvolvimento de programas para processadores utilizados em sistemas embarcados. Esses processadores são definidos por meio de uma descrição de *hardware* que segue o padrão HDL (Linguagem de Descrição de *Hardware*) (Shim et al., 2008). A arquitetura do sistema `MIMOLA` é apresentada na Figura - 2.1. A entrada do sistema é a descrição de uma aplicação (especificação do projeto) na linguagem do `MIMOLA`. A representação intermediária da aplicação é gerada a partir de sua respectiva representação RTL, que possibilita simulação, detecção de paralelismo e síntese do código

final. A alocação de *hardware* necessária para execução da aplicação é amplamente divulgada para o desenvolvedor. Nesta fase é possível gerar programas que podem ser simulados considerando as características do *hardware*.

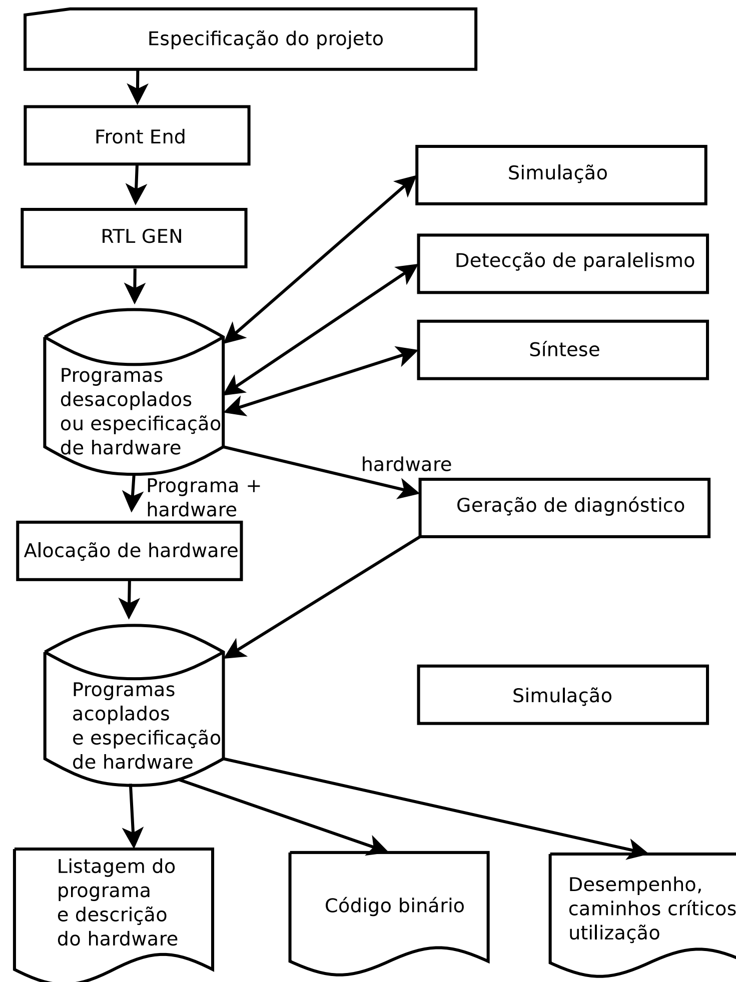

Figura 2.1: Arquitetura do `MIMOLA`

A linguagem de programação utilizada neste sistema é também chamado de `MIMOLA`. Esta linguagem é composta por um subconjunto de instruções da linguagem de programação `PASCAL` (Kolarski et al., 2008). A linguagem do sistema `MIMOLA` é utilizada para descrever uma aplicação, e o modelo HDL para especificar a arquitetura do processador alvo. Como compilador o sistema `MIMOLA` possui o `MSSQ` (Nowak e Marwedel, 1989). Assim como o `GCC` o compilador `MSSQ` também fornece otimização de código, porém apresenta resultados inferiores ao primeiro. Desta forma, isto pode ser considerado uma desvantagem em relação ao MRISC16T.

O MIMOLA possui uma ferramenta para realizar simulação em nível de aplicação utilizado a representação RTL (Nowak e Marwedel, 1989). Isto torna possível validar o correto funcionamento de uma aplicação. Uma das propostas MRISC16T *toll chain* é o desenvolvimento de uma ferramenta para executar o código de uma aplicação para o processador UNB-RISC16 e estimar consumo de recursos de hardware e energia. O sistema MIMOLA também possui uma ferramenta que gera diagnósticos sobre a utilização do *hardware*, no entanto não realiza estimativa de consumo de energia.

2.1.2 Lcc

LCC (Little C Compiler) (Newburn et al., 2011) é um compilador para a linguagem de programação C que gera código para as arquiteturas VAX, Motorola 68020, SPARC, e MIPS R3000. Produzido na Universidade de Princeton pelo laboratório da empresa AT&T (Newburn et al., 2011), o compilador LCC gera código com qualidade comparável ao GCC. Assim como o GCC, o LCC define o processador alvo por meio de um arquivo de descrição de máquina.

O LCC possui dois interessantes recursos para auxiliar o desenvolvimento de código pelo programador. Esses recursos são disponibilizados por duas ferramentas, *Debug* e *profile*.

Debug permite ao programador identificar erros como ponteiros vazios, que é um erro comum em programação em linguagem C. Alguns compiladores, como por exemplo o GCC permitem a construção de uma aplicação que apresente esse problema resultando em falha de segmentação. Se o código fonte de uma aplicação apresentar ponteiros vazios, o LCC reporta o nome do arquivo e a linha de código que ocasiona o problema.

O *profile* retorna em um arquivo chamado prof.out, o diagnóstico sobre a execução de uma aplicação exibindo o número de chamadas realizadas em cada função. O *toolchain* MRISC16T possui a ferramenta (r16-eprof), que realiza a estimativa de consumo de energia e acessos realizados a memória no momento da execução de uma aplicação desenvolvida para o processador UNB-RISC16.

2.1.3 Spam

O Spam (Sudarsanam et al., 2002) é um conjunto de ferramentas colaborativas de desenvolvimento de aplicações para sistemas embarcados. O sistema Spam implementa um compilador para linguagem C incorporando duas ferramentas, SUIT (Sudarsanam et al., 2002) e TWIF (Sudarsanam et al., 2002). A etapa de análise léxica, sintática e geração da representação intermediária é realizada pela ferramenta SUIT desenvolvida na linguagem

C. Já o *back-end* que constitui as etapas de geração de código `Assembly`, otimização e geração de código alvo é implementado na ferramenta `TWIF` desenvolvida em `C++`.

A descrição dos processadores é realizada por uma classe chamada *Compacted Instruction* que contém a definição das operações em `Assembly`, juntamente com os respectivos *opcodes* a serem gerados. A arquitetura do compilador SPAM pode ser visualizado na Figura - 2.2. O `Spam` aplica otimizações na representação intermediária, como também no código `Assembly`.

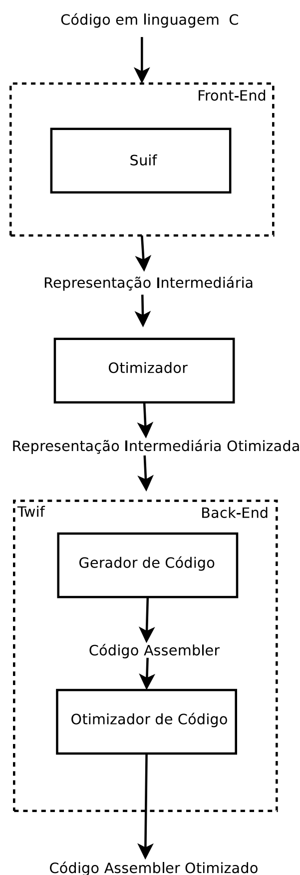

Figura 2.2: Arquitetura do *toolchain* `SPAM`

O foco da ferramenta SPAM é a otimização e geração de código. Quando comparado com as ferramentas descritas anteriormente, `Spam` apresenta a desvantagem de não fornecer ferramentas para análise de código e análise da utilização dos recursos de hardware. Quando comparado com a ferramenta `Spam` o *toolchaim* MRISC16T apresenta vantagens, pois proporciona tanto análise dos recursos computacionais quanto do consumo de energia.

2.1.4 Cosy

Cosy (Engel et al., 2011) é um compilador para arquiteturas customizáveis de *hardware* criado pela ACE. O *front-end* da ferramenta reconhece um conjunto de linguagens de programação que inclui as linguagens C e C++. Cosy possui um otimizador de código em sua arquitetura que proporciona a geração de código Assembly com qualidade semelhante aos produzidos pelo GCC, porém apresenta uma arquitetura modular melhor definida que facilita a adição de passos de otimização por meio de uma interface simples. A customização da máquina alvo é realizada de modo semelhante ao GCC por meio de arquivos MDF (machine description format), que contém a descrição das arquiteturas.

Os dois principais focos da ferramenta Cosy são a geração de códigos otimizados, e a customização das arquiteturas de processador. O sistema não apresenta ferramentas para análise de fluxo de programa, e consumo de *hardware*. Quanto ao esforço para descrever uma nova arquitetura, o compilador Cosy apresenta uma complexidade semelhante ao compilador GCC. Sendo assim, é possível constatar que os recursos encontrados na ferramenta Cosy não oferecem algo de inovador em relação ao que existe na ferramenta GCC. Desta forma, o *toolchain* MRISC16T representa inovação quando comparado com a ferramenta Cosy, pois além de produzir código otimizado o MRISC16T ainda disponibiliza a ferramenta de análise R16-EPROF.

2.1.5 Lance

Lance (Werner-Allen et al., 2008) é um sistema de software utilizado em projetos de compiladores para sistemas embarcados. Essa ferramenta é composta por um *front-end* C, um gerador de representação intermediária, um analisador de fluxo de dados, uma interface para o *back-end* e dois otimizadores (otimizador de representação intermediária e otimizador de código Assembly).

O *front-end* C realiza uma análise do código fonte e gera uma representação intermediária em um nível mais baixo. A representação intermediária resultante não contém nenhuma informação sobre a máquina alvo.

A ferramentas de otimização contém uma biblioteca de componentes que realizam eliminação de trechos de código desnecessários. O compilador Lance possui uma ferramenta de análise do fluxo de dados de um programa descrito em linguagem C. Essa ferramenta possibilita a visualização gráfica da estrutura do programa. De acordo com as características citadas, o sistema Lance é uma boa opção como ferramenta de desenvolvimento de aplicações em sistemas embarcados. No entanto, ao contrário do

toolchain MRISC16T o sistema Lance não apresenta ferramentas para estimativa de consumo de recursos computacionais e de energia.

2.1.6 Marion

O sistema MARION (Bradlee et al., 1991) é um sistema utilizado para geração de código para arquiteturas RISC. Essa ferramenta produz geradores de códigos para os processadores Motorola 88000, MIPS R2000, e Intel i860. MARION é composto por três componentes: o construtor de geradores de código, o *front-end* e o *back-end*. O gerador de código constituído pela ferramenta, utiliza como entrada a descrição de máquina que especifica a arquitetura do processador alvo. O *front-end* utilizado no sistema MARION é o mesmo do LCC. Como mencionado anteriormente esse *front-end* gera uma representação intermediária que será aplicada como entrada para o *back-end*.

MARION impõe três restrições quanto a escolha da arquitetura alvo. Primeira, os registradores de números inteiros, o endereçamento, e as operações com ponto flutuante, devem ser de propósito geral. Segunda, as unidades funcionais e os estágios de pipeline utilizados em cada ciclo devem ser conhecidos em tempo de compilação. Terceira, somente as operações *load* e *store* devem ter acesso a memória. As arquiteturas RISC geralmente obedecem estes requisitos.

Diferentemente da proposta *toolchain* MRISC16T, o sistema MARION não apresenta ferramentas de análise de consumo de energia despendido pela aplicação. A Tabela - 2.1 contém os recursos disponíveis nos *toolchains* MIMOLA, LCC, SPAM, COSY, LANCE, MARION e MRISC16T.

Tabela 2.1: Recursos dos *Toolchains*

Recursos	MIMOLA	LCC	SPAM	COSY	LANCE	MARION	MRISC16t
Linguagem	Mimola	C	C	C C++	C	C	GCC
Histograma	Não	Não	Não	Não	Não	Não	Sim
Nível de Simulação	Aplicação	Instrução	Não	Não	Não	Não	Instrução
Profiles	Não	Função	Não	Não	Fluxo de Dados	Não	Função
Estimativa	Não	Não	Não	Não	Não	Não	Sim
Código Otimizado	Sim	Sim	Sim	Sim	Sim	Sim	Sim

2.2 SIMULADORES

2.2.1 Power Tossim

`Power Tossim` (Perla et al., 2008; Shnayder et al., 2004) é uma ferramenta de simulação de RSSF que utilizam como nós a plataforma de hardware Mica2 (Gorlatova et al., 2010). Essa ferramenta simula uma RSSF considerando os gastos de energia em cada nó. Isto caracteriza um avanço quando comparado com o simulador `TOSSIM` (Levis et al., 2003) (Asikainen et al., 2009), que simula uma RSSF levando em conta apenas o comportamento da rede.

O consumo de energia é mensurado por meio de simuladores existentes para cada dispositivo (rádio, processador, sensores e outros periféricos) do nó sensor. Estes simuladores são chamados de *micro-benchmarks* e funcionam de maneira independente.

`Power Tossim` é uma extensão do `TOSSIM`, que é uma ferramenta de simulação para RSSFs, que estima o consumo de energia da rede para `Mica2` motes. Uma nova versão desta ferramenta fornece uma simulação para MicaZ (Crossbow, 2011) *motes*. O `Power Tossim` simula uma RSSF considerando o custo de energia em cada nó, no entanto não fornece informações sobre a lógica da aplicação. Portanto o `Power Tossim` não executa programas em nível de instrução, sendo esta a sua maior limitação. Por outro lado, a ferramenta R16-EPROF efetivamente executa o aplicativo sendo capaz de estimar o consumo de energia gasto no processamento despendido por uma aplicação.

2.2.2 Avrora

O `AVRORA` (de Paz Alberola e Pesch, 2008; Titzer et al., 2005) é uma ferramenta escrita na linguagem de programação JAVA (Deitel e Deitel, 2008). Sendo assim, para melhor utilizar os recursos da linguagem a simulação ocorre com um alto nível de abstração, pois cada componente utilizado na simulação é representado por objetos. A simulação de uma aplicação retorna uma estimativa do consumo de energia, e a expectativa do tempo de vida para uma RSSF por meio de um *profile* apresentado no final da execução de uma simulação.

O ambiente de simulação do `AVRORA` pode simular vários nós de uma RSSF, e cada um desses nós são representados por uma *thread* de execução. Assim, o número de nós é

limitado pelo AVRORA de acordo com o número máximo de *threads* que podem ser alocadas para cada processo no sistema operacional hospedeiro. O fluxo de execução são sincronizados periodicamente por um relógio global que controla as rotinas de sincronização que são mantidas em uma fila.

AVRORAZ (de Paz Alberola e Pesch, 2008) é a versão do AVRORA (Titzer et al., 2005) para MICAz. Esta versão proporciona simulação em nível de ciclos de instrução, semelhante a versão anterior. O diferencial do AVRORAZ é a adição do modelo IEEE 802.15.4, possibilitando a simulação dos componentes de rádio que utilizam esse padrão.

O ambiente de simulação do AVRORAZ permite simular uma RSSF de vários nós, sendo cada um desses nós representados em uma *thread* de execução. Então o número de nós escalonáveis é limitado pelo número máximo de *threads* que podem ser alocadas para cada processo no sistema operacional. As *threads* de execução são sincronizadas periodicamente por rotinas de sincronização.

A ferramenta AVRORA fornece simulação de uma aplicação em nível de ciclos de instrução. Desta forma, esta ferramenta é capaz de estimar o consumo de energia de todos os serviços de uma RSSF. Das ferramentas citadas até este ponto, AVRORA é a mais próxima a R16-EPROF em relação aos recursos disponíveis. Porém a ferramenta R16-EPROF detalha o consumo de energia em termos de código do usuário e sistema operacional. Assim, R16-EPROF é uma ferramenta mais precisa para determinar o consumo de energia de um aplicativo específico.

2.2.3 Atemu

Atemu (Polley et al., 2004) proporciona a simulação do funcionamento de cada nó sensor Mica2 (Gorlatova et al., 2010) em nível de hardware. O simulador Atemu emula o funcionamento de vários componentes de um nó sensor, como o processador, temporizadores e dispositivos de rádio. As aplicações são simuladas em um nó sensor, no entanto os nós possuem interface de iteração proporcionando a simulação de toda a rede. Embora o Atemu apresente apenas suporte para a plataforma de *hardware* Mica2, a arquitetura é generalizada o suficiente para permitir que outras plataformas de *hardware* possam ser inseridas.

Atemu tem o objetivo de retardar o uso das plataformas de hardware reais até a etapa de implantação do sistema e ainda ser capaz de manter um bom nível de confiança. Embora este simulador realize uma descrição do consumo de energia, este não proporciona uma análise detalhada do consumo como a ferramenta R16-EPROF faz. A Tabela - 2.2 contém os recursos disponíveis nos simuladores P. Tossim, Avrora, Atemu e R16-EPROF.

Tabela 2.2: Recursos dos Simuladores

Recursos	`P. Tossim`	`Avrora`	`Atemu`	R16-EPROF
Plataforma	Mica2	Mica2, MIcaz	Mica2	UNB-RISC16
Simulação	Comportamento da Rede	Aplicação	Comportamento da Rede	Aplicação
Histograma	Não	Sim	Não	Sim
Histograma Funções	Não	Não	Não	Sim
Tempo de Vida	Não	Sim	Não	Sim

3

REFERENCIAL TEÓRICO

Nesse tópico são abordados os conceitos relacionados ao desenvolvimento do *toolchain* MRISC16T. O primeiro subtópico apresenta uma visão geral sobre RSSF como foco principal nos processadores. O segundo subtópico aborda as características que definem o processador `UNB-RISC16`, pois estes conhecimentos foram necessários para a construção do MRISC16T visando este processador. O terceiro subtópico aborda os conceitos fundamentais sobre compiladores e sobre o processo de desenvolvimento de ferramentas para geração de código para processadores por meio da utilização de compiladores redirecionáveis. As subseções seguintes abordam o compilador `GCC`, pois esse foi o compilador redirecionável portado para o processador `UNB-RISC16`.

3.1 SISTEMAS DE RSSF

RSSFs são redes cujo os nós são equipados com uma variedade de sensores, tais como acústico, sísmico, infravermelho, vídeo-câmera, calor, temperatura e pressão. Esses nós podem ser organizados em grupos *clusters*, sendo que pelo menos um dos sensores deve ser capaz de detectar um evento na região onde esta inserido (Choi et al., 2010; Fummi et al., 2012; Nabi et al., 2009). As RSSF tem a capacidade de monitorar fenômenos em diversos ambientes. O monitoramento de atividades vulcânicas, monitoramento oceânico de microorganismos, monitoramento de anfíbios, observatório de oceano, e casa inteligente são exemplos de atividades que podem ser realizadas por uma RSSF (Aquino et al., 2008; Goumopoulos e Kameas, 2009; Herlien et al., 2010). Nos tipos de ambientes mencionados anteriormente, não é possível realizar manutenção dos nós sensores de uma maneira

simples. Por isso, é muito importante que o *software* em execução reduza ao máximo o consumo de energia.

Em um nó sensor existem quatro tipos de dispositivos: sensoriamento, processamento, armazenamento e comunicação. Os dispositivos de sensoriamento são responsáveis por coletar dados de um ambiente, e podem ser ativados a qualquer momento em uma RSSF. O processamento de dados em uma RSSF é realizado por meio de microcontroladores que possuem uma UCP interna. Os microcontroladores devem ser capazes de "dormir"e "acordar"com eficiência, pois em RSSFs estas ações são adotadas entre intervalos na realização de sensoriamentos como estratégia para economizar energia. Os dispositivos de armazenamento são memórias *Ram* e *Rom* com baixa capacidade de armazenamento (na ordem de alguns Kbytes). Os dispositivos de comunicação podem utilizar dois tipo de meio de transmissão: comunicação óptica, rádio freqüência. O dispositivo RF utiliza ondas eletromagnéticas para realizar comunicação com freqüência variando de dezenas de KHz a centenas de GHz. O tamanho da antena deve ser um quarto do comprimento de onda utilizada na comunicação para economizar energia. Na comunicação óptica o transmissor utiliza raios laser para enviar informações(Choi et al., 2010; Fummi et al., 2012; Nabi et al., 2009).

A tecnologia aplicada nas RSSFs apresentaram avanços nos últimos anos. Alguns desses avanços estão relacionados a minituarização dos dispositivos de sensoriamento e processamento. O processamento de aplicações em RSSF é realizado por microcontroladores que possuem processador e dispositivos de memória integrados. Além da minituralização dos nós sensores, os microcontroladores aplicados a RSSF são desenvolvidos visando eficiência energética. Sendo assim, para realizar os objetivos mencionados os microcontroladores apresentam severas limitações quanto a quantidade de memória disponível e poder de processamento. O processador `UNB-RISC16` foi projetado para ser utilizado em RSSFs é o processador para o qual o *toolchain* MRISC16T gera código. Sendo assim, a seguir será abordado algumas de suas características.

3.2 O PROCESSADOR `UNB-RISC16`

O Grupo de Engenharia Elétrica da UNB (Universidade de Brasília) desenvolveu o processador `UNB-RISC16`, sendo este um processador RISC (*Reduced Instruction Set Computer*) (Costa, 2004; Khan et al., 2009) de 16-bits projetado para uso em sistemas embarcados ubíquos (Bell e Dourish, 2007). O processador `UNB-RISC16` opera com freqüência de 16 MHz, e seus principais componentes são: uma unidade lógica aritmética operando em ponto fixo, um banco de 16 registradores, uma memória ROM (Read Only

Memory) com 2 Kbytes (marc Masgonty et al., 2001; Vinnakota, 1995) e 8 kBytes de memória SRAM (*Static Random Access Memory*) (Cabe et al., 2010; Kim et al., 2007). A tecnologia de fabricação é a CMOS 0:35 (*Complementary Metal Oxide Semiconductor*) (Rad e Tehranipoor, 2007; Yang et al., 1999) com tensão de alimentação de 3.3V.

O processador `UNB-RISC16` (Costa, 2004) foi projetado para consumir um ciclo de processamento para cada instrução, com exceção das instruções de transferência de dados que consomem até dois ciclos. O acesso à memória pode consumir um ou dois ciclos, um para gravação de dados e dois para leitura de dados. Isto acontece porque o primeiro ciclo é dedicado para pré-carga. O conjunto de instruções é composto por dezesseis instruções, divididas em quatro categorias: aritmética, lógica, transferência, e desvios. A Tabela 2.1 exibe o conjunto de instruções do processador `UNB-RISC16`, e a Figura - 3.1 exibe o formato das instruções que é exibido na quinta coluna da Tabela 2.1.

Tabela 3.1: Instruções do processador `UNB-RISC16`

Categoria	Instrução	Opcode	Exemplo	Tipo
Aritmética	Add	0010_2	Add \$s1,\$s2,\$s3	R
	Sub	0011_2	sub \$s1,\$s2,\$s3	R
	Addi	1000_2	addi \$s1,100	J
	Shift	1001_2	Sft \$s1,8	J
Lógica	And	0100_2	And \$s1,\$s2,\$s3	R
	Or	0101_2	or \$s1,\$s2,\$s3	R
	Not	1010_2	Not \$s1	J
	Xor	0110_2	xor \$s1,\$s2,\$s3	R
	Slt	0111_2	Slt \$s1,\$s2,\$s3	R
Transferência	Lw	0000_2	lw \$s1,\$s2,\$s3	R
	Sw	0001_2	sw \$s1,\$s2,\$s3	R
	Lui	1011_2	Lui \$s1,100	J
Desvio Condicional	Beq	1100_2	beq \$s1,\$s2,5	I
	Blt	1101_2	blt \$s1,\$s2,5	I
Desvio incondicional	J	1110_2	J \$s1,100	J
	Jal	1111_2	Jal \$s1,100	J

Instruções do tipo R (Registradores) são instruções que utilizam como parâmetro somente registradores. Estas instruções utilizam quatro bits para codificação da operação, quatro bits para codificar o primeiro registrador fonte, quatro para codificar o segundo registrador fonte, e por fim mais quatro bits para codificar o registrador destino. Instruções do tipo I (Imediato) são instruções que utilizam como parâmetro dois registradores e um valor inteiro chamado de imediato. Instruções do tipo I também utilizam quatro bits para codificar as operações, quatro para codificar o registrador fonte, e quatro para

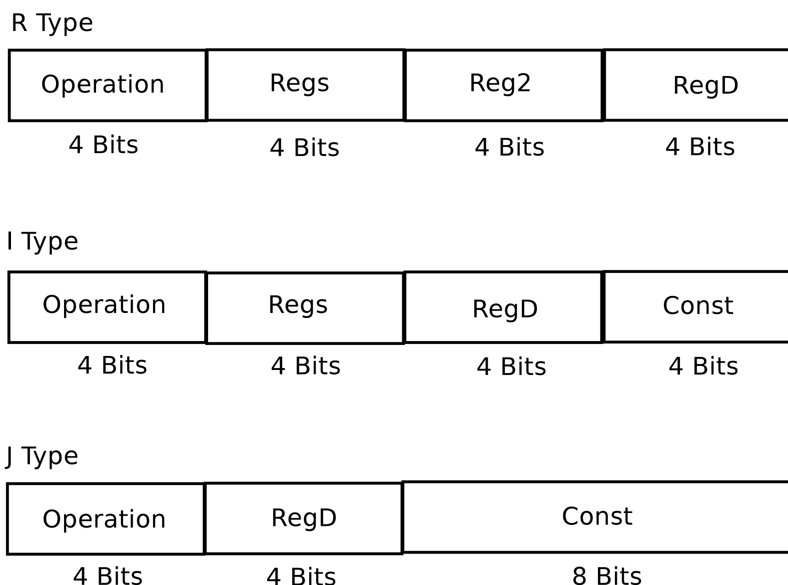

Figura 3.1: Formato das instruções do processador UNB-RISC16

codificar o registrador destino, no entanto os últimos quatro bits são dedicados para armazenar uma constante com tamanho máximo de quatro bits. Instruções do tipo J (*Jump*, salto) são instruções utilizadas para "saltar"o registrador PC (*Program Counter*) para pontos do programa especificados por um registrador e um inteiro que representa o deslocamento. Instruções do tipo J codificam as operações nos quatro primeiros bits, o registrador de destino nos quatro bits seguintes, enquanto os últimos oito bits são reservados para armazenar uma constante de oito bits (valor imediato).

A 3.2 exibe os registradores do processador `UNB-RISC16` e suas respectivas representações em código binário.

O registrador *zero* armazena sempre a constante zero. Os registradores temporários são utilizados para manter os valores resultantes de operações. Os registradores de argumento são utilizados em operações aritméticas, procedimentos e chamadas de função. Os registradores salvos armazenam valores durante a chamada de procedimentos e função. O registrador GP (*global pointer register*), é utilizado como ponteiro global. O registrador SP (*Stack Pointer*) é um ponteiro para a pilha abstrata. O registrador PC (*Program Counter*) aponta para a próxima instrução a ser executada. O registrador RA (*Return Address*) aponta para o endereço de retorno de uma subrotina.

Para atender aos requisitos de um sistema de baixo consumo de energia, área relativamente pequena e desempenho dinâmico adequado, foi proposta uma memória estática SRAM para armazenar instruções e dados necessários para o funcionamento do sistema. A capacidade de memória foi definida de acordo com o conjunto de instruções

Tabela 3.2: Banco de registradores do processador `UNB-RISC16`

Código	Símbolo	Função	Descrição
0000_2	$zero	Constante zero	Constante 0 de 16 bits
0001_2 0010_2 0011_2	$t0 $t1 $t2	Temporários	Registradores Auxiliares
0100_2 0101_2 0110_2	$a0 $a1 $a2	Argumento	Argumentos para operações aritméticas e procedimentos
0111_2 1000_2 1001_2 1010_2 1011_2	$s0 $s1 $s2 $s3 $s4	salvos	Armazena valores durante chamadas de procedimento
1100_2	$gp	Apontador global	Aponta para as variáveis globais na pilha
1101_2	$sp	Apontador pilha	Aponta para o topo da pilha
1110_2	$pc	Contador de programa	Aponta para a próxima instrução
1111_2	$ra	Endereço de Retorno	Armazena o endereço de retorno de uma rotina

e o processamento necessário para o bom funcionamento do *hardware*. O processador também possui um sistema de processamento para controle de interrupção. O sistema de interrupção é composto por três tipos de interface para comunicação (recepção de dados pela unidade de rádio, interface serial, e conversor A/D (Analógico/Digital)), e duas interfaces para manipular sinais de erro de execução como estouro da pilha, e erros de endereçamento. O subtópico seguinte apresenta os principais aspectos envolvidos na implementação do processador `UNB-RISC16`.

3.2.1 Descrição da Implementação do Processador `UNB-RISC16`

Como mencionado anteriormente, o processador `UNB-RISC16` executa instruções que pertencem a quatro categorias principais: lógica, aritmética, instruções de transferência, e instruções de desvio. Os passos necessários para executar estas instruções são semelhantes independente do tipo de suas respectivas classes. Os dois primeiros passos na execução das instruções são idênticos. Primeiro o registrador PC aponta para a próxima instrução a ser buscada na memória que contêm todo o código do aplicativo. Na segunda etapa, os registradores são associados aos campos das instruções. Para as instruções SFT, J, JAL, LUI, ADDI e NOT, apenas um registrador precisa ser acessado. No entanto, na maioria das outras instruções é necessário a leitura de dois ou três registradores.

Após estas duas etapas, a ação para completar as instruções depende de seus tipos. Para cada um dos tipos de instruções (de referência à memória (transferência), aritmética-lógica, e desvio), as ações são parecidas independentemente da instrução. Isto se deve á simplicidade e regularidade do conjunto de instruções do processador

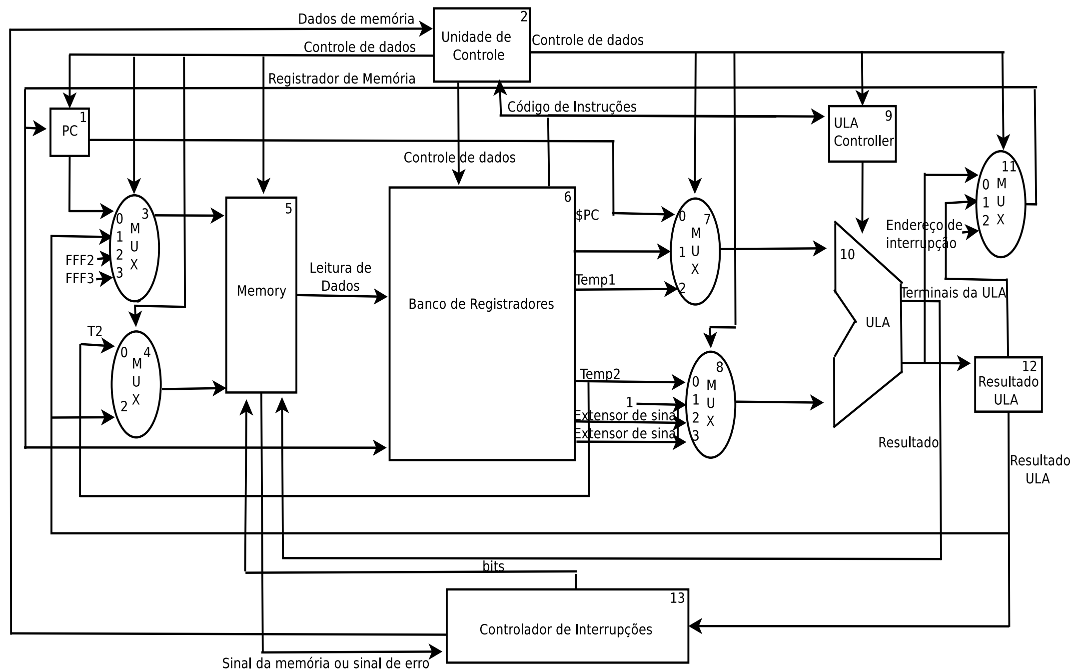

Figura 3.2: Arquitetura do processador `UNB-RISC16`

`UNB-RISC16`. Por exemplo, todos os tipos de instruções, exceto desvio incondicional, utilizam a unidade lógica aritmética (ULA) depois de ler os registradores. Para cálculo de endereços, as instruções de referência à memória utilizam a ULA. Para executar uma operação, as instruções aritméticas e lógicas, também utilizam a ULA. E, finalmente, as instruções de desvio utilizam a ULA para realizar comparação. Depois de utilizar a ULA, diferentes ações são realizadas para cada tipo de instrução. Uma instrução de referência à memória deve acessar a memória para ler os dados armazenados, ou escrever dados para armazenamento. As instruções lógicas e aritméticas devem escrever os dados na ULA a partir de um registrador. Para instruções de desvio o endereço da instrução seguinte deve ser modificado de acordo com o resultado da comparação, Caso contrário o registrador PC deverá ser incrementado apontando dois bytes a frente (ou quatro posições, uma vez que cada posição armazena um quarto da codificação de uma instrução). A Figura - 3.2 exibe o caminho de dados simplificado do processador `UNB-RISC16`.

No topo da Figura - 3.2 é exibido a unidade de controle (2) do processador `UNB-RISC16` que modifica os valores atribuídos ao registrador PC (1). A unidade de controle (2) é responsável por certificar que as instruções serão executadas corretamente. As entradas do multiplexador superior esquerdo (3) é dado pelo resultado da ULA(10) e pelos registradores mapeados na memória (0xFFF2 e 0xFFF3) juntamente com a saída PC da ULA (10), e um sinal de controle que indica quando a instrução é de desvio. No

canto inferior esquerdo da Figura o MUX(4) recebe como entrada o valor armazenado no registrador \$t2 e o resultado da ULA (12), e modifica a saída da memória (5). O multiplexador no canto superior direito (11) é utilizado para orientar a saída da ULA quando a instrução é aritmética ou lógica. Caso a instrução seja LW, a unidade ULA armazena o endereço de memória do dado para escrever em um registrador registo. O multiplexador direito no canto superior (7) modifica o registrador PC implementado no *Hardware*, o registrador PC do banco(6) e o registrador \$t1 para ser a primeira entrada da unidade ULA. O controlador da ULA (9) é um dispositivo que utilizado em operações aritméticas e lógicas do processador. O controlador de interrupção é um sistema de processamento que aceita três tipos de interrupções diferentes para as interfaces de comunicação, e duas para sinalização de erros. Finalmente, o multiplexador no canto direito inferior(8) é utilizado para determinar se a segunda entrada da ULA tem como origem dados armazenados em registradores. A segunda entrada da ULA será utilizada se esta for um campo de *offset* para instruções de *load* e *store*, ou se a instrução a ser executada ser á do tipo aritmética, lógica ou de desvio. As linhas que saem da unidade de controle representam as operações realizadas na ULA (nesse caso, estas instruções determinam se a memória de dados deve lida ou escrita).

Elementos do Caminho de Dados do Processador `UNB-RISC16`

Os elementos que constituem o caminho de dados na implementação do processador `UNB-RISC16`, possuem dois tipos de elementos lógicos: elementos que operam sobre os valores de dados, e elementos que contêm estados. Os elementos que operam sobre valores de dados são elementos combinacionais. Os elementos combinacionais são chamados assim porque dado uma entrada as operações retornam sempre uma mesma saída. Os outros tipos de elementos são os lógicos não combinacionais. Estes elementos são capazes de armazenar dados internamente, e por isso são conhecidos como elementos de estado. Os elementos de estado possuem este nome porque são capazes de armazenar estados de execução de um programa. Por exemplo, se por algum motivo na execução de uma aplicação o fornecimento de energia para o processador for interrompido, é possível reiniciar o funcionamento deste processador carregando os elementos de estado com os mesmos valores que continham antes da energia tornar se indisponível. Em outras palavras, os estados de execução de um programa podem ser salvos e restaurados possibilitando que o computador continue a execução de um programa a partir do estado que estava no momento em que a falha de energia ocorreu. Desta forma, pode se dizer que os elementos de estado caracterizam uma máquina computacional.

Um elemento de estado tem pelo menos duas entradas e uma saída. As entradas necessárias são os valores dos dados a serem gravados no elemento, e o relógio para registrar o momento que os dados foram escritos. A saída de um elemento de estado fornece os dados escritos no ciclo de *clock* anterior. Os elementos de estado exibidos na Figura - 3.2 são: o contador de programa e a memória. A memória de instrução só precisa fornecer acesso a leitura uma vez que o caminho de dados não escreve instruções, apenas as recebe de um programa e armazena. Como a memória de instruções fornece apenas operação de leitura sobre as instruções, pode se tratar estas como elementos de lógica combinacional. Isto significa que em qualquer momento da execução de um programa a saída remete ao conteúdo especificado pelo endereço de entrada, sem depender de qualquer tipo de sinal de controle que possam alterar a leitura. Em outras palavras, uma vez que o código de um aplicativo simplesmente são carregados na memória de programa, pode se utilizar as instruções contidas neste código como elementos combinacionais.

O registrador PC e a Unidade ULA

O contador de programa PC é um registrador de dezesseis bits que aponta para a próxima instrução que será executada. sendo assim, o registrador PC deve ser modificado ao final de cada instrução. Portanto, o conteúdo do registrador PC não precisa receber qualquer tipo de sinal de controle para informar quando ele deve ser escrito. Os somadores são ULAs alocadas para executar soma a partir dos registradores especificados em suas entradas gerando o resultado em sua saída. As duas unidades necessárias para implementar instruções de *LOAD* e *STORE* além da ULA são: a unidade de memória de dados e a unidade de extensão de sinal. A unidade de memória é um elemento com entradas para endereços de dados a serem escritos, e uma saída para o resultado. Existem controles separados para leitura e escrita. A unidade de memória necessita de um sinal de leitura, pois ao contrário dos registradores a leitura do valor de um endereço de memória incorreto causa erro de execução.

O processador `UNB-RISC16` utiliza a ULA na execução de instruções de desvio para avaliar a condição relacionada a instrução, e um somador separado para calcular as devidas ações relacionadas as condições de desvio. Por fim, este mesmo somador é utilizado novamente para realizar a incrementação do registrador PC. As instruções de desvio incondicional possuem um campo de oito bits para representar o deslocamento, que será somado ao valor armazenado no registrador especificado. A unidade de controle é utilizada para calcular o quanto o registrador PC deve ser incrementado. No conjunto de instruções de um processador RISC como por exemplo no `MIPS` (Gschwind e Maurer,

1996; Hennessy e Patterson, 2011; Suh e Dubois, 2009), as instruções de desvio são postergadas. Isto significa que a instrução seguinte a instrução de desvio é imediatamente executada, independentemente da condição de desvio ser verdadeira ou falsa. Quando a condição for falsa, a execução se comporta como se não existisse a instrução de desvio, e a próxima instrução é ativada normalmente. Quando a condição de uma instrução de desvio é verdadeira, o salto é atrasado e a primeira instrução a seguir é executada em ordem sequencial antes da ocorrência do salto para o endereço especificado. O atraso das instruções de desvio é necessário para evitar um comportamento incorreto do programa, uma vez que o `MIPS` (*Microprocessor with Interlocked Pipeline Stages*, ou em português Microprocessador com estágios de pipeline interligados) possui *pipelining*. No entanto o processador `UNB-RISC16` não possui *pipeline*. Desta forma, as instruções de desvio no processador `UNB-RISC16` não precisam ser postergadas em nenhum caso. Conhecendo como foi especificado e implementado o processador `UNB-RISC16`, é possível portar o *toolchain* `GCC` para gerar código *Assembly* para este processador. Para isso é necessário conhecer os conceitos de compiladores, como também, a estrutura do compilador `GCC` abordados nas próximas seções.

3.3 COMPILADORES

Um compilador é um programa de computador que traduz o código fonte escrito em uma linguagem de programação (programa de origem) em linguagem de máquina (linguagem alvo) (Aho et al., 2006). A linguagem alvo, possui um formato binário conhecido como código objeto. A razão mais comum para traduzir o código fonte é criar um programa executável.Um compilador é utilizado principalmente para traduzir código fonte de uma linguagem de programação de alto nível (por exemplo `C` (Ritchie e Kernighan, 1988), `JAVA` (Deitel e Deitel, 2008), `FORTRAN` (Chivers e Sleightholme, 2011)) para uma linguagem de baixo nível (por exemplo a linguagem *Assembly* (Ahmed et al., 2010; Hunter, 2005), ou código de máquina). Se um determinado programa compilado pode ser executado em um computador cuja UCP (Inoue et al., 2010; Kandemir e Ozturk, 2008) ou o sistema operacional (Silberschatz et al., 2008) é diferente daquele em que o compilador é executado, o compilador é conhecido como *cross-compiler* (compilador cruzado).

Os compiladores atuais (por exemplo `GCC` (Hou e Chen, 2007), `LLVM` (Zhao et al., 2012), `LCC` (Newburn et al., 2011)) realizam a maioria ou todas as seguintes operações:

- Análise léxica. É a ação de verificar a entrada do programa em linhas de caracteres e produzir uma seqüência de símbolos conhecidos como "símbolos léxicos" (*Tokens*).

- Análise sintática. É a ação de analisar a estrutura gramatical de um programa de acordo com a gramática formal de uma linguagem.
- Análise semântica. É a ação de analisar se o programa possui erros semânticos como por exemplo operações entre tipos de dados diferentes.
- Geração de código. É a ação de gerar código intermediário da aplicação.
- Otimização de código. É a ação de examinar o código intermediário produzido durante a fase de geração de código com o objetivo de executar possíveis melhorias visando melhor eficiência.

3.3.1 A Estrutura de um Compilador

Compiladores traduzem programas escritos em uma linguagem de alto nível para linguagem de máquina (*hardware*), para isso um compilador deve executar as seguintes ações:

- determinar se a sintaxe e a semântica dos programas estão corretas;
- gerar o código objeto correto de forma eficiente; e
- promover saída formatada de acordo com as convenções utilizadas pelo montador (*Assembler*) e/ou *linker* (Baldassin et al., 2008).

Um compilador é composto por três partes principais: o *front-end*, *middle-end*, e o *back-end*. O *front-end* verifica se o programa está corretamente escrito em termos da sintaxe e semântica da linguagem de programação. Caso ocorram erros, estes são relatados. A verificação de tipo também é realizada. O *front-end* então gera uma representação intermediária do código fonte para processamento pelo *middle-end*. A RI (Representação Intermediária) é independente da máquina (ou arquitetura) de destino para permitir otimizações genéricas que serão compartilhadas entre diferentes linguagens, e processadores de destino suportados pelo compilador (ver compiladores redirecionáveis).

O *middle-end* é a camada na qual a otimização ocorre. Transformações típicas para otimização são: a remoção de códigos inúteis ou inacessíveis, a descoberta e propagação de valores constantes, a realocação de computação de trechos menos executados, ou alocação de computação com base no contexto. O *middle-end* gera outra RI para o *back-end*. A maioria dos esforços para otimização estão focados no *middle-end*.

O *back-end* é responsável por traduzir a RI gerada pelo *front-end* em código de montagem, como também otimizar o código. As instruções da máquina alvo são escolhidas

para cada instrução escrita em RI. Os compiladores possuem mais de dois módulos, entretanto estes módulos são geralmente considerados como sendo parte do *front end* ou o *back-end*. A Figura - 3.3 apresenta os módulos *front-end, middle-end e back-end* de um compilador.

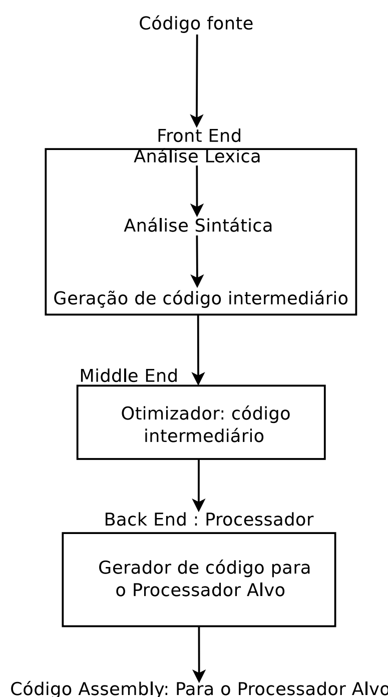

Figura 3.3: Estrutura de um compilador

Um compilador para uma linguagem relativamente simples escrita por uma pessoa pode ser um único componente de *software*. Quando a linguagem de origem é grande e complexa, e as caraterísticas do processador alvo requisitar um código ajustado, o projeto pode ser dividido em várias etapas relativamente independentes. Processador alvo é um termo utilizado como referência aos processadores destino, para o qual o compilador deverá gerar código. O uso de fases distintas de desenvolvimento possibilita que a construção de um compilador possa ser dividido em vários módulos, sendo que cada módulo possa ser escrito por desenvolvedores diferentes. Uma outra vantagem do desenvolvimento dividido em módulos, é a facilidade de substituição ou adição de novos módulos (por exemplo, otimizações adicionais).

Esta abordagem *front-end/middle-end/back-end* torna possível combinar *front-ends* para linguagens diferentes, e *back ends* para para CPUs diferentes. Exemplos práticos desta abordagem são o `GNU Compiler Collection` (Chiplunkar et al., 2011), `LLVM` (Zhao et al., 2012), que possuem vários *front-ends* e mantém estruturas de análises partilhadas

e múltiplos *back-ends*. Estes compiladores são conhecidos como compiladores redirecionáveis.

3.3.2 Compiladores Redirecionáveis

Existem compiladores que produzem código *assembly* para um único processador alvo. Se houver a necessidade de alterar a descrição do processador alvo e o compilador utilizado pelo desenvolvedor da aplicação não oferecer suporte a esta nova arquitetura, o compilador deverá ser substituído. O problema em adotar esta prática é que se caso o compilador for parte de um sistema de desenvolvimento que possua outros componentes como ferramentas de análise e montador, ocorrerá problemas de compatibilidade.

Um compilador redirecionável pode conter em sua estrutura vários *back-ends*. Esta característica torna o compilador capaz de gerar código para diferentes arquiteturas de processador sem modificar sua estrutura, ou a estrutura de um sistema ao qual faça parte (Leupers e Marwedel, 1998). Além disso um compilador redirecionável é passível de customização na descrição de um processador. Sendo assim o desenvolvedor pode adicionar uma nova descrição de processador ao compilador, possibilitando com isso a geração de código para esta nova arquitetura. Os compiladores redirecionáveis geralmente possibilitam compilação cruzada.

Compilação cruzada é uma técnica de compilação utilizada em compiladores redirecionáveis. Compilação cruzada não é um assunto trivial em computação, pois o processador alvo tem uma influência acentuada no comportamento do compilador. Esta é uma questão importante que faz com que o conceito de compilação cruzada seja uma tema complexo. No caso geral a técnica de compilação cruzada é conhecida como *Canadian cross* (Chiplunkar et al., 2011). Neste tipo de sistema, os recursos de compilação são fornecidos quando o código do compilador é de fato compilado. A descrição da máquina alvo é utilizada para construir o *back-end* do compilador cruzado. Os códigos binários do compilador gerado, são hospedados no sistema operacional atual para instalação e posteriormente execução.

Desta forma a construção do compilador gera um sistema de compilação em cruz, pois apenas o código binário do *back end* da arquitetura escolhida (que não é a mesma da máquina hospedeira) será gerado para constituir o compilador. Um compilador cruzado pode ser redirecionável tanto para um sistema operacional quanto para a plataforma de destino. Pois tanto o sistema operacional quanto a plataforma de processador podem ser diferentes dos utilizados na construção da aplicação. Neste caso as informações do

processador alvo são replicados para o sistema operacional utilizado para desenvolvimento e para hospedagem da aplicação.

3.4 GCC: GNU COMPILER COLLECTION

O GCC (Gnu Compiler Collection) (Gough e Stallman, 2005) (Chiplunkar et al., 2011) é um *toolchain* que envolve um conjunto de compiladores redirecionáveis para as linguagens de programação C, C++, Fortram, ADA, JAVA, Objective-C e Treelang. Essa ferramenta é um software livre distribuído pela *Free Software Foundation* (Brown, 2005); e foi originalmente escrito por Richard Stallman o fundador do projeto GNU iniciado em 1984. Esse projeto tinha como objetivo a criação de um sistema operacional livre baseado na plataforma Unix. No entanto para desenvolver um sistema operacional baseado nessa plataforma, era necessário um compilador para a linguagem de programação C. Como não existia naquela época nenhum compilador livre disponível para a linguagem C, a solução adotada pela *Free Software Foundation* foi desenvolver um. O trabalho foi financiado por meio de doações realizadas por pessoas físicas e empresas. A primeira versão do GCC foi desenvolvida em 1987 proporcionando um avanço na tecnologia de compiladores da época. Isso porque era o primeiro compilador portátil e livre da linguagem C.

Desde seu lançamento pela *Free software Foundation*, o GCC é uma das principais ferramentas desenvolvidas pela organização. Inicialmente o GCC era sigla para *GNU C Compiler*, pois suportava apenas a linguagem C. Com o passar do tempo novas linguagens foram adicionadas a ferramenta e a mesma passou a ter o nome *GNU Compiler Collection.* A arquitetura do GCC possui uma estrutura que permite tanto a adição de novas linguagem de programação quanto a adição de novos processadores alvo.

3.4.1 A Arquitetura do GCC

O GCC possui uma arquitetura modular que permite a adição de novas linguagens de programação a seu *front-end*, e a descrição de novas arquiteturas de processador ao *back-end*. Ao especificar uma nova linguagem de programação ao *front-end* do GCC é possível gerar código *Assembly* para qualquer arquitetura de processador suportada pela ferramenta, a partir de um código fonte escrito nesta nova linguagem. Da mesma forma, ao especificar uma nova arquitetura de processador ao *back-end* do GCC, é possível utilizar qualquer linguagem de programação descrita na ferramenta para gerar código *Assembly* para esta nova arquitetura.

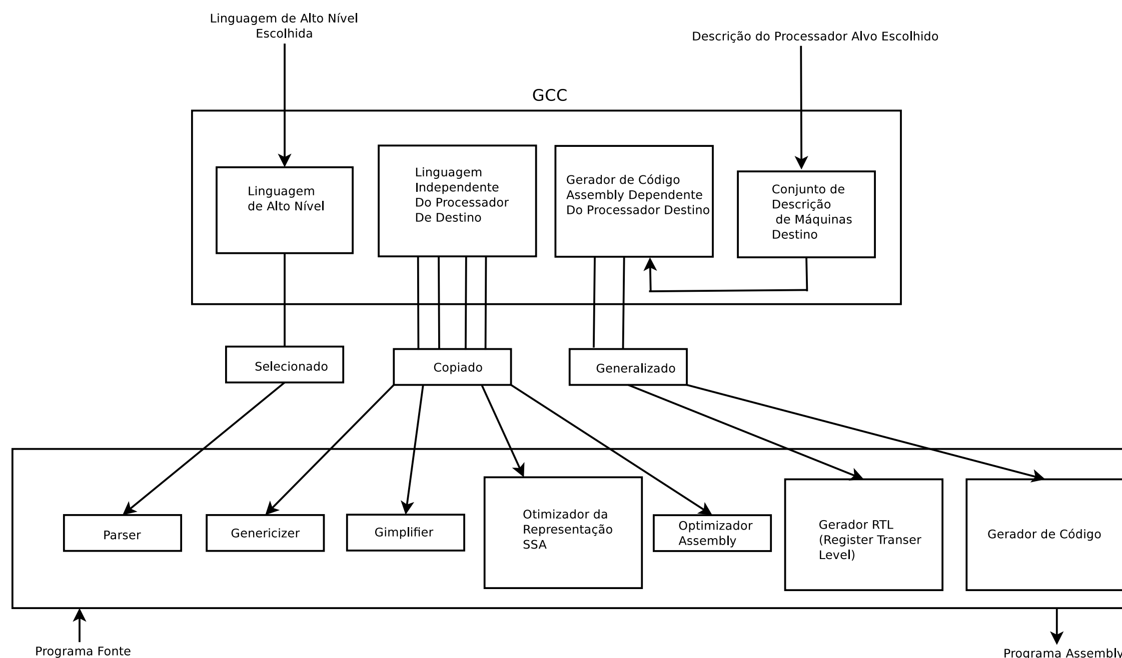

Figura 3.4: Arquitetura do GCC

No compilador GCC a representação intermediária recebe o nome de SSA (*Static Single Assignment*, atribuição única e estática). A Figura - 3.4 mostra que uma mesma representação SSA é aplicável para qualquer processador descrito no conjunto de arquiteturas do GCC, pois a representação SSA é descrita em uma linguagem independente do processador de destino. A linguagem de programação selecionada é vinculada ao *parser* que é o analisador sintático, e o parser utiliza as regras formais desta linguagem para realizar análise sintática. A representação de uma aplicação em linguagem independente de processador é copiada para o `gimplifier`, `genericizer`, para o otimizador de SSA e para o otimizador de código `Assembly`. O `gimplifier` gera uma representação genérica de três endereços (Aho et al., 2006) chamada GIMPLE a partir da representação SSA. O `genericizer` converte a representação intermediária SSA em uma representação genérica caso seja necessário. A partir deste ponto os otimizadores de código intermediário SSA e *Assembly* são ativados. O gerador de código `Assembly` e o gerador de representação RTL *Register Transfer Language*, são generalizados de acordo com a descrição do processador alvo escolhido. Neste contexto, o termo generalizar está associado ao fato dos códigos binários do *back-end* do GCC específicos de um processador destino serem fixados no momento da compilação do próprio GCC.

Uma característica importante do GCC é que este compilador pode ser utilizado para construção de um compilador cruzado (Yang et al., 2010). Isto significa que utilizando o GCC é possível desenvolver código de montagem para processadores remotos. O conjunto de

arquiteturas suportadas pelo GCC são: Alpha, ARM, H8/300, System/370, System/390, x86, x86-64, IA-64 "Itanium", Motorola 68000, Motorola 88000, MIPS, PA-RISC, PDP-11, PowerPC, SuperH, PARC, VAX.

Uma grande vantagem em utilizar o GCC para desenvolver código *Assembly*, é que essa ferramenta apresenta diferentes níveis de otimização para melhorar a qualidade do código final. Os diferentes níveis de otimizações utilizados pelo GCC são os seguintes:

- O1. Sem a otimização O1 o objetivo do compilador será o de reduzir o custo de compilação e depuração na produção de código, Com O1 o compilador tenta reduzir o tamanho do código e o tempo de execução sem executar qualquer otimizações que levam uma grande quantidade de tempo de compilação.

- O2. Com esta opção o GCC executa otimização no tamanho de código e otimização de desempenho de execução, porém não realiza *speed-Time Trade Off*, que é uma técnica que visa a economia de memória em troca de uma execução de programa mais lenta. O compilador também não executa *unrolling loop* (para otimização de loop). Em comparação com O1, a otimização O2 aumenta o tempo de compilação, porém melhora o desempenho do código gerado.

- O3. Este nível liga todas as otimizações especificadas por O2 e também ativa as opções *-finline-functions* e *-frename-registers* com o objetivo de melhorar o desempenho obtido utilizando O2.

- Os. Este nível habilita todas as otimizações de O2 que normalmente não aumentam o tamanho do código. Esta opção também realiza otimizações adicionais projetadas para reduzir o tamanho do código.

Os conhecimentos necessários para adicionar o processador UNB-RISC16 a ferramenta GCC, são abordados a seguir.

3.4.2 Portando O GCC

A estrutura do GCC apresenta três importantes períodos de tempo definidos para melhor identificar os principais elementos do *back end*. Estes períodos possibilitam uma compreensão sobre como e quando deve ser especificado uma nova arquitetura de processador para o GCC, e os efeitos que esta nova especificação causam na estrutura do compilador. Os elementos que constituem o *back end* do GCC são desenvolvidos e aplicados em etapas bem definidas. Estas etapas são descritas a seguir:

- O desenvolvimento do fluxo gráfico de controle, denotado pelo tempo de desenvolvimento;
- A construção do compilador para geração de código alvo específico, denotado por tempo de constução; e
- E o uso do compilador construído para compilar um programa, denotado por tempo de execução.

A descrição de máquina alvo (ou processador alvo) deve ser aplicada no tempo de desenvolvimento. A nova arquitetura é especificada em um arquivo de descrição de máquina escrito em RTL (Chiplunkar et al., 2011), que é uma linguagem específica com base na linguagem de programação `Scheme` (Welsh et al., 2002). A nova arquitetura deve ser incorporada no compilador `GCC` em tempo de construção e utilizado em tempo de execução.

Para inserir uma nova arquitetura de máquina no tempo de desenvolvimento é necessário conhecer o comportamento do compilador no momento de sua compilação, e no momento de sua execução. No tempo de construção o compilador `GCC` gera a representação da nova arquitetura antes que o desenvolvedor utilize o compilador. No tempo de execução o `GCC` gera uma segunda representação da arquitetura alvo, que descreve os conceitos específicos do funcionamento da nova arquitetura para geração de um código *Assembly* mais ajustado. No tempo de execução a descrição da seqüência de operações definidas no tempo desenvolvimento é necessária na compilação de um programa para direcionar a montagem do programa desenvolvido.

A descrição RTL é utilizada no `GCC` para especificar as propriedades da máquina alvo em tempo de desenvolvimento, gerando assim a representação da entrada a ser compilada em tempo de execução. Os operadores da linguagem RTL geram expressões chamadas de RTXs para atender a semântica das instruções da nova arquitetura alvo. A representação RTX é uma representação gerada para utilização dentro do próprio `GCC`. No tempo de desenvolvimento, a representação RTXs utilizam a descrição de máquina na linguagem RTL para obter diferentes tipos de especificações, como por exemplo formatos das instruções, nomes de registradores, e tipos de dados.

O processador em linguagem RTL é caracterizado por um conjunto de registradores, um conjunto de instruções e informações sobre os tipo de dados e seus respectivos tamanhos. Os dados que representam as características de um processador em RTL, são tipos de dados simples como números inteiros e *strings*. As estruturas formadas pela junção destas características são definidas como `GCC` *Internals*. O conjunto de registradores é um

exemplo de como um compilador é influenciado pela máquina alvo, pois nas especificações da máquina podem ser definidos atributos adicionais aos registradores. Estes atributos podem definir se os registradores serão dedicados a uma operação específica. Por exemplo, na arquitetura do processador `UNB-RISC16` há registradores temporários que são utilizados em operações lógicas e aritméticas, registradores que são utilizados como argumentos de função, e registradores específicos para ponteiros. Sendo assim, o comportamento do compilador muda de uma arquitetura para outra, pois a utilização do compilador deve resultar em instruções que utilizem os registradores corretamente de acordo com a implementação da máquina alvo.

O sistema operacional também pode modificar o funcionamento de um compilador, porém esta influência não acontece de forma direta. O sistema operacional determina como os parâmetros devem ser passados ao compilador, e qual(s) o(s) caracter(es) é(são) reservado(s) para representar o separador de caminho. O sistema operacional também determina como é realizada as chamadas de sistema (Baugh e Zilles, 2007). Isto ocorre porque as interfaces para chamada de sistema são diferentes de um sistema operacional para outro. Tais chamadas devem ser conhecidas pelo compilador, ou devem ser especificadas em arquivos de cabeçalho pelo próprio sistema operacional. Estes arquivos de cabeçalho são necessários para que o `GCC` possa gerar código `Assembly` corretamente, especialmente quando ele é utilizado como um compilador cruzado (Yang et al., 2010).

Em resumo, um compilador é um dos instrumentos necessários para traduzir completamente um programa em um processo (Sebesta, 2009) de um sistema operacional. Uma vez que o compilador `GCC` é parte do MRISC16T ele deve gerar código compatível com as outras ferramentas inseridas no *toolchain*. Desta forma, o compilador deve conhecer a estrutura sintática da linguagem *Assembly* da arquitetura `UNB-RISC16`.

4

O TOOLCHAIN MRISC16t

O *toolchain* proposto neste trabalho foi nomeado MRISC16T (Manna Reduced Instruction Set Computer Toolchain), pois está sendo desenvolvido no laboratório do grupo Manna localizado na Universidade Estadual de Maringá. O objetivo principal deste conjunto de ferramentas é fornecer um ambiente de desenvolvimento de aplicações para o processador `UNB-RISC16` escritas na linguagem de programação `C`. O *toolchain* MRISC16T possui ferramentas classificadas em dois grupos a saber: tradução e análise. As ferramentas de tradução compilam o código fonte e geram o código alvo para o processador `UNB-RISC16`. E as ferramentas de análise fornecem perfis sobre o código da aplicação e o consumo de energia. A 4 exibe as ferramentas que compõem o *toolchain* MRISC16T, juntamente com uma breve descrição de cada ferramenta.

Tabela 4.1: Ferramentas que compõem o *toolchain* MRISC16T

Grupo	**Ferramenta**	**Descrição**
Tradução	R16-AS	Assembler
	R16-CC	Compilador
	R16-CPP	Preprocessador C
	R16-LD	Linker
	R16-OBJDUMP	Lista informações de um arquivo objeto
Análise	R16-EPROF	Mostra as informações sobre o consumo de energia
	R16-GPROF	Mostra informações sobre o fluxo de execução

As ferramentas de tradução são: (r16-ld, r16-cpp, r16-cc, r16-objdump). O R16-AS é o assembler do `UNB-RISC16`, sendo sua finalidade principal montar a saída do compilador `C` para o uso do *linker* R16-LD. O compilador `C GCC` foi devidamente alterado para adequadamente gerar código de máquina para o processador `UNB-RISC16`. O pré-processador `C`, conhecido como `cpp`, é um processador de macro que é usado automaticamente pelo

compilador para transformar o programa antes da compilação. É chamado de processador de macro porque permite que o desenvolvedor defina macros, que são abreviações para construções longas. A ferramenta `cpp` é útil para gerar o código do bloco de programa para uma arquitetura de processador específica levando em conta suas particularidades. A ferramenta `ld` é um *linker* para arquivos objeto de um programa. Um arquivo passado como entrada para o `ld` é composto por uma série de arquivos objeto. A ferramenta `ld` realoca os dados e associam as referências feitas a símbolos. Normalmente, o último passo na compilação de um programa é executar o `ld`. Desta forma, o `ld` constrói um arquivo executável ou uma biblioteca.

No *toolchain* MRISC16T há três ferramentas de análise: R16-GPROF, R16-OBJDUMP, e R16-EPROF. A ferramenta R16-GPROF é um *profiler* que permite identificar blocos de programa que resultam em uma execução pouco eficiente. O *profile* gerado exibe para o desenvolvedor blocos de programa que são mais lentos do que o desejado, para que o desenvolvedor possa reescrever tais blocos e fazer o programa executar mais rápido. Além disso, apresenta a frequência de chamada de cada função. Isso auxilia na localização de erros. A ferramenta R16-OBJDUMP exibe informações sobre um ou mais arquivos objeto, tais informações são úteis para os desenvolvedores analisarem o comportamento do seu programa. A ferramenta R16-EPROF do *toolchain* MRISC16T é utilizada para análise do consumo de energia e recursos de hardware.

4.1 FERRAMENTAS DE TRADUÇÃO

Este tópico descreve as ferramentas de tradução que compõem o *toolchain* MRISC16T. As ferramentas de tradução são utilizadas para gerar os códigos objeto, e executável de uma aplicação.

4.1.1 A Ferramenta r16-as

A ferramenta R16-AS é o montador para o processador `UNB-RISC16`. Por padrão o `GAS` é utilizado como *back-end* do `GCC`. Em outras palavras se pode dizer que a ferramenta R16-AS é utilizada para construir a saída de um compilador `C`, e a entrada para o *linker* R16-LD.

O R16-AS executa a leitura do arquivo *Assembly* e o pré processamento do mesmo. Em seguida o R16-AS analisa e monta cada linha de entrada do arquivo. Por fim, o R16-LD realoca os símbolos não resolvidos durante o processamento das expressões específicas

para a arquitetura de processador utilizada. Após estes passos é gerado o arquivo objeto que contém as informações para cada símbolo, e as informações necessárias para *debug*.

4.1.2 A Ferramenta r16-ld

A ferramenta R16-LD é o *linker* para o processador UNB-RISC16. Esta ferramenta é utilizada para organizar uma série de arquivos objeto, realocando seus dados e relacionando as referências feitas a símbolos. Geralmente esta ferramenta é chamada após a compilação de um programa. É bastante comum para um vinculador criar um programa a partir de múltiplos subprogramas, e criar um programa de saída que começa a partir do endereço zero na memória (Levine, 1999). A ferramenta R16-LD recebe como entrada arquivos relocáveis no formato objeto UNB-RISC16 especificado no tópico 4, e pode gerar como saída um novo arquivo objeto relocável, ou um arquivo objeto executável.

4.1.3 A Ferramenta r16-cpp

O pré-processador R16-CPP (Baldassin et al., 2008) implementa a linguagem de macros utilizada para transformar trechos de programas C, C++ e Objective-C antes de serem compilados. O R16-CPP pode também ser chamado de processador de macros pois permite que macros sejam definidas. Estas macros são abreviações de trechos de código. O pré-processador C pode ser utilizado somente nas linguagens de programação C, C++ e Objective-C. As versões mais atuais do GNU *Assembler* possui um processador de macro implementado em sua estrutura.

4.2 FERRAMENTAS DE ANÁLISE

Este tópico descreve as ferramentas de análise disponíveis no *toolchain* MRISC16T. Estas ferramentas são utilizadas para realização de avaliações sobre estrutura de código e estimativa de energia.

4.2.1 A Ferramenta r16-objdump

O R16-OBJDUMP (Baldassin et al., 2008) mostra as informações sobre um ou mais arquivos objeto. Estas informações são úteis para os desenvolvedores de ferramentas de compilação. Este é um dos principais recursos do *toolchain* MRISC16T. Utilizando a ferramenta R16-OBJDUMP é possível visualizar as seções pertencentes a um arquivo objeto relocável do processador UNB-RISC16.

4.2.2 A Ferramenta r16-gprof

R16-GPROF (Garcia et al., 2011) é utilizada para gerar *profile* sobre a estrutura do código de uma aplicação. Um *profile* permite que ao desenvolvedor saber quanto tempo as funções de uma aplicação gastam para chamar umas as outras. Estas informações podem ajudar o desenvolvedor a avaliar os trechos de código de sua aplicação e prever os que podem causar perda de desempenho portanto, pode ajudar os desenvolvedores a identificar as partes de código que devem ser reescritas. Uma vez que R16-GPROF coleta informações em tempo de execução, é necessário que a aplicação seja compilada e executada.

4.2.3 A Ferramenta r16-eprof

Pelo fato da ferramenta R16-EPROF fornecer diversas funcionalidades e ser a principal ferramenta de análise do *toolchain* MRISC16T, foi dedicado o próximo tópico para descrever a ferramenta R16-EPROF.

4.3 FORMATO DE ARQUIVO OBJETO `UNB-RISC16`

Um formato de arquivo objeto, é o formato de arquivo com o nível lógico mais próximo do *hardware* para qualquer plataforma (Zhang, 2002). Esse formato é projetado com o o objetivo principal de fornecer códigos de máquina binários formatados para execução e vinculação de programas. O formato de arquivo objeto também fornece informações sobre dados inicializados, não inicializados e informações de depuração de programa.

Tipicamente um sistema embarcado é um sistema baseado em micro processador com um software de aplicação específica e auto-suficiente armazenado na memória ROM e sem sistema operacional prontamente reconhecível (Zhang, 2002). Sendo assim, o arquivo objeto de uma aplicação para sistemas embarcados deve conter as informações de como configurar os segmentos de texto e dados nos diferentes locais de memória. Uma outra questão que deve ser esclarecida pelo formato de arquivo objeto, é como inicializar a execução de uma aplicação a partir de um endereço de memória específico.

Seguindo este raciocínio, a principal diferença entre um arquivo executável e um arquivo objeto de uma aplicação reside na resolução dos endereços absolutos. No entanto, a especificação de um formato de arquivo objeto para sistema embarcado, e para um sistema de computador convencional, são semelhantes. Nesse contexto endereços absolutos, são os endereços de memória que os seguimentos de dados e texto de uma aplicação serão armazenados na memória principal.

Um arquivo objeto executável para um sistema de computador é sempre relocável, entretanto, um arquivo objeto executável de um sistema embarcado não é relocável. Isso significa que os endereços contidos no segmento de texto e os endereços do segmento de dados são imutáveis. Para garantir a corretude dos endereços contidos nos segmentos do arquivo objeto, após o código de uma aplicação ser traduzido em um arquivo objeto relocável, e vinculado com outros arquivos objeto, é necessário corrigir os endereços absolutos para cada segmento. A ferramenta `LD` do *toolchain* `MRISC16t` possui em sua implementação o corretor dos endereços absolutos. Na próxima seção será discutido o formato de Arquivo Objeto proposto para o processador `UNB-RISC16`.

4.3.1 Segmentos do Formato `UNB-RISC16`

A Figura - 4.1, exibe uma representação visual da organização dos segmentos presentes no formato de arquivo objeto relocável do processador `UNB-RISC16` (DAmato et al., 2012). O objeto realocável do formato `UNB-RISC16` possui os campos `Header` (cabeçalho), `ExternTable` (tabela de dados externos), `DataSection` (seção de dados), `TextSection`

(seção de texto), `RelocTable` (tabela de simbolos realocáveis), e `ObjectSymbolsTable` (tabela de símbolos).

Cabeçalho	Tabela de dados externos	Seção de Dados	Seção de Texto	Tabela de símbolos Relocáveis	Tabela de símbolos

Figura 4.1: Formato de arquivo objeto relocável do processador `UNB-RISC16`

As informações contidas nesses campos serão utilizadas pela ferramenta `r16-LD` para realizar a vinculação de diferentes módulos realocáveis de um programa. As seções de dados e texto são vetores do tipo `Unsigned Char`, sendo a seção de dados utilizada para armazenar variáveis inicializadas e não inicializadas, e a seção de texto utilizada para armazenar o código do programa. Geralmente em outros formatos, as variáveis não inicializadas são armazenadas em uma seção diferente da seção Data. No entanto, para simplificar o formato do código objeto realocável do processador `UNB-RISC16`, as variáveis não inicializadas foram armazenadas na seção `Data`.

A estrutura que representa a seção *Header* ou cabeçalho, especifíca o tamanho e o deslocamento de cada segmento de um programa. Estas informações são utilizadas pela ferramenta `LD` para realizar o cálculo dos endereços dos segmentos presentes em dois ou mais módulos de programa na etapa de vinculação. A estrutura do segmento de cabeçalho e seus respectivos campos são descritos na 4.3.1

Tabela 4.2: Campos da seção de cabeçalho do formato `UNB-RISC16`

Campo	Tipo	Descrição
Header Size	char[64]	tamanho do cabeçalho
Header Offset	unsigned short	deslocamento do segmento header
Extern Size	unsigned short	tamanho do segmento Extern
Extern Offset	unsigned short	deslocamento do segmento Extern
Data Size	unsigned short	tamanho do segmento de dados
Data Offset	unsigned short	deslocamento do segmento de dados
Text Size	unsigned short	tamanho do segmento de programa
Text Offset	unsigned short	deslocamento do segmento de programa
Symbols Table Size	unsigned short	tamanho da tabela de símbolos
Symbols Table Offset	unsigned short	deslocamento da tabela de símbolos
Symbols Table Elementnum	unsigned short	número de elementos da tabela de símbolos
Reloc Size	unsigned short	tamanho do segmento da tabela de símbolos realocáveis
Reloc Offset	unsigned short	deslocamento do segmento da tabela de símbolos realocáveis
Module Size	unsigned int	tamanho total do módulo de programa
Type	unsigned short	tipo do módulo (0: realocavél, 1: realocável com endereços absolutos)
Version	unsigned short	versão do módulo

A estrutura que representa um elemento da seção da tabela `Extern` armazena o nome dos símbolos que não pertencem ao módulo do programa corrente, e que serão procurados

nos segmentos de dados de outros módulos na etapa de vinculação. A estrutura que representa um elemento do segmento `Extern` e seus respectivos campos são descritos na Tabela - 4.3.

Tabela 4.3: Campos da seção extern do formato `UNB-RISC16`

Campo	Tipo	Descrição
Name	char[64]	nome do símbolo
Call Position Offset	unsigned short	endereço relativo do programa que o símbolo foi chamado

A estrutura que representa um elemento do segmento da tabela de símbolos realocáveis armazena o nome dos símbolos que possuem endereços relativos, e o endereço em que ocorre chamadas de cada um. Estas informações são utilizadas após a etapa de cálculo de endereçamento absoluto para correção dos endereços relacionados as chamadas dos símbolos. A estrutura que representa um elemento do segmento da tabela de símbolos realocáveis e seus respectivos campos são descritos na Tabela - 4.4.

Tabela 4.4: Campos da seção de símbolos realocáveis do formato `UNB-RISC16`

Campo	Tipo	Descrição
Name	char[64]	nome do símbolo
Call Position Offset	unsigned short	endereço relativo do programa que o símbolo foi chamado

A estrutura que representa um elemento da tabela de símbolos armazena o endereço de um símbolo, a seção que o símbolo pertence, o tamanho do símbolo e específica se o endereço associado a ele é relativo ou absoluto. A estrutura que representa um elemento do segmento da tabela de símbolos e seus respectivos campos são descritos na Tabela - 4.5.

Tabela 4.5: Campos da tabela de símbolos do formato `UNB-RISC16`

Campo	Tipo	Descrição
Name	char[64]	nome do símbolo
Value	unsigned short	endereço do símbolo
Section	unsigned short	segmento que o símbolo pertence, 1: extern; 2: data; 3: text
Size	unsigned short	tamanho do símbolo
Rel	unsigned short	tipo do símbolo (0: realocavél, 1: com endereço absoluto)

Na próxima seção será discutida as restrições encontradas no processador `UNB-RISC16` que afetam diretamente a geração de código das aplicações para esse processador. Para cada restrição será discutida as soluções encontradas para lidar com as mesmas. As soluções encontradas foram implementadas no compilador `GCC`.

4.4 RESTRIÇÕES DO PROCESSADOR UNB-RISC16 PARA GERAÇÃO DE CÓDIGO

Gerar código para processadores utilizados em sistemas embarcados não é uma tarefa trivial, pois as restrições existentes nesse tipo de *hardware* também impacta no código resultante. Os reduzidos conjuntos de instruções e registradores do processador UNB-RISC16 juntamente com o tamanho fixo do formato das instruções, que é de dezesseis bits, resultam em severas restrições na geração de código. Um outro fator que restringe a geração de código para o processador é o endereçamento e referências á memória, que acontecem exclusivamente de maneira indireta por meio da utilização de registradores. As soluções abordadas neste tópico foram utilizadas para solucionar as restrições para geração de código existentes no processador UNB-RISC16, foram implementadas no *back-end* do compilador GCC com o objetivo de manter a expressividade da linguagem de programação C.

O reduzido conjunto de instruções do processador UNB-RISC16 é uma consequência da utilização da arquitetura RISC (Costa, 2004), no entanto a utilização dessa arquitetura fornece vantagens em relação a arquitetura CISC (Hennessy e Patterson, 2011). A arquitetura CISC (*Complex Instruction Set Computing*), utilizada nos processadores processadores Intel e AMD suporta mais instruções, no entanto mais lenta é a execução delas.

O tamanho das instruções de apenas dezesseis bits, impossibilita atribuições de valores maiores do que oito bits em uma única instrução em registradores e memória. Isto ocorre porque somente para codificar uma instrução é necessário oito bits, restando apenas mais oito bits para representar o valor a ser atribuído. A geração de código para o processador UNB-RISC16 para instruções que manipulam valores maiores que oito bits ocorre em duas etapas, ou seja, primeiro é carregado no registrador destino a parte alta do valor, e seguida a parte baixa. Por exemplo:

```
Lui $t0,hi(0xf4fe)
Sft $t0,-8
Lui $t1,lo(0xf4fe)
Or $t0,$t0,$t1
```

No exemplo o valor 0xf4fe deve ser atribuido ao registrador \$t0, porém o valor 0xf4fe não pode ser atribuido em uma única instrução, visto que, o mesmo é maior do que oito bits. Sendo assim, a solução é carregar a parte alta do valor em \$t0 e realizar deslocamento

para esquerda em oito bits deixando o valor parcial 0xf4 nos bits mais altos de $t0. O registrado $t1 recebe a parte baixa do valor 0xf4fe, ou seja, o conteúdo de $t1 fica sendo 0xfe. Por último os valores contidos em $t0 e $t1 são alocados em $t0 por meio da operação lógica Or. Desta forma, o registrador $t0 recebe o valor 0xf4fe.

No processador `UNB-RISC16` o endereçamento para acesso a memória acontece sempre de maneira indireta por meio de registradores. No modo de endereçamento indireto um endereço de memória a ser acessado sempre estará armazenado em um registrador. Por exemplo: No exemplo, a instrução carrega o conteúdo da palavra na posição de memória

```
Lw $t0,$t1,$zero
```

cujo endereço está no registrador $t1, para o registrador $t0 com o deslocamento zero. Os endereçamentos tanto no acesso a escrita, quanto leitura, são realizados na memória dessa forma no processador `UNB-RISC16`.

4.4.1 Restrições Relativa ao Conjunto de Instruções

O conjunto de apenas dezesseis instruções do processador `UNB-RISC16` resulta na ausência de instruções importantes pertencentes as categorias aritméticas, transferência e desvio condicional. Instruções de manipulação de pilha também não estão presentes no processador `UNB-RISC16`, visto que este processador não possui em sua definição formal uma pilha explícita. No entanto, as instruções disponíveis são suficientes para implementar no compilador as instruções que não estão presentes no processador `UNB-RISC16`. A seguir será discutida a abordagem utilizada para tratar as limitações do conjunto de instruções do processador `UNB-RISC16` por categoria de instruções.

Transferência

As instruções de transferência que não compõem o conjunto de instruções do processador `UNB-RISC16` e as respectivas estratégias utilizadas para suprir essas instruções são as seguintes:

Mov Registrador_Destino, Registrador_Fonte: essa instrução foi implementada no compilador utilizando as instruções Add Registrador_Destino, Registrador_fonte,$zero. Por exemplo:

```
Add $t0,$t1,$zero
```

A instrução Add soma o conteúdo do registrador $t1 com o registrador $zero, que tem como conteúdo sempre o valor zero. Desta forma, o Registrador $t0 recebe o conteúdo do Registrador $t1.

Mov Registrador, Imediato: para produzir o mesmo efeito dessa instrução é possivel utilizar a instrução Lui do processador `UNB-RISC16` que atribui o valor imediato ao registrador destino. Por exemplo:

```
Lui $t0,100
```

No exemplo o valor 100 é carregado no registrador $t0 diretamente.

Mov registrador, memória: o processador `UNB-RISC16` possui duas instruções de acesso a memória sendo elas: Lw e Sw. A primeira atribui ao registrador de destino o conteúdo do endereço de memória especificado no registrador fonte com o deslocamento especificado em um terceiro registrador. Portanto se a utilizarmos o registrador $zero para representar o deslocamento a instrução Lw pode ser utilizada para implementar o Mov cujo os parâmetros são um registrador e um endereço de memória. Por exemplo:

```
Lw $t0,$t1,$t2
```

No exemplo o registrador $t0 recebe o conteúdo do endereço de memória contido no registrador $t1 com o com o deslocamento contido no registrador $t2.

Aritmética

As instruções aritméticas que não compõem o conjunto de instruções do `UNB-RISC16` e as respectivas estratégias utilizadas para suprir essas instruções são apresentadas a seguir:

Mult: as instruções de multiplicação não estão presentes no conjunto de instruções do processador `UNB-RISC16`. Para lidar com esta restrição foi implementado a instrução Mult no compilador utilizando sucessivas adições por meio da instrução Addi, e as instruções Blt e J para gerar um laço de repetição. Por exemplo:

```
xor  $t0,$t0,$t0
beq  $t1,$zero,4
addi $t1,-1
add  $t0,$t0,$t2
blt  $zero,$t1,-6
add  $t2,$t0,$zero
```

No exemplo acima deseja-se multiplicar o conteúdo do registrador $t2, com o conteúdo em $t1 e atribuir o resultado a $t2.

Div: as instruções de divisão também não estão presentes no processador `UNB-RISC16`, e foi implementada no compilador utilizando sucessivas subtrações. A estratégia foi subtrair do dividendo o valor do divisor o quanto possível.

```
xor  $t0,$t0,$t0
beq  $t2,$zero,6
sub  $t2,$t2,$t1
addi $t0,1
blt  $zero,$t2,-6
add  $t2,$t0,$zero
```

No exemplo acima deseja-se dividir o conteúdo do registrador $t2, pelo o conteúdo em $t1 e atribuir o resultado a $t2.

Desvio Condicional

Estão presentes no processador `UNB-RISC16` apenas as instruções Blt (menor) que representa a função de comparação menor, e Beq (iqual) que representa a função de comparação iqual. Entretanto estas funções são suficientes para implementação das instruções de desvio Ble (menor iqual), Bz (iqual a zero), Bge (maior ou iqual), Bgt (maior), que não estão presentes no processador `UNB-RISC16`.

A instrução Ble foi implementada utilizando as instruções Blt e a instrução Beq, a instrução Bz foi implementada utilizando a instrução Beq e o registrador $zero. A instrução Bge foi implementada utilizando as instruções Blt e Beq. Por exemplo (Bge):

```
Blt $t0,$t1,2
Beq $zero,$zero,(valor do salto)
```

Manipulação de pilha

No processador `UNB-RISC16` não há manipulação da pilha implicíta, no entanto, há o registrador $sp, que é o apontador para pilha. Utilizando o registrador $sp e as instruções Lw e Sw é possível implementar as instruções de empilhar e desempilhar (Pop e Push). A instrução Pop (empilha) foi implementada no compilador utilizando as instruções Addi para incrementar o apontador $sp que aponta pro topo da pilha, então a instrução Sw para armazenar o novo valor. Por exemplo:

```
addi $sp,-2
Sw   $t0,$sp,$zero
```

O \$t0 contém o valor a ser armazenado no topo da pilha, \$sp é o endereço do topo da pilha, e o \$zero indica que o valor em \$t0 será armazenado no endereço em \$sp sem deslocamento. A instrução Push (desempilhar) foi implementada no compilador utilizando as instruções Lw para carregar em um registrador o valor armazenado no topo da pilha, e addi para decrementar o valor do topo da pilha. Por exemplo:

```
Lw   $t0,$sp,$zero
Addi $sp,2
```

No exemplo, o registrador \$sp aponta para o topo da pilha, o valor armazenado no topo da pilha é atribuido ao registrador \$t0, e então o valor do apontador do topo da pilha retrocede dois Bytes indicado pelo valor inteiro dois. Tanto para as instruções Pop quanto para a instrução Push é importante lembrar que o endereço inicial da pilha é 65535. Isto significa que ao somar valores positivos ao registrador \$sp estamos na verdade acessando valores contidos no interior da pilha, ou seja, estamos decrementando o topo da pilha. As restrições e soluções discutidas neste tópico encerra o assunto sobre geração de código no *toolchain* MRISC16T, a seguir será abordado os recursos presentes no mesmo.

5

A FERRAMENTA R16-EPROF

Este tópico aborda o modelo utilizado na implementação do R16-EPROF para estimar o consumo de energia e descreve todos os recursos disponíveis na ferramenta. R16-EPROF estima o consumo de energia de uma aplicação, quando executada pelo processador UNB-RISC16. Esta ferramenta pode ser executada em dois modos: estático e dinâmico. O modo estático executa o aplicação e estima o consumo de energia. Por outro lado, o modo dinâmico permite ao desenvolvedor fornecer o contexto de execução da aplicação. Este contexto é expresso por um arquivo de configuração que informa como e quando a aplicação é invocada, e a quantidade de energia disponível. Assim, é possível estimar com mais precisão o consumo de energia despendido, e determinar o tempo de vida de um nó de acordo com a quantidade de processamento e os tipos de interrupções que ocorreram.

5.1 A ARQUITETURA DA FERRAMENTA r16-eprof

A Figura - 5.1 exibe a arquitetura da ferramenta R16-EPROF, e todas as possíveis entradas e saídas do sistema. Como entrada a ferramenta R16-EPROF pode receber o código de um programa em representação hexadecimal escrito no terminal, ou arquivo objeto de um programa, ou ainda um arquivo que contém a saída da ferramenta R16-OBJDUMP. Como saídas, R16-EPROF gera arquivos de análise da simulação de um programa, ou apresenta estas análises no terminal. O núcleo da ferramenta R16-EPROF é composto por um simulador em nível de instrução e uma fila de interrupções para armazenar interrupções programadas no modo dinâmico. As interfaces de interrupção se comunicam com o núcleo do sistema R16-EPROF para realizar simulação dos dispositivos alocados em cada interface (comunicação, conversor A/D, serial).

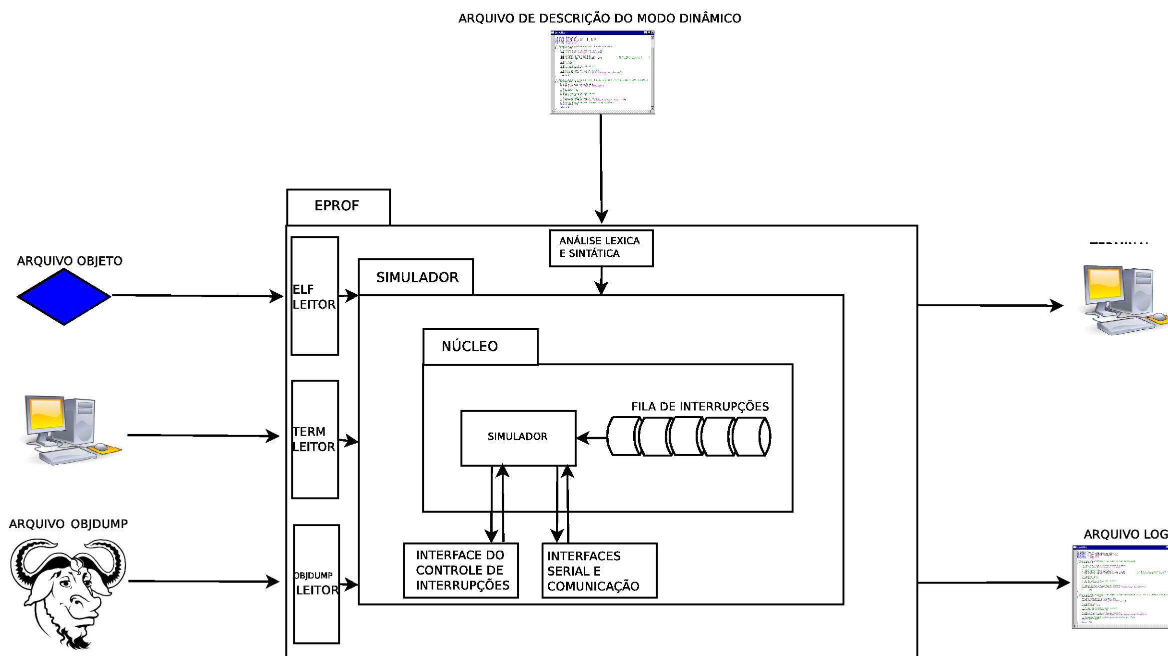

Figura 5.1: Entradas e saídas do R16-EPROF

A fim de simular a execução da aplicação, uma fila de interrupções é utilizada. A fila de interrupções mantém as interrupções ocorridas na execução de uma aplicação no modo dinâmico. Esta estrutura de dados permite emular com mais confiança o funcionamento do processador `UNB-RISC16`. A motivação para o uso da fila de interrupções no simulador é o ganho de desempenho na simulação. Isso ocorre porque as aplicações em nós sensores tendem a entrar em modo *sleep* durante uma grande parcela de tempo. Os nós devem utilizar algoritmos que explorem ao máximo o modo *sleep* quando possível. Isto possibilitará que o processador `UNB-RISC16` gaste menos energia. Em RSSFs, existe a necessidade de um baixo consumo de energia, pois essas redes são implantadas geralmente em áreas remotas onde as baterias não podem ser substituídas ou recarregadas frequentemente. Consequentemente, os programas tendem entrar em modo *sleep* a maior quantidade de tempo possível, acordando para coletar dados do ambiente e se comunicar. Este intervalo pode variar de algumas vezes por segundo para algumas vezes por hora.

O fato é que os nós ativam o modo *sleep* para economizar energia, e isso é simulado na ferramenta R16-EPROF pela fila de interrupções. Se a aplicação coloca o processador `UNB-RISC16` em modo *sleep*, então apenas uma interrupção pode acordar o processador. Tal evento pode ser gerado por simulação no chip ou no ambiente (arquivo de configuração do modo dinâmico). Estas interrupções são executadas pelo simulador em um momento apropriado no futuro. Conseqüentemente, as interrupções devem ser colocadas na fila

antes do tempo que devem ocorrer. Somente uma interrupção ocorrida anteriormente ao processamento da fila pode influenciar a simulação quando o UNB-RISC16 está dormindo, uma vez que não há instruções sendo executadas. Portanto a ferramenta R16-EPROF só precisa processar as instruções de uma aplicação em ordem, até que uma delas faça com que uma interrupção de *hardware* ocorra. Após o processamento destas interrupções, continua a execução normal das instruções pelo processador.

5.2 O MODELO UTILIZADO PARA ESTIMAR O CONSUMO DE ENERGIA

Uma maneira intuitiva de estimar a energia dissipada dinamicamente pelo processador é por meio da utilização de ciclos de *clock* (Hennessy e Patterson, 2011), que determina quando os eventos ocorreram no *hardware*. É conveniente pensar sobre o desempenho em ciclos de *clock*, que é o intervalo de tempo para a ocorrência de um pulso de *clock*, sendo que este ocorre em um ritmo constante. De fato, a quantidade de energia dissipada pelo processador é proporcional à quantidade de ciclos utilizado durante o processamento de uma tarefa específica.

O desempenho do processador pode ser estimado pela quantidade total de ciclos de *clock* ocorridos durante a execução:

$$Ciclos = Ciclos_{efetivos} + Ciclos_{nao_efetivos} \tag{5.1}$$

O desempenho global é baseado na quantidade de ciclos que a unidade de processamento central (UCP) (Hennessy e Patterson, 2011; Stallings, 2010; Tanenbaum, 2006) gasta no processamento útil e com o processamento em modo *sleep* (não efetivo). Processamento útil indica que o processador é utilizado em processos de usuário ou do sistema operacional (Tanenbaum, 2008) (Silberschatz et al., 2008). O modo *sleep* pode ocorrer basicamente em três situações: quando a fila de interrupções está vazia (Hennessy e Patterson, 2011), quando ocorre uma interrupção que ative este modo, ou em caso de erro na *cache* (Hennessy e Patterson, 2011). O segundo caso ocorre devido a instruções de interrupção (código em nível de usuário, ou em nível de sistema operacional). O terceiro caso, ocorre faz com que o processador gaste ciclos esperando dado(s) da memória principal.

A última equação não incluí qualquer referência à quantidade de instruções necessárias para o programa de usuário e/ou sistema operacional. No entanto, tanto o programa do usuário quanto o sistema operacional possuem um fluxo de instruções, que é executado

pelo *hardware*. Portanto, os ciclos efetivos dependem do número de instruções. Como instruções diferentes gastam diferente quantidade de tempo para serem executadas, o número de ciclos efetivos é a soma de todas as instruções multiplicado pelo número de ciclos necessários para sua implementação. Assim, a quantidade de ciclos efetivos é:

$$Ciclos_{efetivos} = \sum_{i=1}^{n}(Ciclos_i \times Instrucoes_i) \tag{5.2}$$

Então a quantidade de ciclos total do sistema incluindo os ciclos não efetivos quando o processador entra em modo *sleep* é dada por:

$$Ciclos = \sum_{i=1}^{n}(Ciclos_i \times Instrucoes_i) + Ciclos_{sleep} \tag{5.3}$$

Além de gastar ciclos para executar o processo do usuário e processos do sistema operacional, o *hardware* também gasta ciclos executando rotinas do sistema de entrada/saída. Neste caso, a quantidade total de ciclos é dada por:

$$Ciclos_{total} = Ciclos_{usuario} + Ciclos_{SO} + Ciclos_{I/O} \tag{5.4}$$

Aplicando a equação 5.3 dentro da 5.4 e aplicando a equação 5.3 dentro da 5.4, a quantidade total de ciclos é dada por:

$$\begin{aligned} Ciclos_{total} = & \sum_{i=1}^{n}(Ciclos_{usuario_i} \times Instrucoes_{usuario_i}) \\ & + \sum_{i=1}^{n}(Ciclos_{SO_i} \times Instrucoes_{SO_i}) \\ & + Ciclos_{sleep} \\ & + Ciclos_{I/O} \end{aligned} \tag{5.5}$$

portanto, a quantidade de energia dissipada pela aplicação em uma execução é definida como segue:

$$Energia_{aplicacao} = Voltage \times Ampere \times TempoCiclo \times Total_{ciclos} \tag{5.6}$$

sendo que *Voltage* é a voltagem utilizada pelo processador, *Ampere* é o consumo de corrente do processador em estado ativo, *TempoCiclo* é o período de um ciclo de *clock*, e $Total_{ciclos}$ é o número total de ciclos gastos pelo processador para executar a aplicação.

Este modelo de energia para o processamento de dados contêm todas as condições para a estimativa do consumo de corrente do processador, exceto a dissipação de energia estática (Hennessy e Patterson, 2011), que ocorre devido ao vazamento de corrente que flui mesmo quando o transistor está desligado. Observe que a quantidade total de ciclos ($Total_{ciclos}$) resulta diretamente em um maior consumo de energia.

5.3 O MODELO DE ENERGIA DO PROCESSADOR UNB-RISC16

Como o conjunto de instruções do processador `UNB-RISC16` é dividido entre: instruções aritméticas, instruções lógicas,instruções de transferência, instruções de desvio condicional e instruções de desvio incondicional, então o modelo matemático proposto anteriormente pode ser ajustado da seguinte forma:

$$
\begin{aligned}
UNB - RISC16_{energia} = &\ Voltage \\
&\times Ampere \\
&\times TempoCiclo \\
&\times (\sum_{i=1}^{n}(C_{aritmetica_i} \times I_{aritmetica_i}) \\
&+ \sum_{i=1}^{n}(C_{logica_i} \times I_{logica_i}) \\
&+ \sum_{i=1}^{n}(C_{dTransfer_i} \times I_{dTransfer_i}) \\
&+ \sum_{i=1}^{n}(C_{cDesv_i} \times I_{cDesv_i}) \\
&+ \sum_{i=1}^{n}(C_{iDesv_i} \times I_{iDesv_i}))
\end{aligned}
\tag{5.7}
$$

sendo que C_x é a quantidade de ciclos gastos em instruções do tipo x e I_x é a quantidade de instruções do tipo x.

Uma vez que o processador `UNB-RISC16` não possui *pipeline*, memória cache, e operações de entrada/saída é possível refinar a equação 5.7 para os ciclos gastos pelo processo de usuário, e do sistema operacional. Portanto, a equação 5.7 representa o consumo de energia em Joules do processador `UNB-RISC16` no modo ativo.

5.4 Modos da Ferramenta r16-eprof

5.4.1 Estático

O modo estático da ferramenta R16-EPROF executa as funcionalidades mencionadas nas seções anteriores, considerando apenas as características do processador. O modo estático não simula interrupções que não estejam especificadas no código.

5.4.2 Dinâmico

O modo dinâmico permite que o contexto de aplicação seja especificado considerando as características do nó sensor. O nó sensor representado no modo dinâmico pode receber uma combinação de até três dispositivos (dispositivo de comunicação, conversor-AD e porta serial). O modo dinâmico do R16-EPROF pode simular diferentes sinais de interrupções para estes dispositivos, que devem ser especificados no arquivo de configuração.

A primeira característica que pode ser considerada no arquivo de configuração é o contexto em que o processador `UNB-RISC16` irá entrar em modo *sleep*. O desenvolvedor pode especificar o intervalo de ciclos que o nó sensor irá entrar em modo de suspensão. Uma outra forma com que o modo `sleep` pode ser configurado é considerando que o processador entre em modo de suspensão, se a fila de interrupções programada está vazia.

A segunda característica a ser considerada é o comportamento da simulação em relação ao tempo. O desenvolvedor pode especificar se o tempo de execução da aplicação considerado pelo R16-EPROF será o tempo real (ou tempo natural do relógio), ou o tempo do processador `UNB-RISC16`. O tempo do processador é simulado de acordo com a quantidade de ciclos executados.

Uma outra característica que pode ser definida no modo dinâmico é o intervalo de ciclos que ocorre uma determinada interrupção. Por exemplo, um desenvolvedor pode definir que a interrupção do dispositivo serial irá ocorrer a cada trinta e dois milhões de ciclos. Isso corresponderia a dois segundos na execução de uma aplicação pelo processador `UNB-RISC16`. Em outras palavras, a cada dois segundos de operação contínua no processador `UNB-RISC16` a interrupção do dispositivo serial irá ocorrer. Se diretiva de tempo real é ativada, o desenvolvedor também pode especificar o tempo em segundos que uma interrupção permanecerá ativa. Para utilizar este recurso é necessário que o desenvolvedor declare as diretivas .real-time , e .time no arquivo de configuração.

Para possibilitar a geração de instâncias de interrupção em tempo real, foi utilizada na implementação do modo dinâmico do *toolchain* MRISC16T a função *Alarm*. *Alarm*

é uma função da biblioteca `unistd` (Silberschatz et al., 2008) utilizada para comunicação com o sistema operacional. Esta comunicação tem como finalidade gerar interrupções no processo em execução. A função *Alarm* gera um sinal SIGALRM para o processo depois de uma quantidade especifica de segundos em tempo real. O SIGALRM gera uma chamada de interrupção para ativar uma função de manipulação, que pode conter rotinas armazenadas para responder a ocorrência de tal interrupção.

Solicitações de alarmes gerados não são empilhadas, portanto, apenas uma ocorrência de SIGALRM pode ser programado de cada vez. A função *alarm* é uma boa solução para gerar interrupção por sua eficiência em escalonar estas interrupções, sem que haja atraso na resposta. Assim, pode se dizer que a função alarme pode ser utilizada em aplicações com restrições de tempo real.

O arquivo de configuração utilizado para definir o contexto de aplicação deve ser escrito utilizando as seguintes diretivas:

- **custom-device**. Utilizado para especificar os dispositivos do um nó sensor. Estes dispositivos podem ser, de comunicação, serial e conversor-AD. A quantidade de bateria e suas respectivas cargas também podem ser descritas por meio das diretivas *batery-amount* e *batery-charge*.

- **custom-sleep**. Permite especificar como e quando o processador `UNB-RISC16` entra em modo *sleep*.

- **like-rtime**. Define o comportamento da aplicação para o contexto temporal considerando o tempo real.

- **cycles-period**. Permite agendar uma determinada interrupção para ocorrer a cada quantidade específica de ciclos.

- **on-the-cycle**. É utilizado para agendar uma interrupção para ocorrer após uma determinada quantidade de ciclos.

- **on-the-time**. É utilizado para agendar uma interrupção para ocorrer após uma determinada quantidade de segundos.

Estas diretivas compõem uma pequena linguagem de configuração (Sebesta, 2009). Um exemplo de arquivo de configuração do sistema R16-EPROF pode ser visto a seguir.

5.5 OS RECURSOS DO ENERGY PROFILER

Para realizar uma estimativa correta, a ferramenta R16-EPROF executa as instruções de uma aplicação por meio de um simulador implementado em seu núcleo. O simulador executa com precisão as instruções de uma aplicação, obtendo assim a estimativa correta dos ciclos de processamento consumidos. A ferramenta R16-EPROF fornece as seguintes informações sobre um aplicativo:

- **Histograma de instruções**. Histograma de instruções, fornece um perfil sobre a porcentagem de ciclos e o consumo de energia para cada instrução utilizadas no programa;
- **Histograma do conjunto de instruções**. Histograma do conjunto de instruções, esta informação fornece um perfil sobre a porcentagem de ciclos e consumo de energia para cada categoria de instruções utilizadas no programa;
- ***Profile* do processador**. Fornece uma estimativa detalhada do consumo de energia para todos os estados do processador (*sleep* ou ativo), porém esta funcionalidade esta presente apenas no modo dinâmico;
- **Tempo de vida do sistema**. Fornece uma avaliação detalhada sobre o consumo de energia e a vida útil do sistema (processamento de dados), a qual está disponível somente no modo dinâmico;

Os recursos mencionados anteriormente podem ser acessados por meio das seguintes classes de opções disponíveis no R16-EPROF:

- **Simulation**. A opção *Simulation* executa simulação, ou rotinas de análise de código. Esta classe de opções apresenta como foco a execução de uma aplicação afim de avaliar o seu comportamento, e os efeitos causados pelo fluxo de sua execução na pilha implícita.
- **Monitors**. O simulador R16-EPROF possui como uma de suas funcionalidades os monitores. Este recurso serve para explorar o código fonte de uma aplicação do processador `UNB-RISC16` afim de estudar e analisar o consumo de energia, ou os efeitos dos componentes de *hardware* em relação ao consumo energético. As ferramentas de simulação suporta as ações disponibilizadas pelos monitores, permitindo que esta classe possa ser carregada disponibilizando assim instrumentos para geração de relatórios após o programa concluir a sua execução.

- **Incode**. Esta opção trabalha com o formato de entrada da ferramenta R16-EPROF, ao qual é especificado como opção pela entrada fornecida na linha de comando. Este formato de entrada é utilizado pelas ações disponíveis no R16-EPROF. A finalidade da opção *Incode* é determinar como a entrada deve ser interpretada, e a partir desta executar o programa. Por exemplo, o formato de entrada pode ser o resultado da saída da ferramenta `r16 objdump`, do programa escrito no terminal `linux`, ou do código objeto.

5.6 AS OPÇÕES DISPONÍVEIS NA FERRAMENTA R16-EPROF

Neste subtópico será discutido todos os recursos disponíveis na ferramenta R16-EPROF.

5.6.1 Opções da Classe Simulation

A opção `Simulation` tem cinco funções que fornecem simulação e análise de uma aplicação. Os recursos disponíveis por esta opção são expressos por meio de seus respectivos comandos, que são detalhados como segue:

- **Stack-size**. A opção *Stack-size* invoca a análise sobre a construção da pilha de um programa especificado na entrada do sistema R16-EPROF. Este recurso utiliza a representação abstrata da pilha no código *assembly* da aplicação . A partir desta representação é possível determinar qual será o tamanho da pilha abstrata no melhor, e no pior caso da execução de uma aplicação. O melhor caso e o pior caso também podem ser analisados a partir da ocorrência de interrupções, que podem modificar o comportamento da pilha para um determinado trecho de código de programa. É importante lembrar que os gastos energéticos em cada interrupção são ignorados. Desta forma, o gasto considerado com o processamento será até os pontos do programa ao qual ocorre uma interrupções.

- **Control flow graph**. A opção cfg constrói e exibe um gráfico de fluxo de controle do programa de entrada. Isto é útil para melhor compreensão do programa. Com esta opção é possível conhecer a estrutura do programa, permitindo que o desenvolvedor da aplicação possa reestruturar o código para obter ganho de desempenho. Pontos passíveis de otimização também podem ser localizados. O gráfico pode ser exibido em um formato textual no terminal.

- **Text**. A opção *text* implementa um editor de texto no terminal, permitindo ao usuário interativamente realizar a simulação completa de um código escrito em representação hexadecimal. Esta opção permite um teste rápido de código. Na opção de texto é implementado todas as outras opções disponíveis, no entanto as instruções são executadas em tempo real da máquina hospedeira.

- **ObjectDump clean**. A opção *ObjectDump Clean* recebe como parâmetro um arquivo escrito pela saída da ferramenta R16-OBJDUMP e remove todos os comentários deixando apenas o código da aplicação em representação hexadecimal o que resulta em algo mais adequado para a análise automatizada. Em outras palavras, a opção *ObjectDump clean* limpa o arquivo de saída do R16-OBJDUMP deixando apenas o código da aplicação, livre de comentários e informações de cabeçalho.

- **Start**. A opção *Start* fornece simulação de aplicações considerando um único nó sensor. No futuro com esta ação será possível executar simulação considerando vários nós de uma rede de sensores sem fio.

5.7 OPÇÕES DA CLASSE Monitors

A classe *Monitors* possui treze funções que fornecem ao desenvolvedor recursos para monitorar a execução de um programa, a fim de estudar e analisar o consumo de energia ou os efeitos dos componentes de *hardware*. Os comandos juntamente com as funcionalidades da classe *Monitors* são descritos a seguir:

- **Exec**. A opção *exec* monitora o comportamento das chamadas e retornos ocorridas durante a execução de um programa, exibindo o empilhamento das chamadas de função e manipuladores de interrupção. No modo padrão, a opção *instructions-show* exibe todo o código da aplicação em execução;

- **Energy-Consumption**. A opção *energy-consumption* realiza análise do consumo de energia exibindo no final da simulação de uma aplicação o tempo de duração da simulação, a quantidade de ciclos de processamento consumidos, e o consumo total de energia;

- **Energy-log**. A opção *energy-log* exibe o consumo de energia gasto por cada instrução e armazena estas informações em um arquivo chamado energy.log;

- **Energy-Profile**. O comando *energy-profile* calcula o consumo de energia exigido pela aplicação e exibe um relatório no final da execução. O relatório pode ser emitido

na saída padrão, ou em um arquivo. A opção *energy-profile* é semelhante a opção *energy-log*, no entanto retorna uma análise mais detalhada;

- **Terminal**. O comando *terminal* permite ao usuário interagir com o programa enquanto ele é executado por meio de inserção de *breakpoints* para inspecionar o estado da simulação. Quando um *breakpoint* for atingido, o R16-EPROF exibe o total de ciclos de processamento gastos até o ponto de *breakpoint*;

- **Interrupt-Analysis**. A opção *interrupt-analysis* de monitoração exibe as alterações no estado das interrupções, incluindo postagem, habilitação e invocação das interrupções informando o tipo e o momento em que ocorreu as interrupções;

- **Memory-Analysis**. A opção *memory-analysis* coleta informações sobre a utilização da memória pela aplicação, incluindo informações que descrevem o número de leituras e escritas em cada *byte* da memória de dados;

- **Instructions-Profile**. A opção *instructions-profile* monitora o histórico de execução de cada instrução no programa e gera um relatório textual que contém a frequência de execução de todas as instruções;

- **Like-Real-time**. Este recurso é útil quando se deseja conhecer o comportamento do sistema caso ocorra interrupções. Para que se possa conhecer o comportamento do sistema com a ocorrência de interrupções aleatórias e customizáveis é necessário que a ferramenta execute as aplicações em tempo real;

- **Sleep-Mode-Analysis**. O recurso *sleep-mode-analisis* é um monitor que rastreia a frequência com que as aplicações entram em modo *sleep*, incluindo o número total de ciclos em atividade e o número total de ciclos em modo *sleep* durante a simulação;

- **Stack-Analysis**. A opção *stack-analysis* calcula o comprimento da pilha enquanto o programa é executado, relatando o comprimento máximo da pilha detectado;

- **Instructions-Show**. O comando *instructions-show* rastreia a execução de toda a aplicação imprimindo cada instrução a medida com que estas instruções são executadas;

- **Time-Flags**. O comando *time-flags* grava informações sobre a aplicação que consiste no tempo que leva (em média) a execução de uma aplicação atingir um ponto, a partir de outro ponto especificado na aplicação.

5.8 OPÇÕES DA CLASSE INCODE

A classe *Incode* possui quatro funcionalidades que permitem especificar o formato de entrada que será utilizada pelas ações das outras classes, que poderão ser utilizadas com as opções da classe *Incode* para determinar como interpretar a entrada e simular uma aplicação a partir dos arquivos especificados. As opções disponíveis da classe *Incode* e os comandos para acessar estas opções são descritos a seguir:

- **Default**. O formato de entrada padrão carrega uma aplicação a partir de um único arquivo de cada vez. O modo *default* utiliza a extensão do nome do arquivo para decidir qual o melhor leitor a ser utilizado;

- **Gas-Format**. A opção *gas-format* lê os programas que são escritos a partir da linguagem *Assembly*. Esta opção considera arquivos gerados pela ferramenta R16-AS cuja extensão é ".o";

- **r16-Objdump**. O formato dos arquivos resultantes da execução da ferramenta *r16-objdump* podem ser lidos pelo simulador do R16-EPROF. A ferramenta R16-OBJDUMP é um recurso do *toolchain* `MRISC16t`. Um arquivo ELF deve ser desmontado com R16-OBJDUMP para criar um arquivo de texto legível para o R16-EPROF;

- **objdump-clear**. O comando *objdump-clear* limpa o arquivo resultante da saída do R16-OBJDUMP deixando somente o código da aplicação.

No próximo tópico são mostrados os resultados obtidos utilizando as ferramentas do `MRISC16t` para estimar o consumo de ciclos de processamento para um conjunto de aplicações.

6

RESULTADOS

Este tópico aborda os resultados obtidos com a utilização das ferramentas do *toolchain* MRISC16T em um conjunto de programas distintos. Os resultados são abordados considerando o consumo de energia e o tamanho de código final de cada aplicação. O objetivo do experimento realizado é mostrar as funcionalidades das ferramentas do *toolchaim* MRISC16T, sendo elas:

- geração de código para o processador `UNB-RISC16`;
- otimização de código em relação ao tamanho em bytes;
- execução das aplicações;
- estimativa de consumo de energia;
- estimativa de acessos realizados na memória para leitura e escrita;
- avaliar a qualidade do código otimizado considerando os consumos de energia e memória;

O experimento envolve as principais ferramentas do *toolchaim* MRISC16T, exemplificando como esse conjunto de ferramentas é utilizado para desenvolver aplicações de baixo consumo energético.

Tabela 6.1: Conjunto de aplicações utilizadas

Aplicação	Descrição
Binary search	algoritmo de busca em árvores binárias que segue o paradigma de divisão e conquista
Bubble Sort	ordenação pelo método da bolha
Cyclic Redundancy Check	detecção de erros a partir de um receptor de uma mensagem transmitida por meio de um canal com ruído
Deshash	recebe como entrada o valor *hash*, uma *string* criptografada e fornece como saída a *string* original
Dijkstra	encontra o caminho mais curto a partir de uma aresta em um grafo com o valor dos pesos das arestas não negativos
Fast Fourier Transform	calcula a transformada discreta de Fourier
Fibonacci	utilizado para calcular o N-ésimo elemento de uma sequência de fibonacci
Inserction Sort	algoritmo de ordenação simples para um conjunto de números, que trabalha com um elemento de cada vez
Jfd	é um algoritmo muito utilizado em processamento digital de imagens e compressão de dados
LUD	fatora uma matriz como o produto de uma matriz triangular inferior e uma matriz triangular superior
Multiplicação de Matrizes	recebe como entrada duas matrizes e realiza o produto dessas matrizes alocando o resultado em uma terceira matriz
Inversão de Matrizes	a partir de uma matriz dada este algoritmo encontra a sua inversa
Quick Sort	algoritmo de ordenação que utiliza as abordagens recursiva e divisão e conquista
Select Sort	encontra o menor elemento em uma sublista e realiza a troca da posição desse elemento com o elemento armazenado na primeira posição

6.1 METODOLOGIA

Para avaliar o funcionamento de todas as ferramentas presentes no MRISC16T foram utilizados quatorze núcleos de aplicações distintas sendo esses mostrados na Tabela - 6.1.

O objetivo do experimento é avaliar o funcionamento das ferramentas do MRISC16T. Para isso, foram gerados códigos de montagem (*assembly*) de cada aplicação com e sem otimização. Para os códigos gerados com otimização, foi aplicada a otimização `Os` do compilador R16-CC. A partir dos códigos de montagem das aplicações foi possível gerar os códigos binários finais por meio da utilização do montador R16-AS e do vinculador R16-LD.

Os códigos binários das aplicações foram submetidos a execução no simulador R16-EPROF, com a opção `energy-log` habilitada como visto no apêndice b. Com isso foi possível obter o consumo de energia despendido pelas aplicações. Os exemplos de todas as opções da ferramenta R16-EPROF são mostrados no apêndice b. Os exemplos contidos no no apêndice b foram gerados a partir do código fonte da aplicação Dijkstra mostrado no apêndice a.

6.2 RESULTADOS OBTIDOS EM RELAÇÃO AO CONSUMO DE ENERGIA

Os resultados das execuções (com e sem otimização) das aplicações em termos de consumo de energia são mostrados no gráfico da Figura - 6.1. A legenda `Os` representa os resultados

referentes aos códigos otimizados, e a legenda `O0` representa os resultados dos códigos sem otimização. O eixo X do Gráfico da Figura - 6.1 representa os programas utilizados no experimento, e o eixo Y representa o consumo de ciclos de processamento. De uma maneira geral é possível constatar que a aplicação da otimização Os do `GCC` otimiza o consumo de ciclos de processamento, no entanto para os programas Jfd e Lud isto não ocorreu. Isso acontece por dois motivos, a saber:

- o foco da otimização `Os` é a diminuição do tamanho de código; e
- programas estruturados com laços de repetição aninhados tendem a consumir mais ciclos de processamento quando otimizados.

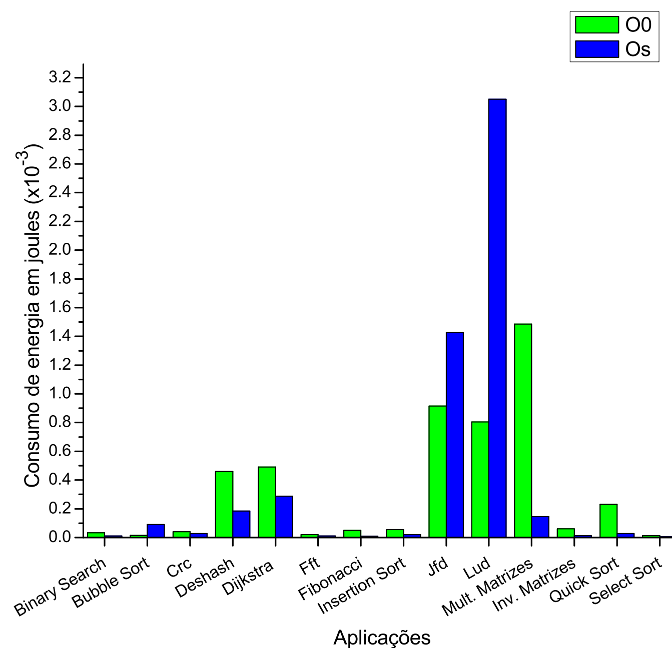

Figura 6.1: Consumo de energia por aplicação

Programas com tamanho de código otimizado nem sempre resultam em uma quantidade menor de ciclos de processamento, pois a otimização de ciclos de processamento depende também da estruturação do código do programa. A otimização `Os` reduz o tamanho do código de uma aplicação com estruturas de repetição por meio da utilização de retrocessos no código realizados por meio de saltos. Isso faz com que um mesmo trecho de código seja executado diversas vezes, porém diminui o tamanho do código. Quanto maior a quantidade de instruções existentes em cada laço de repetição aninhado, maior a quantidade de instruções executadas repetidamente. Sendo assim para a otimização `Os`,

laços de repetição aninhados no código fonte geram um maior consumo de processamento, e consequentemente de energia.

A implementação do programa Jfd possui três laços de repetição aninhados, sendo dois desses laços compostos por 35 instruções cada um. A otimização do código Jfd resultou em uma redução de código de aproximadamente 4 Kb. No entanto, o consumo de energia apresentado pelo código otimizado foi maior, devido ao aninhamento dos laços e a quantidade de instruções existentes em cada um.

A implementação do programa Lud possui uma quantidade de 12 laços de repetição aninhados. Essa quantidade de aninhamento resultou no aumento do consumo de energia na execução do código otimizado. Como a quantidade de aninhamentos de laços de repetição do programa Lud é 4 vezes maior do que no programa Jfd, o programa Lud otimizado obteve um aumento de consumo de energia muito maior que o programa Jfd de acordo com os dados exibidos no gráfico da Figura - 6.1.

As entradas de dados utilizadas nas execuções dos programas foram todas diferentes, sendo assim não será discutido a comparação de consumo de energia entre os programas. Pois o objetivo é avaliar o resultado das otimizações nos códigos das aplicações de maneira individual.

6.2.1 Análise de Consumo de Energia com as Opções Energy-Log e Energy-Profiler

As opções Energy-Log e Energy-Profiler da ferramenta r16-eprof proporcionam uma análise de consumo de energia por função e região de código. As regiões de código são determinadas por marcações inseridas pelo compilador no código *assembly* de um programa. Os exemplos de saídas geradas pelas opções Energy-Log e Energy-Profiler são exibidas no apêndice b. De acordo com os dados fornecidos por essas ferramentas é possível determinar o consumo de energia por função e região de código. O gráfico mostrado na Figura - 6.2 exibe o consumo de energia resultante da execução da aplicação Jfd sem otimização.

Os dados `main`, e `jpeg_Fdct`, são os nomes das funções da aplicação Jfd. L1 e L2 são as *labels* inseridas no código pelo compilador. A função `jpeg_Fdct` e a região de código representada pela *label* L2, são responsáveis pela maior parte do consumo de energia despendido pela aplicação Jfd. Sendo assim, para diminuir o consumo de energia é necessário reestruturar o código da função `jpeg_Fdct` e da região representada pela *label* L2. Desta forma as ferramentas Energy-Log, e Energy-Profiler, são utilizadas para evidenciar partes do código que consomem mais energia.

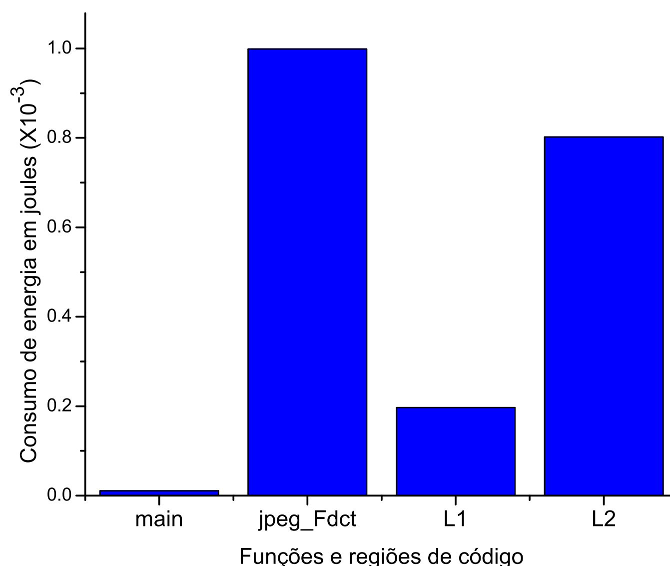

Figura 6.2: Consumo de energia por função e regiões de código da aplicação Jfd otimizada

6.3 RESULTADOS OBTIDOS EM RELAÇÃO AO TAMANHO DE CÓDIGO, E ACESSOS A MEMÓRIA

O processador UNB-RISC16 trabalha em conjunto com uma memória de apenas 64 kilo bytes. Além de armazenar a aplicação, essa quantidade de memória disponível deve ser suficiente para executar a aplicação e armazenar o sistema operacional. Sendo assim medir tamanho de código e quantidade de acessos a memória são processos importantes para avaliar se a aplicação é compatível com a memória disponível. A quantidade de acesso á escrita realizados consomem espaço em memória, sendo assim, quantificando esses acessos é possível saber a quantidade de memória necessária para execução de um programa. O desempenho das aplicações em relação ao tempo de processamento também dependem da quantidade de acessos realizados na memória para leitura e escrita. O tamanho de código binário resultante de cada aplicação são mostrados no gráfico da Figura - 6.3.

O eixo X do Gráfico da Figura - 6.3 representa os programas utilizados no experimento, e o eixo Y representa o tamanho do código binário das aplicações. Todas as aplicações compiladas com a otimização Os apresentaram tamanho de código reduzido quando comparadas com a compilação sem otimização. Isso significa que de fato a otimização Os do compilador R16-CC otimiza o tamanho do código de uma aplicação. Desta forma o *toolchain* MRISC16T gera códigos otimizados que ocupam menos espaço em memória. No entanto, de acordo com o Gráfico da Figura - 6.3 as aplicações Jfd e Lud tiveram

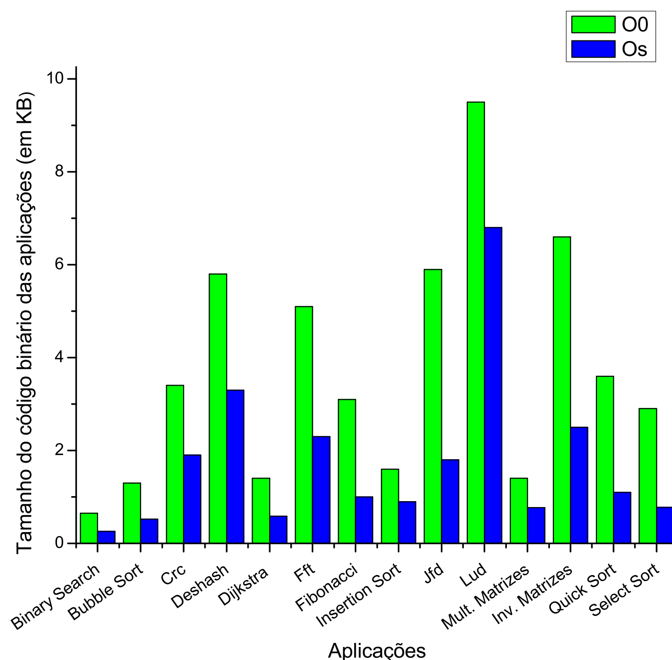

Figura 6.3: Tamanho de código das aplicações

o tamanho de código reduzido pela otimização `Os`, porém consumiram mais energia quando submetidas a execução na ferramenta R16-EPROF. Isso significa que nesses casos compensaria mais utilizar o código sem otimização para economizar energia. Sendo assim, o uso da ferramenta R16-EPROF permite detectar se a otimização impacta positivamente ou negativamente no consumo de energia.

6.3.1 Análise de Consumo de Memória Utilizando a Opção Memory-Analysis

A opção MEMORY-ANALYSIS da ferramenta R16-EPROF quantifica os acessos realizados na memória para leitura e escrita, informando a posição na memória que ocorreu o acesso. O gráfico da Figura - 6.4 exibe a quantidade de acessos realizados na memória para a aplicação Jfd. De acordo com os resultados obtidos é possível constatar que a utilização da otimização `Os` na aplicação Jfd, aumentou a quantidade de acessos realizados na memória tanto para leitura, quanto para a escrita.

Mesmo que a aplicação Jfd possua um tamanho de código reduzido quando otimizada de acordo com os resultados exibidos na Figura - 6.3, a otimização resulta em uma

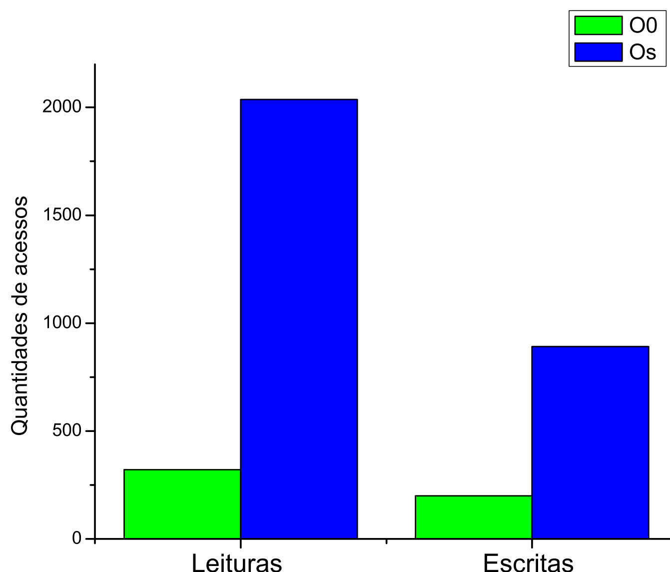

Figura 6.4: Quantidade de acessos a memória pela aplicação Jfd

quantidade maior de memória necessária para execução. O apêndice b contém o exemplo de uma saída gerada pela opção MEMORY-ANALYSIS.

6.4 CONSIDERAÇÕES FINAIS DO EXPERIMENTO

Foram utilizadas para gerar código das aplicações da Tabela - 6.1 as ferramentas do *toolchain* MRISC16T, a saber:

- R16-CC para realizar tradução do código fonte das aplicações escritas em linguagem de programação `C`, para código de montagem do processador `UNB-RISC16`;
- R16-AS para traduzir o código de montagem em código objeto;
- R16-LD para gerar código executável a partir do código objeto das aplicações.

Os códigos executáveis foram submetidos a execução na ferramenta R16-EPROF, sendo possível assim estimar o consumo de energia e memória, pelas aplicações. Com esse experimento foi possível constatar que, em alguns casos a otimização do tamanho de código resulta em um maior consumo de energia. Sendo assim, para que seja possível otimizar o consumo de energia, as funções do algoritmo da aplicação devem ser reestruturadas pelo programador. Para isso é necessário que o programador saiba quais as funções que consomem maior quantidade de energia. Com a utilização das opções ENERGY-PROFILER,

ou Energy-Log, da ferramenta r16-eprof é possível obter o consumo de energia de uma aplicação por função. Sendo assim, é possível saber as funções de uma aplicação que devem ser reestruturadas para que o algoritmo consuma menos energia.

7

CONCLUSÃO E TRABALHOS FUTUROS

MRISC16T é um conjunto de ferramentas desenvolvidas para fornecer suporte ao desenvolvimento de aplicações para o processador UNB-RISC16 escritas em linguagem C. Esse *toolchain* tem como diferencial fornecer ferramentas para execução e estimativa de consumo de energia de aplicações para o processador UNB-RISC16. O processador UNB-RISC16 é um processador projetado para ser utilizado em nós sensores de RSSFs.

Para o o desenvolvimento de código de aplicações, o *toolchain* MRISC16T possui uma versão do conjunto de ferramentas de compilação GCC para o processador UNB-RISC16. Sendo o GCC uma ferramenta redirecionável, foi possível implementar e incluir o *back-end* para o processador UNB-RISC16. Isso tornou possível a geração de código de montagem de aplicações para o processador UNB-RISC16.

A partir do código de montagem, o montador R16-AS realiza tradução gerando o código objeto da aplicação seguindo o formato UNB-RISC16 proposto. O vinculador R16-LD é utilizado para ligar arquivos objetos de um programa e gerar o arquivo binário executável. A ferramenta R16-OBJDUMP é utilizada para desmontar o código objeto realocável de um programa, exibindo informações de cada segmento de maneira textual.

As aplicações geradas pelas ferramentas R16-CC do GCC, R16-AS e R16-LD podem ser executadas na ferramenta R16-EPROF. Em RSSF o consumo de energia é um fator primordial para a vida útil de um nó sensor. Sendo assim, o *toolchain* MRISC16T por meio da ferramenta R16-EPROF é utilizado para executar e estimar o consumo de energia para aplicações de RSSF que utilizem o processador UNB-RISC16. Com isso é possível identificar se a aplicação desenvolvida é viável considerando o consumo de energia.

Para avaliar o funcionamento do MRISC16T foi realizado experimento com o código fonte de 14 aplicações distintas. A partir desses códigos foi possível obter o código executável de cada aplicação. Esses códigos foram submetidos a execução no simulador R16-EPROF, tornando possível a extração de dados para análise de consumo de energia, e acessos a memória. Em alguns casos foi constatado que a otimização do tamanho de código realizado pelo compilador R16-CC, não garante otimização do consumo de energia.

Sendo assim, o programador deverá melhorar o código da aplicação reestruturando as funções com maior gasto energético. Para encontrar essas funções o R16-EPROF disponibiliza as opções ENERGY-LOG e ENERGY-PROFILER. Essas opções exibem de maneira textual o consumo de energia por função e região de código. Sendo assim, a utilização das ferramentas do *toolchain* MRISC16T possibilita o desenvolvimento de aplicações para o processador `UNB-RISC16`, visando economia de energia.

O trabalho a ser realizado no futuro será o de ampliar a funcionalidade da ferramenta R16-EPROF para simular uma RSSF completa. Desta forma será possível estimar o consumo de energia despendida por toda RSSF. O modo dinâmico será ampliado para customização de novos dispositivos de sensoriamento. Atualmente, a ferramenta R16-EPROF estima que o consumo de energia de uma aplicação, quando executada pelo processador `UNB-RISC16`. Os protocolos e os pacotes utilizados na comunicação dos nos sensores também poderá ser especificado. Uma funcionalidade muito importante a ser adicionada a ferramenta R16-EPROF será a realização de simulação de uma RSSF utilizando a descrição de centenas de nós heterogêneos.

No futuro também poderá ser adicionado no *toolchain* MRISC16T a ferramenta R16-APROF para ser um *profiler* de código fonte, cujo objetivo será coletar informações sobre a estrutura da aplicação que possam ser utilizadas para melhorar a qualidade do código fonte. Para alcançar este objetivo, esta ferramenta realizará duas análises no código fonte (análise de fluxo de dados e de fluxo de controle) e fornecerá como saída um arquivo contendo as informações sobre os dados e a estrutura da aplicação. O arquivo de saída do R16-APROF poderá conter sugestões para alteração do código fonte, e informar o consumo de energia por cada estrutura.

A idéia é que no futuro a ferramenta R16-EPROF possibilite a simulação e a estimativa do consumo de energia para uma RSSF inteira que utilize o processador `UNB-RISC16`, levando em conta todas as características possíveis de comunicação e arquitetura de uma RSSF. A ferramenta R16-APROF poderá ser utilizada na localização de trechos de códigos que causam consumo de energia inapropriado, possibilitando que o desenvolvedor realize a reestruturação do código de modo eficiente.

REFERÊNCIAS

AHMED, A.; APPEL, A. W.; RICHARDS, C. D.; SWADI, K. N.; TAN, G.; WANG, D. C. Semantic foundations for typed assembly languages. *ACM Trans. Program. Lang. Syst.*, p. 7:1–7:67, 2010.

AHO, A. V.; SETHI, R.; ULLMAN, J. D. *Compilers: principles, techniques, and tools.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

AL SAAD, M.; FEHR, E.; KAMENZKY, N.; SCHILLER, J. ScatterClipse: A Model-Driven Tool-Chain for Developing, Testing, and Prototyping Wireless Sensor Networks. In: *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*, 2008, p. 871–885.

AQUINO, A. L. L.; FIGUEIREDO, C. M. S.; LOUREIRO, A. A. F.; MINI, R. A. F.; NAKAMURA, E. F.; OLIVEIRA, H. A. B. F.; RUIZ, L. B. Wireless sensor networks for monitoring amphibians. In: *Proceedings of Grand Challenges in Computer Science Research in Latin America Workshop*, Belo Horizonte, MG, BR: SBC, 2008, p. 25–49.

ASIKAINEN, M.; HAATAJA, K.; HONKANEN, R.; TOIVANEN, P. Designing and simulating a sensor network of a virtual intelligent home using tossim simulator. In: *Proceedings of the 2009 Fifth International Conference on Wireless and Mobile Communications*, Washington, DC, USA: IEEE Computer Society, 2009, p. 58–63.

BALDASSIN, A.; CENTODUCATTE, P.; RIGO, S.; CASAROTTO, D.; SANTOS, L. C. V.; SCHULTZ, M.; FURTADO, O. An open-source binary utility generator. *ACM Trans. Des. Autom. Electron. Syst.*, v. 13, n. 2, p. 27–27, 2008.

BAUGH, L.; ZILLES, C. An analysis of i/o and syscalls in critical sections and their implications for transactional memory. In: *In Proceedings of the Second ACM SIGPLAN Workshop on Languages, Compilers, and Hardware Support for Transactional Computing*, New York, NY, USA: ACM, 2007, p. 43–50.

BELL, G.; DOURISH, P. Yesterdays tomorrows: Notes on ubiquitous computings dominant vision. *Personal Ubiquitous Comput.*, v. 11, 2007.

BRADLEE, D. G.; HENRY, R. R.; EGGERS, S. J. The marion system for retargetable instruction scheduling. *SIGPLAN Not.*, v. 26, p. 229–240, 1991.

BROWN, P. Eof: the free software foundation at 20. *Linux J.*, v. 2005, n. 137, p. 15–20, 2005.

CABE, A. C.; QI, Z.; STAN, M. R. Stacking sram banks for ultra low power standby mode operation. In: *Proceedings of the 47th Design Automation Conference*, New York, NY, USA: ACM, 2010, p. 699–704 (*DAC '10*, v.1).

CHIPLUNKAR, N.; NEELIMA, B.; DEEPAK, D. Multithreaded programming framework development for gcc infrastructure. In: *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, 2011, p. 54 –57.

CHIVERS, I. D.; SLEIGHTHOLME, J. Compiler support for the fortran 2003 and 2008 standards revision 7. *j-FORTRAN-FORUM*, v. 30, n. 2, p. 28–37, 2011.

CHOI, Y.-S.; JEON, Y.-J.; PARK, S.-H. A study on sensor nodes attestation protocol in a wireless sensor network. In: *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, Piscataway, NJ, USA, 2010, p. 574–579.

COSTA, J. D. *Implementation of a CMOS 16-bit RISC processor in a System on-Chip.* Dissertação de Mestrado, University of BrasÃlia, BrasÃlia DF, Brazil, 2004.

CROSSBOW MicaZ Datasheet. 2011.

DAMATO, A.; DA SILVA, A. F.; DA COSTA, J. C.; RUIZ, L. B. Eprof: An accurate energy consumption estimation tool. In: *Proceedings of the XXX International Conference of the Chilean Computer Science Society*, Curico, Chile: Chilean Computer Science Society, 2011a.

DAMATO, A.; DA SILVA, A. F.; DA COSTA, J. C.; RUIZ, L. B. Mrisc16t: The unb-risc16 toolchain. In: *Proceedings of IADIS International Conference Applied Computing*, Rio de Janeiro, Brasil: IADIS, 2011b.

DAMATO, A.; DA SILVA, A. F.; DA COSTA, J. C.; RUIZ, L. B.; HUBNER, R.; FOLEISS, J. H. The unb-risc16 object file format. In: *Proceedings of the XXXI International Conference of the Chilean Computer Science Society*, Val ParaÃso, Chile: Chilean Computer Science Society, 2012.

DEITEL, H. M.; DEITEL, P. J. *Java how to program.* 7th ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.

ENGEL, F.; LEUPERS, R.; ASCHEID, G.; FERGER, M.; BEEMSTER, M. Enhanced structural analysis for c code reconstruction from ir code. In: *Proceedings of the 14th International Workshop on Software and Compilers for Embedded Systems*, New York, NY, USA: ACM, 2011, p. 21–27 (*SCOPES 11*, v.1).

FUMMI, F.; PERBELLINI, G.; ACQUAVIVA, A.; QUAGLIA, D. *Wireless sensor network architectures: Modeling, verification and mapping of applications (hardback).* 4th ed. Upper Saddle River, NJ, USA: Springer-Verlag New York Inc., 2012.

GARCIA, S.; JEON, D.; LOUIE, C. M.; TAYLOR, M. B. Kremlin: Rethinking and rebooting gprof for the multicore age. *SIGPLAN Not.*, v. 46, p. 458–469, 2011.

GORLATOVA, M.; SHARMA, T.; SHRESTHA, D.; XU, E.; CHEN, J.; SKOLNIK, A.; PIAO, D.; KINGET, P.; KYMISSIS, J.; RUBENSTEIN, D.; ZUSSMAN, G. Prototyping energy harvesting active networked tags (enhants) with mica2 motes. In: *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*, 2010, p. 1–3.

GOUGH, B. J.; STALLMAN, R. M. *An introduction to gcc.* second edition ed. USA: Network Theory Ltd, 2005.

GOUMOPOULOS, C.; KAMEAS, A. Ambient Ecologies in Smart Homes. *Comput. J.*, v. 52, p. 922–937, 2009.

GRAAF, B.; LORMANS, M.; TOETENEL, H. Embedded software engineering: The state of the practice. *IEEE Softw.*, v. 20, p. 61–69, 2003.

GSCHWIND, M.; MAURER, D. An extendible mips-i processor in vhdl for hardware/-software co-design. 1996.

HENNESSY, J. L.; PATTERSON, D. A. *Computer organization and design (4nd ed.): the hardware/software interface.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

HERLIEN, R.; OÅ½REILLY, T.; HEADLEY, K.; EDGINGTON, D.; TILAK, S.; FOUNTAIN, T.; SHIN, P. An ocean observatory sensor network application. In: *Proceedings of IEEE Sensors Conference*, New York, NY, USA: ACM, 2010, p. 1837–1842.

HOU, T.-W.; CHEN, F.-G. An anomaly in an interpreter using gcc source-code-level register allocation. *SIGPLAN Not.*, v. 42, p. 9–13, 2007.

HUNTER, S. B. Teaching assembly language without using (as much) assembly language. *J. Comput. Small Coll.*, v. 20, p. 68–78, 2005.

INOUE, H.; SAKAI, J.; EDAHIRO, M. A robust seamless communication architecture for next-generation mobile terminals on multi-cpu socs. *ACM Trans. Embed. Comput. Syst.*, v. 9, p. 19:1–19:28, 2010.

JEREMIASSEN, T. E. Sleipnir:an instruction-level simulator generator. In: *Proceedings of the 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, Washington, DC, USA: IEEE Computer Society, 2000, p. 23–.

KANDEMIR, M.; OZTURK, O. Software-directed combined cpu/link voltage scaling fornoc-based cmps. *SIGMETRICS Perform. Eval. Rev.*, v. 36, p. 359–370, 2008.

KHAN, A.; SAQIB, M.; KALEEM, Z. Functional unit level parallelism in risc architecture. In: *Proceedings of the 7th International Conference on Frontiers of Information Technology*, New York, NY, USA: ACM, 2009, p. 71:1–71:4 (*FIT '09*, v.11).

KIM, K.; MAHMOODI, H.; ROY, K. A low-power sram using bit-line charge-recycling technique. In: *Proceedings of the 2007 international symposium on Low power electronics and design*, New York, NY, USA: ACM, 2007, p. 177–182 (*ISLPED '07*, v.1).

KOLARSKI, A.; KIRILOV, M.; ASENOV, K. C++/pascal developer's handbook. In: *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, New York, NY, USA: ACM, 2008, p. 86–86 (*CompSysTech '08*, v.1).

LEUPERS, R.; MARWEDEL, P. Retargetable code generation based on structural processor descriptions. In: *In Design Automation for Embedded Systems*, USA: Kluwer Academic Publishers, 1998, p. 1–36.

LEVINE, J. R. *Linkers and loaders.* 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

LEVIS, P.; LEE, N.; WELSH, M.; CULLER, D. Tossim: accurate and scalable simulation of entire tinyos applications. In: *Proceedings of the 1st international conference on Embedded networked sensor systems*, New York, NY, USA: ACM, 2003, p. 126–137.

Luk, W.; Coutinho, J.; Todman, T.; Lam, Y.; Osborne, W.; Susanto, K.; Liu, Q.; Wong, W. A High-level Compilation Toolchain for Heterogeneous Systems. In: *SOC Conference, 2009. SOCC 2009. IEEE International*, 2009, p. 9–18.

marc Masgonty, J.; Cserveny, S.; Piguet, C. Low-power sram and rom memories. In: *in Intern. Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS), Yverdon-Les-Bains*, New York, NY, USA: ACM, 2001, p. 12–21.

Nabi, M.; Blagojevic, M.; Basten, T.; Geilen, M.; Hendriks, T. Configuring multi-objective evolutionary algorithms for design-space exploration of wireless sensor networks. In: *Proceedings of the 4th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, New York, NY, USA: ACM, 2009, p. 111–119 (*PM2HW2N '09*, v.1).

Newburn, C. J.; So, B.; Liu, Z.; McCool, M.; Ghuloum, A.; Toit, S. D.; Wang, Z. G.; Du, Z. H.; Chen, Y.; Wu, G.; Guo, P.; Liu, Z.; Zhang, D. Intel's array building blocks: A retargetable, dynamic compiler and embedded language. In: *Proceedings of the 2011 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, Washington, DC, USA: IEEE Computer Society, 2011, p. 224–235 (*CGO '11*, v.1).

Nowak, L.; Marwedel, P. Verification of hardware descriptions by retargetable code generation. In: *Proceedings of the 26th ACM/IEEE Design Automation Conference*, New York, NY, USA: ACM, 1989, p. 441–447.

de Paz Alberola, R.; Pesch, D. AvroraZ: Extending Avrora with an IEEE 802.15.4 Compliant Radio Chip Model. In: *Proceedings of the ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, New York, NY, USA: ACM, 2008, p. 43–50.

Perla, E.; Catháin, A. O.; Carbajo, R. S.; Huggard, M.; Mc Goldrick, C. Powertossim z: realistic energy modelling for wireless sensor network environments. In: *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, New York, NY, USA: ACM, 2008, p. 35–42.

Polley, J.; Blazakis, D.; McGee, J.; Rusk, D.; Baras, J. Atemu: a fine-grained sensor network simulator. In: *Proceedings of the Sensor and Ad Hoc Communications and Networks*, New York, NY, USA: ACM, 2004, p. 145–152.

RAD, R. M.; TEHRANIPOOR, M. Evaluating area and performance of hybrid fpgas with nanoscale clusters and cmos routing. *J. Emerg. Technol. Comput. Syst.*, v. 3, n. 3, 2007.

RITCHIE, D. M.; KERNIGHAN, B. W. *The c programming language*. second edition ed. USA: Prentice Hall, 1988.

SEBESTA, R. W. *Concepts of programming languages (9. ed.)*. New York, NY, USA: Addison-Wesley-Longman, I-XV, 1-670 p., 2009.

SHIM, K.; CHO, Y.; KIM, N.; BAIK, H.; KIM, K.; KIM, D.; KIM, J.; MIN, B.; CHOI, K.; CIESIELSKI, M.; YANG, S. A fast two-pass hdl simulation with on-demand dump. In: *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, Los Alamitos, CA, USA: IEEE Computer Society Press, 2008, p. 422–427 (*ASP-DAC '08*, v.1).

SHNAYDER, V.; HEMPSTEAD, M.; CHEN, B.-R.; ALLEN, G. W.; WELSH, M. Simulating the power consumption of large-scale sensor network applications. In: *Proceedings of the International Conference on Embedded Networked sensor Systems*, New York, NY, USA: ACM, 2004, p. 188–200.

SILBERSCHATZ, A.; BAER, P.; GALVIN; GAGNE, G. *Operating system concepts*. 7 ed. New York, NY, USA: John Wiley e Sons, 1837–1842 p., 2008.

STALLINGS, W. *Computer organization and architecture*. New York, NY, USA: Prentice Hall, 1837–1842 p., 2010.

SUDARSANAM, A.; MALIK, S.; FUJITA, M. Readings in hardware/software co-design. cáp. A retargetable compilation methodology for embedded digital signal processors using a machine-dependent code optimizaton library, Norwell, MA, USA: Kluwer Academic Publishers, p. 506–515, 2002.

SUH, J.; DUBOIS, M. Dynamic mips rate stabilization in out-of-order processors. In: *Proceedings of the 36th annual international symposium on Computer architecture*, New York, NY, USA: ACM, 2009, p. 46–56 (*ISCA '09*, v.1).

TANENBAUM, A. S. *Structured computer organization*. New York, NY, USA: Prentice Hall, 170–189 p., 2006.

TANENBAUM, A. S. *Modern operating systems*. New York, NY, USA: Pearson, 137–142 p., 2008.

Titzer, B.; Lee, D.; Palsberg, J. Avrora: Scalable sensor setwork simulation with precise timing. In: *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, Piscataway, NJ, USA: IEEE, 2005, p. 477–482.

Vinnakota, B. Implementing multiplication with split read-only memory. *IEEE Transactions on Computers*, v. 44, p. 1352–1356, 1995.

Welsh, N.; Solsona, F.; Glover, I. Schemeunit and schemeql: Two little languages. In: *In Third Workshop on Scheme and Functional Programming*, New York, NY, USA: ACM, 2002, p. 24–40.

Werner-Allen, G.; Dawson-Haggerty, S.; Welsh, M. Lance: optimizing high-resolution signal collection in wireless sensor networks. In: *Proceedings of the 6th ACM conference on Embedded network sensor systems*, New York, NY, USA: ACM, 2008, p. 169–182 (*SenSys '08*, v.1).

Yang, D. X. D.; Gamal, A. E.; Fowler, B.; Tian, H. A 640 times; 512 cmos image sensor with ultrawide dynamic range floating-point pixel-level adc. *IEEE Journal of Solid-State Circuits*, v. 34, p. 1821–1834, 1999.

Yang, Y.; Qixian, C.; Sen, G. A new construction method of common embedded cross compiler tool based on newlib. In: *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on*, 2010, p. 817–819.

Zhang, M. *Analysis of object file formats for embedded systems.* Relatório Técnico, Intel Corp, 2002.

Zhao, J.; Nagarakatte, S.; Martin, M. M.; Zdancewic, S. Formalizing the llvm intermediate representation for verified program transformations. *SIGPLAN Not.*, v. 47, n. 1, p. 427–440, 2012.

APÊNDICE A

A.1 Programa Dijkstra Escrito em Linguagem de Programação C

```
#include <stdio.h>

#define GRAPHSIZEM1 5
#define GRAPHSIZE 6
#define INFINITY FF

int dist[GRAPHSIZE][GRAPHSIZE] = {   {32, 32, 54, 12, 52, 56},
                                 {76, 54, 65, 14, 89, 69},
                                 {38, 31, 75, 40, 61, 21},
                                 {80, 16, 53, 14, 94, 29},
                                 {59, 7, 14, 78, 79, 45},
                                 {94, 41, 3, 61, 27, 19}
                              };

int d[GRAPHSIZE];
int visited[GRAPHSIZE];

void dijkstra(int s) {
    int i, k, mini;

    for (i = 0; i < GRAPHSIZE; ++i) {
        d[i] = INFINITY;
        visited[i] = 0;
    }
    d[s] = 0;
    for (k = 0; k < GRAPHSIZE; ++k) {
            mini = -1;
            for (i = 0; i < GRAPHSIZE; ++i)
                if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
                mini = i;

            visited[mini] = 1;

            for (i = 0; i < GRAPHSIZE; ++i)
                if (dist[mini][i])
                if (d[mini] + dist[mini][i] < d[i])
                d[i] = d[mini] + dist[mini][i];
```

```
    }

}

int main() {
dijkstra(1);
return 0;
}
```

A.2 Código Assembly do Programa Dijkstra

```
 .equ GRAPHSIZEM1,5
    .equ GRAPHSIZE,6
    .equ INFINITY,0xFFFF
    .equ INODE,1
    .equ M1_H,0xFF
    .equ M1_L,0xFE

    xor $sp,$sp,$sp             ; $sp Aponta para final da memória
    lui $t0,0xFF
    sft $t0,-8
    lui $t0,0xFE
    or  $sp,$t0,$sp

    lui $t0,hi(.main)
    sft $t0,-8
    j $t0,lo(.main)

.dist:

    .short 32,32,54,12,52,56
    .short 76,54,65,14,89,69
    .short 38,31,75,40,61,21
    .short 80,16,53,14,94,29
    .short 59,07,14,78,79,45
    .short 94,71,03,61,27,19


.d:
    .short INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY

.visited:
    .short 0,0,0,0,0,0

.dijkstra:
    xor $t0,$t0,$t0
    lui $t0,6                ;alocar espaço para i k e mini na pilha
    sub $sp,$sp,$t0
```

```
    xor $s0,$s0,$s0
    add $s0,$sp,$zero
    addi $s0,6
    lw $s0,$s0,$zero        ; $s0 = s

    xor $t0,$t0,$t0
    add $t0,$s0,$zero       ; $t0 = s
    sft $t0,-1              ; $t0*2

    xor $t1,$t1,$t1         ;
    lui $t1,.d              ;
    add $t0,$t0,$t1         ; $t0 endereço de d na posição s
    sw $zero,$t0,$zero      ; d[s] = 0

    xor $s2,$s2,$s2         ; k = 0  do primeiro for

.codigoPrimeiroFor:
    xor $s3,$s3,$s3          ;
    lui $s3,M1_L             ;
    lui $t0,M1_H             ;
    sft $t0,-8               ;
    or $s3,$s3,$t0           ; mini = -1

    xor $s1,$s1,$s1          ; i = 0  do segundo for

.codigoSegundoFor:
    xor $t0,$t0,$t0
    add $t0,$s1,$zero       ; $t0 = i
    sft $t0,-1              ; $t0*2

    xor $t1,$t1,$t1         ;
    lui $t1,.visited        ;
    add $t0,$t0,$t1         ; $t0 endereço de visited na posição i
    lw  $t0,$t0,$zero       ; $t0 = visited[i]

    beq $t0,$zero,.corpoPrimeiroIf       ;Primeiro if
    xor $t0,$t0,$t0
    lui $t0,.fimPrimeiroIf
    j $t0,0

.corpoPrimeiroIf:
    xor $t0,$t0,$t0          ;
    lui $t0,M1_L             ;
    lui $t1,M1_H             ;
    sft $t1,-8               ;
    or $t0,$t0,$t1           ; $t0 = -1

    beq $t0,$s3,.gotoX1      ; Primeiro if interno
    xor $t0,$t0,$t0
    lui $t0,.segundoIfInterno
    j $t0,0

.gotoX1:
    xor $t0,$t0,$t0
```

```
    lui $t0,.X
    j $t0,0

.segundoIfInterno:
    xor $t0,$t0,$t0
    add $t0,$s1,$zero        ; $t0 = i
    sft $t0,-1               ; $t0*2

    xor $t1,$t1,$t1          ;
    lui $t1,.d               ;
    add $t0,$t0,$t1          ; $t0 endereço de d na posição i
    lw  $t0,$t0,$zero        ; $t0 = d na posição i

    xor $t1,$t1,$t1
    add $t1,$s3,$zero        ; $t1 = mini
    sft $t1,-1               ; $t1*2

    xor $t2,$t2,$t2          ;
    lui $t2,.d               ;
    add $t1,$t1,$t2          ; $t1 endereço de d na posição mini
    lw  $t1,$t1,$zero        ; $t1 = d na posição mini


    blt $t0,$t1,.X
    xor $t0,$t0,$t0
    lui $t0,.fimPrimeiroIf
    j $t0,0

.X:
    add $s3,$s1,$zero

.fimPrimeiroIf:

    addi $s1,1               ; Incremento do segundo for
    xor $t0,$t0,$t0
    lui  $t0,GRAPHSIZE

    beq  $t0,$s1,.fimSegundoFor
    xor $t0,$t0,$t0
    lui $t0,.codigoSegundoFor
    j $t0,0

.fimSegundoFor:
    xor $t0,$t0,$t0
    add $t0,$s3,$zero        ; $t0 = mini
    sft $t0,-1               ; $t0*2

    xor $t1,$t1,$t1          ;
    lui $t1,.visited         ;
    add $t0,$t0,$t1          ; $t0 endereço de visited na posição mini
    lui $t1,1
    sw  $t1,$t0,$zero        ; $t0 = visited na posição mini

    xor $s1,$s1,$s1
```

```
.codigoTerceiroFor:

    xor $t2,$t2,$t2
    ;addi $t2,1
    xor $t0,$t0,$t0
    ;lui $t0,GRAPHSIZE

.mult1:
    ;blt $t2,$s3,.fimMult1
    beq $t2,$s3,.fimMult1
    addi $t2,1
    addi $t0,GRAPHSIZE
    lui $t1,hi(.mult1)
    sft $t1,-8
    j $t1,lo(.mult1)

.fimMult1:

    sft $t0,-1                  ; Graphsize * mini * 2

    xor $t1,$t1,$t1
    add $t1,$s1,$zero           ; $t1 = i * 2
    sft $t1,-1

    add $t0,$t0,$t1
    addi $t0,.dist              ; $t0 é o endereço de dist[mini][i]
    lw $t0,$t0,$zero            ; $t0 = dist[mini][i]

    beq $t0,$zero,.finalizarSegundoIf     ; Segundo If
    lui $t1,hi(.terceiroIf)
    sft $t1,-8
    j $t1,lo(.terceiroIf)

.finalizarSegundoIf:

    lui $t0,hi(.fimSegundoIf)
    sft $t0,-8
    j $t0,lo(.fimSegundoIf)


.terceiroIf:

    ; Lembrando $t0 = dist[mini][i]
    xor $t1,$t1,$t1
    add $t1,$s3,$zero        ; $t1 = mini
    sft $t1,-1               ; $t1*2

    xor $t2,$t2,$t2          ;
    lui $t2,.d               ;
    add $t1,$t1,$t2          ; $t1 endereço de d na posição mini
    lw  $t1,$t1,$zero        ; $t1 = d na posição mini
```

```
    add $t0,$t0,$t1          ; $t0 = d[mini] + dist[mini][i]

    xor $t1,$t1,$t1
    add $t1,$s1,$zero        ; $t1 = i
    sft $t1,-1               ; $t1*2

    xor $t2,$t2,$t2          ;
    lui $t2,.d               ;
    add $t1,$t1,$t2          ; $t1 endereço de d na posição i
    lw  $t1,$t1,$zero        ; $t1 = d na posição i


                             ; Terceiro If
    blt $t0,$t1,.atualizarDi
    lui $t2,hi(.fimTerceiroIf)
    sft $t2,-8
    j $t2,lo(.fimTerceiroIf)

.atualizarDi:

    xor $t1,$t1,$t1
    add $t1,$s1,$zero        ; $t1 = i
    sft $t1,-1               ; $t1*2

    xor $t2,$t2,$t2          ;
    lui $t2,.d               ;
    add $t1,$t1,$t2          ; $t1 endereço de d na posição i

    sw $t0,$t1,$zero

.fimTerceiroIf:
.fimSegundoIf:

    addi $s1,1               ; Incremento do terceiro for
    xor $t0,$t0,$t0
    lui  $t0,GRAPHSIZE

    beq  $t0,$s1,.fimTerceiroFor
    lui $t0,hi(.codigoTerceiroFor)
    sft $t0,-8
    j $t0,lo(.codigoTerceiroFor)

.fimTerceiroFor:


    addi $s2,1               ; Incremento do primeiro for
    xor $t0,$t0,$t0
    lui  $t0,GRAPHSIZE

    beq  $t0,$s2,.fimPrimeiroFor
    xor $t0,$t0,$t0
    lui $t0,.codigoPrimeiroFor
    j $t0,0
```

```
.fimPrimeiroFor:

    j $ra,0

.main:
    xor $t0,$t0,$t0
    lui $t0,.dijkstra
    xor $t1,$t1,$t1
    lui $t1,INODE
    sw $t1,$sp,$zero
    jal $t0,0
    addi $gp,0xff
```

A.3 Arquivo de Configuração Utilizado no Modo Dinâmico

```
.dynamic

custom-device {
        device:  communication 45, serial 2 , ad-conversor 4
        batery-amount: 2
        batery-charge: 2300

}
custom-sleep {

    each: 4000000 500
    event-queue-empty: 15000000
}

like-rtime


.appcontext


cycles-period  10000000 0:


on-the-cycle   10  2,
on-the-cycle    5 2,
on-the-cycle    540  1,
on-the-cycle    45 0

.time

on-the-time 5 2
on-the-time 2 1
```

APÊNDICE B

B.1 Saída Da Opção time-flags com os Breakpoints `.dijkstra` e `.main`

```
-----------------------------------------------------------------------------------------
-     TOOLCHAIN  UNB-RISC16                -
-     Energy Profile Version 1.00          -
-                                          -
-     Trip Time Flags                      -
-     Options: time-flags & BreakPoints: .dijkstra and .main
-                                          -        Sumary
-     Program   dijkstra.o
-                                          -
-                                          -        1-Times
-                                          -           1.1    Real time
-                                          -           1.2    Processor Time
-                                          -        2-Flags Reached
------------------------------------------------------------------------------------------
- Section 1 [Times]
-
-      1.1  Time Consumption - Real Time (in seconds):                 0
-      1.2  Time Consumption - Processor UNB-RISC16 (in seconds):      0,000000688
------------------------------------------------------------------------------------------

- Function .d Reached After 0 Cycles From the Last Begin
- Function .main Reached After 8 Cycles From the Last Point
------------------------------------------------------------------------------------------
```

B.2 Saída da Opção energy-log

```
-------------------------------------------------------------------------------------
-     TOOLCHAIN  UNB-RISC16                -
-     Energy Profile Version 1.00          -
-     Energy Consumption Estimative        -
-     Options: energy-log & seconds = 0
-     Program  dijkstra.oc
-                                          -           Sumary
-                                          -            1-Cycles
```

```
-                                             -             2-Times
-                                             -               2.1    Real time
-                                             -               2.2    Processor Time
-                                             -             3-Energy Consumption
-------------------------------------------------------------------------------------
- Section 1 [Cycles]
-
- Cycles Consumption                                                   3363
-------------------------------------------------------------------------------------
- Section 2 [Times]
-
-      2.1  Time Consumption - Real Time (in seconds):                 0.000000
-      2.2  Time Consumption - Processor UNB-RISC16 (in seconds):      0.000210
-------------------------------------------------------------------------------------
- Section 3 [Energy Consumption]
-
- Section                                   Energy Spending (Joules)
-  <.main>:                                           1.2692307690
-  <.fimPrimeiroFor>:                                 0.1586538461
-  <.fimTerceiroFor>:                                 6.1874999989
-  <.fimTerceiroIf>:                                  37.1249999933
-  <.atualizarDi>:                                    10.1538461520
-  <.terceiroIf>:                                     116.1346153635
-  <.finalizarSegundoIf>:                             0.0000000000
-  <.fimMult1>:                                       68.5384615260
-  <.mult1>:                                          91.3846153680
-  <.codigoTerceiroFor>:                              11.4230769210
-  <.fimSegundoFor>:                                  9.5192307675
-  <.fimPrimeiroIf>:                                  37.1249999933
-  <.X>:                                              2.0624999996
-  <.segundoIfInterno>:                               44.2644230689
-  <.gotoX1>:                                         2.8557692302
-  <.corpoPrimeiroIf>:                                27.1298076874
-  <.codigoSegundoFor>:                               58.5432692201
-  <.codigoPrimeiroFor>:                              5.7115384605
-  <.dijkstra>:                                       2.6971153841
-  <._start>                                          1.1105769229
                                                    ---------------------
- Total                                             533.3942306723 x10^-6
--------------------------------------------------------------------------------------
```

B.3 Saída da Opção exec

```
------------------------------------------------------------------------------------
-     TOOLCHAIN  UNB-RISC16                   -
-     Energy Profile Version 1.00             -
-                                             -
-     Energy Consumption Estimative           -
-     Options: exec & seconds = 0
-                                             -           Sumary
-     Program  dijkstra.oc
-                                             -
```

```
-                                              -             1-Cycles
-                                              -             2-Times
-                                              -               2.1    Real time
-                                              -               2.2    Processor Time
-                                              -             3-Functions Calls
--------------------------------------------------------------------------------------
- Section 1 [Cycles]
-
- Cycles Consumption                                                    3363
--------------------------------------------------------------------------------------
- Section 2 [Times]
-
-      2.1  Time Consumption - Real Time (in seconds):                  0.000000
-      2.2  Time Consumption - Processor UNB-RISC16 (in seconds):       0.000210
--------------------------------------------------------------------------------------
- Section 3 [Functions Calls]

Section  <.main>: Call! On Cycle 8
Section  <.dijkstra>: Call! On Cycle 15
Section  <.X>: Call! On Cycle 56
Section  <.codigoSegundoFor>: Call! On Cycle 64
Section  <.segundoIfInterno>: Call! On Cycle 82
Section  <.codigoSegundoFor>: Call! On Cycle 107
Section  <.segundoIfInterno>: Call! On Cycle 125
Section  <.fimPrimeiroIf>: Call! On Cycle 145
Section  <.codigoSegundoFor>: Call! On Cycle 152
Section  <.segundoIfInterno>: Call! On Cycle 170
Section  <.fimPrimeiroIf>: Call! On Cycle 190
...
Section  <.codigoTerceiroFor>: Call! On Cycle 3188
Section  <.terceiroIf>: Call! On Cycle 3203
Section  <.fimTerceiroIf>: Call! On Cycle 3224
Section  <.codigoTerceiroFor>: Call! On Cycle 3231
Section  <.terceiroIf>: Call! On Cycle 3246
Section  <.fimTerceiroIf>: Call! On Cycle 3267
Section  <.codigoTerceiroFor>: Call! On Cycle 3274
Section  <.terceiroIf>: Call! On Cycle 3289
Section  <.fimTerceiroIf>: Call! On Cycle 3310
Section  <.codigoTerceiroFor>: Call! On Cycle 3317
Section  <.terceiroIf>: Call! On Cycle 3332
Section  <.fimTerceiroIf>: Call! On Cycle 3353
Section  <.main>: Call! On Cycle 3362
--------------------------------------------------------------------------------------
```

B.4 Saída da Opção instructions-Show + stack-size

```
-----------------------------------------------------------------------------------------
-     TOOLCHAIN  UNB-RISC16                  -
-     Energy Profile Version 1.00            -
-                                            -
-     Options: instructions-Show & stack-size
-     & Seconds = 0
```

```
-                                            -
-                                            -         Sumary
-     Program   dijkstra.oc
-                                            -
-                                            -         1-Cycles
-                                            -         2-Times
-                                            -            2.1    Real time
-                                            -            2.2    Processor Time
-                                            -         3-Instructions Calls
-----------------------------------------------------------------------------------------
- Section 1 [Cycles]
-
- Cycles Consumption                                                  0
-----------------------------------------------------------------------------------------
- Section 2 [Times]
-
-      2.1  Time Consumption - Real Time (in seconds):                0.000000
-      2.2  Time Consumption - Processor UNB-RISC16 (in seconds):     0.000210
-----------------------------------------------------------------------------------------
- Section 3 [Instructions Calls]

- 6ddd  b1ff  9188  b1fe  5d1d  b101  9188  e18c  6111  b170  6222  b201
- 12d0  f100  6111  b106  3dd1  6777  27d0  8706  0770  6111  2170  9181
- 6222  b258  2112  1010  6999  6aaa  bafe  b1ff  9188  5aa1  6888  6111
- 2180  9181  6222  b264  2112  0110  c108  6111  b1fe  b2ff  9288  5112
- c1a8  6111  b1ec  e100  2a80  8801  6111  b106  c188  6111  b19a  e100
- 6111  2180  9181  6222  b264  2112  0110  c108  6111  b1fe  b2ff  9288
...

- 2280  9281  6333  b358  2223  0220  d128  b301  9388  e36e  8801  6111
- b106  c188  b101  9188  e10e  6333  6111  c3ac  9181  6222  2280  9281
- 2112  8110  0110  c108  b201  9288  e23a  6222  22a0  9281  6333  b358
- 2223  0220  2112  6222  2280  9281  6333  b358  2223  0220  d128  b301
- 9388  e36e  8801  6111  b106  c188  b101  9188  e10e  6333  6111  c3ac
- 9181  6222  2280  9281  2112  8110  0110  c108  b201  9288  e23a  6222
- 22a0  9281  6333  b358  2223  0220  2112  6222  2280  9281  6333  b358
- 2223  0220  d128  b301  9388  e36e  8801  6111  b106  c188  8901  6111
- b106  c198  ef00  8cff
-----------------------------------------------------------------------------------------

  The Program Stack Height is 65534

-----------------------------------------------------------------------------------------
```

B.5 Saída do Modo Dinâmico

```
Load Dynamic configuration ! Please Wait ...
START ...
Interrupt 1 Done!!!
Interrupt 2 Done!!!
Sleep For Dynamic Rule!
Interrupts Occurred:
```

```
 Interrupt By AD-Conversor Device!
 Interrupt By Communication Device!
 Interrupt By Serial Device!
 Interrupt By AD-Conversor Device!
 Interrupt By Communication Device!
 Interrupt By Communication Device!
On Sleep!
Sleep For Dynamic Rule!
Sleep For Dynamic Rule!
Awake!
Sleep For Dynamic Rule!
Interrupts Occurred:
 Interrupt By Communication Device!
On Sleep!
Sleep For Dynamic Rule!
-----------------------------------------------------------------------------------------
-     TOOLCHAIN  UNB-RISC16                  -
-     Energy Profile Version 1.00
-                                            -
-     Dynamic Mode Statistcs                 -
-     Options: dynamic & Seconds = 2
-                                            -
-                                            -             Sumary
-     Program   dijkstra.oc
-                                            -             1-Cycles
-                                            -             2-Times
-                                            -               2.1    Real time
-                                            -               2.2    Processor Time
-                                            -             3-Statistics
------------------------------------------------------------------------------------------
- Section 1 [Cycles]
-
- Cycles Consumption                                                    32000000
------------------------------------------------------------------------------------------
- Section 2 [Times]
-
-      2.1  Time Consumption - Real Time (in seconds):                 30
-      2.2  Time Consumption - Processor UNB-RISC16 (in seconds):      2
------------------------------------------------------------------------------------------
- Section 3 [Statistics]
-
- Efetive Cycles: 14880547
- No Efetive Cycles: 17119454
- System Energy Consuption: 2392955.000000 x10^-6 Joules
- System Life Time:  4.727273 hours
-----------------------------------------------------------------------------------------
```

B.6 Saída da Opção instructions-profiler

```
-----------------------------------------------------------------------------------------
-     TOOLCHAIN  UNB-RISC16                  -
-     Energy Profile Version 1.00            -
```

```
-     Instructions Hystogram                -
-     Options: instructions-profiler & seconds = 0
-     Program   dijkstra.oc                                Sumary
-                                           -              1-Cycles
-                                           -              2-Times
-                                           -                2.1    Real time
-                                           -                2.2    Processor Time
-                                           -              3-Hystogram Per Instruction
-                                           -              4-hystogram Per Class
-------------------------------------------------------------------------------------------
- Section 1 [Cycles]
-
- Cycles Consumption                                                3363
-------------------------------------------------------------------------------------------
- Section 2 [Times]
-
-      2.1  Time Consumption - Real Time (in seconds):               0.000000
-      2.2  Time Consumption - Processor UNB-RISC16 (in seconds):     0.000210
-------------------------------------------------------------------------------------------
- Section 3 [Hystogram Per Instruction]
-
- Instruction                  Executed-Times            Per Cent of Total Instruction
- Add                          428                           13.493064
- Sub                          1                             0.031526
- Addi                         296                           9.331652
- Shift                        438                           13.808323
- And                          0                             0.000000
- Or                           28                            0.882724
- Not                          0                             0.000000
- Xor                          616                           19.419924
- Slt                          0                             0.000000
- Lw                           175                           5.517024
- Sw                           16                            0.504414
- Lui                          560                           17.654477
- Beq                          297                           9.363178
- Blt                          51                            1.607818
- J                            265                           8.354351
- Jal                          1                             0.031526
-------------------------------------------------------------------------------------------
- Total                        3172
-------------------------------------------------------------------------------------------
- Section 4 [Hystogram Per Instruction Class]
-
- Class                        Executed-Times            Per Cent of Total Instruction
- Arithmetic                     1163                        36.6645649433
- Logic                          644                         20.3026481715
- Transfer                       751                         23.6759142497
- Conditional Branches           348                         10.9709962169
- Inconditional Branches         266                         8.3858764187
```

B.7 Saída da Opção memory

```
----------------------------------------------------------------------------------------
-     TOOLCHAIN  UNB-RISC16                -
-     Energy Profile Version 1.00          -
-     Memory Hystogram                     -
-     Options: memory & Seconds = 0
-     Program   dijkstra.oc                         Sumary
-                                          -        1-Cycles
-                                          -        2-Times
-                                          -           2.1    Real time
-                                          -           2.2    Processor Time
-                                          -        3-Hystogram Per Memory Position
-                                          -        4-Hystogram Per Read/Writen Access
-----------------------------------------------------------------------------------------
- Section 1 [Cycles]
-
- Cycles Consumption                                               3363
-----------------------------------------------------------------------------------------
- Section 2 [Times]
-
-      2.1  Time Consumption - Real Time (in seconds):               0.000000
-      2.2  Time Consumption - Processor UNB RISC16 (in seconds):     0.000210
-----------------------------------------------------------------------------------------
- Section 3 [Hystogram Per Memory Position]
-
- Memory Position         Access-Times         Read        Writen         Per Cent
-
- 0                       23                  23          0             12.0418848168
- 1                       15                  15          0             7.8534031414
- 3                       2                   2           0             1.0471204188
- 6                       1                   1           0             0.5235602094
- 7                       2                   2           0             1.0471204188
- 12                      2                   2           0             1.0471204188
- 14                      10                  9           1             5.2356020942
- 16                      2                   2           0             1.0471204188
- 19                      2                   2           0             1.0471204188
- 21                      2                   2           0             1.0471204188
- 27                      2                   2           0             1.0471204188
- 28                      1                   1           0             0.5235602094
- 29                      2                   2           0             1.0471204188
- 30                      1                   1           0             0.5235602094
- 31                      2                   2           0             1.0471204188
...

- 149                     1                   1           0             0.5235602094
- 1023                    8                   8           0             4.1884816754
-----------------------------------------------------------------------------------------
- Section 4 [Hystogram Per Read/Writen Acess]
-
- Total Reads                           175 (91.623037 percent)
- Total Writes                          16 (8.376963 percent)
-----------------------------------------------------------------------------------------
```

B.8 Saída da Opção energy-profiler

```
-----------------------------------------------------------------------------------------
-     TOOLCHAIN  UNB-RISC16               -
-     Energy Profile Version 1.00         -
-                                         -
-     Instructions Profile                -
-     Options: energy-profiler & Seconds = 0
-                                         -
-                                         -            Sumary
-     Program   dijkstra.o
-                                         -            1-Cycles
-                                         -            2-Times
-                                         -              2.1    Real time
-                                         -              2.2    Processor Time
-                                         -            3-Profile for Each Instructions
-----------------------------------------------------------------------------------------
- Section 1 [Cycles]
-
- Cycles Consumption                                                    3363
-----------------------------------------------------------------------------------------
- Section 2 [Times]
-
-      2.1  Time Consumption - Real Time (in seconds):                  0.000000
-      2.2  Time Consumption - Processor UNB RISC 16 (in seconds):     0.000210
-----------------------------------------------------------------------------------------
- Section 3 [Profile for Each Instructions]
-
-Section Begin Position | Instruction | Memory Position |  Access Times | Energy Spending

:-Section <.main>:
-396                    8cff         408               1             0.1586538461 x10^-6
-396                    f100         406               1             0.1586538461 x10^-6
-396                    12d0         404               1             0.3173076922 x10^-6
-396                    b201         402               1             0.1586538461 x10^-6
-396                    6222         400               1             0.1586538461 x10^-6
-396                    b170         398               1             0.1586538461 x10^-6
-396                    6111         396               1             0.1586538461 x10^-6
-
-Energy Consumption By Section <.main>:  1.2692307690 x 10^-6 Joules
-

:-Section <.fimPrimeiroFor>:
-394                    ef00         394               1             0.1586538461 x10^-6
-
-Energy Consumption By Section <.fimPrimeiroFor>:  0.1586538461 x 10^-6 Joules
-

:-Section <.fimTerceiroFor>:
-380                    e100         392               5             0.7932692306 x10^-6
-380                    b18e         390               5             0.7932692306 x10^-6
-380                    6111         388               5             0.7932692306 x10^-6
-380                    c198         386               6             0.9519230767 x10^-6
-380                    b106         384               6             0.9519230767 x10^-6
```

```
-380                    6111       382              6             0.9519230767 x10^-6
-380                    8901       380              6             0.9519230767 x10^-6
-
-Energy Consumption By Section <.fimTerceiroFor>:  6.1874999989 x 10^-6 Joules
-

:-Section <.fimTerceiroIf>:
-366                    e10e       378              30             4.7596153837 x10^-6
-366                    9188       376              30             4.7596153837 x10^-6
-366                    b101       374              30             4.7596153837 x10^-6
-366                    c188       372              36             5.7115384605 x10^-6
-366                    b106       370              36             5.7115384605 x10^-6
-366                    6111       368              36             5.7115384605 x10^-6
-366                    8801       366              36             5.7115384605 x10^-6
-
-Energy Consumption By Section <.fimTerceiroIf>:  37.1249999933 x 10^-6 Joules
-

:-Section <.atualizarDi>:
-352                    1120       364              8             2.5384615380 x10^-6
-352                    2223       362              8             1.2692307690 x10^-6
-352                    b358       360              8             1.2692307690 x10^-6
-352                    6333       358              8             1.2692307690 x10^-6
-352                    9281       356              8             1.2692307690 x10^-6
-352                    2280       354              8             1.2692307690 x10^-6
-352                    6222       352              8             1.2692307690 x10^-6
-
-Energy Consumption By Section <.atualizarDi>:  10.1538461520 x 10^-6 Joules
-

:-Section <.terceiroIf>:
-314                    e36e       350              28             4.4423076915 x10^-6
-314                    9388       348              28             4.4423076915 x10^-6
-314                    b301       346              28             4.4423076915 x10^-6
-314                    d128       344              36             5.7115384605 x10^-6
-314                    0220       342              36             11.4230769210 x10^-6
-314                    2223       340              36             5.7115384605 x10^-6
-314                    b358       338              36             5.7115384605 x10^-6
-314                    6333       336              36             5.7115384605 x10^-6
-314                    9281       334              36             5.7115384605 x10^-6
-314                    2280       332              36             5.7115384605 x10^-6
-314                    6222       330              36             5.7115384605 x10^-6
-314                    2112       328              36             5.7115384605 x10^-6
-314                    0220       326              36             11.4230769210 x10^-6
-314                    2223       324              36             5.7115384605 x10^-6
-314                    b358       322              36             5.7115384605 x10^-6
-314                    6333       320              36             5.7115384605 x10^-6
-314                    9281       318              36             5.7115384605 x10^-6
-314                    22a0       316              36             5.7115384605 x10^-6
-314                    6222       314              36             5.7115384605 x10^-6
-
-Energy Consumption By Section <.terceiroIf>:  116.1346153635 x 10^-6 Joules
-
```

```
:-Section <.finalizarSegundoIf>:
-308                    e16e        312                 0             0.0000000000 x10^-6
-308                    9188        310                 0             0.0000000000 x10^-6
-308                    b101        308                 0             0.0000000000 x10^-6
-
-Energy Consumption By Section <.finalizarSegundoIf>:  0.0000000000 x 10^-6 Joules
-

:-Section <.fimMult1>:
-286                    e23a        306                 36             5.7115384605 x10^-6
-286                    9288        304                 36             5.7115384605 x10^-6
-286                    b201        302                 36             5.7115384605 x10^-6
-286                    c108        300                 36             5.7115384605 x10^-6
-286                    0110        298                 36             11.4230769210 x10^-6
-286                    8110        296                 36             5.7115384605 x10^-6
-286                    2112        294                 36             5.7115384605 x10^-6
-286                    9281        292                 36             5.7115384605 x10^-6
-286                    2280        290                 36             5.7115384605 x10^-6
-286                    6222        288                 36             5.7115384605 x10^-6
-286                    9181        286                 36             5.7115384605 x10^-6
-
-Energy Consumption By Section <.fimMult1>:  68.5384615260 x 10^-6 Joules
-

:-Section <.mult1>:
-274                    e212        284                 90             14.2788461513 x10^-6
-274                    9288        282                 90             14.2788461513 x10^-6
-274                    b201        280                 90             14.2788461513 x10^-6
-274                    8106        278                 90             14.2788461513 x10^-6
-274                    8301        276                 90             14.2788461513 x10^-6
-274                    c3ac        274                 126             19.9903846117 x10^-6
-
-Energy Consumption By Section <.mult1>:  91.3846153680 x 10^-6 Joules
-

:-Section <.codigoTerceiroFor>:
-270                    6111        272                 36             5.7115384605 x10^-6
-270                    6333        270                 36             5.7115384605 x10^-6
-
-Energy Consumption By Section <.codigoTerceiroFor>:  11.4230769210 x 10^-6 Joules
-

:-Section <.fimSegundoFor>:
-252                    6888        268                 6             0.9519230767 x10^-6
-252                    1210        266                 6             1.9038461535 x10^-6
-252                    b201        264                 6             0.9519230767 x10^-6
-252                    2112        262                 6             0.9519230767 x10^-6
-252                    b264        260                 6             0.9519230767 x10^-6
-252                    6222        258                 6             0.9519230767 x10^-6
-252                    9181        256                 6             0.9519230767 x10^-6
-252                    21a0        254                 6             0.9519230767 x10^-6
-252                    6111        252                 6             0.9519230767 x10^-6
-
-Energy Consumption By Section <.fimSegundoFor>:  9.5192307675 x 10^-6 Joules
```

```
-

:-Section <.fimPrimeiroIf>:
-238                    e100        250             30              4.7596153837 x10^-6
-238                    b19a        248             30              4.7596153837 x10^-6
-238                    6111        246             30              4.7596153837 x10^-6
-238                    c188        244             36              5.7115384605 x10^-6
-238                    b106        242             36              5.7115384605 x10^-6
-238                    6111        240             36              5.7115384605 x10^-6
-238                    8801        238             36              5.7115384605 x10^-6
-
-Energy Consumption By Section <.fimPrimeiroIf>:  37.1249999933 x 10^-6 Joules
-

:-Section <.X>:
-236                    2a80        236             13              2.0624999996 x10^-6
-
-Energy Consumption By Section <.X>:  2.0624999996 x 10^-6 Joules
-

:-Section <.segundoIfInterno>:
-200                    e100        234             8              1.2692307690 x10^-6
-200                    b1ee        232             8              1.2692307690 x10^-6
-200                    6111        230             8              1.2692307690 x10^-6
-200                    d128        228             15              2.3798076919 x10^-6
-200                    0220        226             15              4.7596153837 x10^-6
-200                    2223        224             15              2.3798076919 x10^-6
-200                    b358        222             15              2.3798076919 x10^-6
-200                    6333        220             15              2.3798076919 x10^-6
-200                    9281        218             15              2.3798076919 x10^-6
-200                    22a0        216             15              2.3798076919 x10^-6
-200                    6222        214             15              2.3798076919 x10^-6
-200                    0110        212             15              4.7596153837 x10^-6
-200                    2112        210             15              2.3798076919 x10^-6
-200                    b258        208             15              2.3798076919 x10^-6
-200                    6222        206             15              2.3798076919 x10^-6
-200                    9181        204             15              2.3798076919 x10^-6
-200                    2180        202             15              2.3798076919 x10^-6
-200                    6111        200             15              2.3798076919 x10^-6
-
-Energy Consumption By Section <.segundoIfInterno>:  44.2644230689 x 10^-6 Joules
-

:-Section <.gotoX1>:
-194                    e100        198             6              0.9519230767 x10^-6
-194                    b1ec        196             6              0.9519230767 x10^-6
-194                    6111        194             6              0.9519230767 x10^-6
-
-Energy Consumption By Section <.gotoX1>:  2.8557692302 x 10^-6 Joules
-

:-Section <.corpoPrimeiroIf>:
-176                    e100        192             15              2.3798076919 x10^-6
-176                    b1c8        190             15              2.3798076919 x10^-6
```

```
-176                      6111       188               15             2.3798076919 x10^-6
-176                      c1a8       186               21             3.3317307686 x10^-6
-176                      5112       184               21             3.3317307686 x10^-6
-176                      9288       182               21             3.3317307686 x10^-6
-176                      b2ff       180               21             3.3317307686 x10^-6
-176                      b1fe       178               21             3.3317307686 x10^-6
-176                      6111       176               21             3.3317307686 x10^-6
-
-Energy Consumption By Section <.corpoPrimeiroIf>:  27.1298076874 x 10^-6 Joules
-

:-Section <.codigoSegundoFor>:
-154                      e100       174               15             2.3798076919 x10^-6
-154                      b1ee       172               15             2.3798076919 x10^-6
-154                      6111       170               15             2.3798076919 x10^-6
-154                      c108       168               36             5.7115384605 x10^-6
-154                      0110       166               36             11.4230769210 x10^-6
-154                      2112       164               36             5.7115384605 x10^-6
-154                      b264       162               36             5.7115384605 x10^-6
-154                      6222       160               36             5.7115384605 x10^-6
-154                      9181       158               36             5.7115384605 x10^-6
-154                      2180       156               36             5.7115384605 x10^-6
-154                      6111       154               36             5.7115384605 x10^-6
-
-Energy Consumption By Section <.codigoSegundoFor>:  58.5432692201 x 10^-6 Joules
-

:-Section <.codigoPrimeiroFor>:
-142                      6888       152               6             0.9519230767 x10^-6
-142                      5aa1       150               6             0.9519230767 x10^-6
-142                      9188       148               6             0.9519230767 x10^-6
-142                      b1ff       146               6             0.9519230767 x10^-6
-142                      bafe       144               6             0.9519230767 x10^-6
-142                      6aaa       142               6             0.9519230767 x10^-6
-
-Energy Consumption By Section <.codigoPrimeiroFor>:  5.7115384605 x 10^-6 Joules
-

:-Section <.dijkstra>:
-112                      6999       140               1             0.1586538461 x10^-6
-112                      1010       138               1             0.3173076922 x10^-6
-112                      2112       136               1             0.1586538461 x10^-6
-112                      b258       134               1             0.1586538461 x10^-6
-112                      6222       132               1             0.1586538461 x10^-6
-112                      9181       130               1             0.1586538461 x10^-6
-112                      2170       128               1             0.1586538461 x10^-6
-112                      6111       126               1             0.1586538461 x10^-6
-112                      0770       124               1             0.3173076922 x10^-6
-112                      8706       122               1             0.1586538461 x10^-6
-112                      27d0       120               1             0.1586538461 x10^-6
-112                      6777       118               1             0.1586538461 x10^-6
-112                      3dd1       116               1             0.1586538461 x10^-6
-112                      b106       114               1             0.1586538461 x10^-6
-112                      6111       112               1             0.1586538461 x10^-6
```

```
-
-Energy Consumption By Section <.dijkstra>:  2.6971153841 x 10^-6 Joules
-

:-Section <.visited>:
-
-Energy Consumption By Section <.visited>:  0.0000000000 x 10^-6 Joules
-

:-Section <.d>:
-88                       ffff         98               0              0.0000000000 x10^-6
-88                       ffff         96               0              0.0000000000 x10^-6
-88                       ffff         94               0              0.0000000000 x10^-6
-88                       ffff         92               0              0.0000000000 x10^-6
-88                       ffff         90               0              0.0000000000 x10^-6
-88                       ffff         88               0              0.0000000000 x10^-6
-
-Energy Consumption By Section <.d>:  0.0000000000 x 10^-6 Joules
-

:-Section <.dist>:
-16                       0013         86               0              0.0000000000 x10^-6
-16                       001b         84               0              0.0000000000 x10^-6
-16                       003d         82               0              0.0000000000 x10^-6
-16                       0003         80               0              0.0000000000 x10^-6
-16                       0047         78               0              0.0000000000 x10^-6
-16                       005e         76               0              0.0000000000 x10^-6
-16                       002d         74               0              0.0000000000 x10^-6
-16                       004f         72               0              0.0000000000 x10^-6
-16                       004e         70               0              0.0000000000 x10^-6
-16                       000e         68               0              0.0000000000 x10^-6
-16                       0007         66               0              0.0000000000 x10^-6
-16                       003b         64               0              0.0000000000 x10^-6
-16                       001d         62               0              0.0000000000 x10^-6
-16                       005e         60               0              0.0000000000 x10^-6
-16                       000e         58               0              0.0000000000 x10^-6
-16                       0035         56               0              0.0000000000 x10^-6
-16                       0010         54               0              0.0000000000 x10^-6
-16                       0050         52               0              0.0000000000 x10^-6
-16                       0015         50               0              0.0000000000 x10^-6
-16                       003d         48               0              0.0000000000 x10^-6
-16                       0028         46               0              0.0000000000 x10^-6
-16                       004b         44               0              0.0000000000 x10^-6
-16                       001f         42               0              0.0000000000 x10^-6
-16                       0026         40               0              0.0000000000 x10^-6
-16                       0045         38               0              0.0000000000 x10^-6
-16                       0059         36               0              0.0000000000 x10^-6
-16                       000e         34               0              0.0000000000 x10^-6
-16                       0041         32               0              0.0000000000 x10^-6
-16                       0036         30               0              0.0000000000 x10^-6
-16                       004c         28               0              0.0000000000 x10^-6
-16                       0038         26               0              0.0000000000 x10^-6
-16                       0034         24               0              0.0000000000 x10^-6
-16                       000c         22               0              0.0000000000 x10^-6
```

```
-16                         0036         20                 0              0.0000000000 x10^-6
-16                         0020         18                 0              0.0000000000 x10^-6
-16                         0020         16                 0              0.0000000000 x10^-6
-
-Energy Consumption By Section <.dist>:  0.0000000000 x 10^-6 Joules
-----------------------------------------------------------------------------------------
```