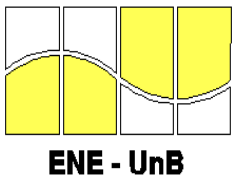
Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica



Desenvolvimento de aplicação e rotina de tratamento de exceções para sistema de comunicações em chip (SOC) sem fio

Relatório Estágio Supervisionado II

Aluna: Juliana Zago França Diniz

Orientador: Professor José Camargo da Costa

Brasília, Setembro de 2002

“A grandeza não consiste em receber honras, mas em merecê-las”

Aristóteles

AGRADECIMENTOS

A Deus, que em todos os momentos fez-se presente em minha vida.

A minha mãe, pelo apoio e pela mão amiga em meu ombro quando precisei dela. Aos meus irmãos pela compreensão e carinho.

Aos professores do departamento de Engenharia Elétrica. Todos, de alguma maneira, me proporcionaram o prazer de enxergar a beleza da ciência. Obrigada.

Em especial ao meu orientador, Prof. José Camargo da Costa. Pela inestimável contribuição à formação do meu caráter, pessoal e profissional, sendo sempre um exemplo de dedicação e ética.

Ao meu co-orientador Ricardo Linder, por possibilitar a realização deste trabalho.

A todos os amigos que de alguma forma contribuíram para esta conquista. A Helen Carvalho do Carmo, por me ouvir sempre que precisei desabafar. A Roberto Monteiro de Barros Reis, por me ensinar a não desistir. A Paulo Eduardo Pinto de Almeida, pela ajuda irretribuível. Ao meu colega Ryan por dividir comigo as angústias e preocupações deste último semestre.

Meus mais sinceros agradecimentos à família Campos, por me acolher. Em especial ao Sr. Paulo Cesar, Sra. Sonia Maria e Luiz Felipe. Sem eles, chegar até aqui jamais teria sido possível. Meu amor e carinho. Sempre.

RESUMO

No presente trabalho foi desenvolvida uma aplicação básica para o sistema de comunicações em chip sem fio, por meio das especificações de *software* e *hardware*. A aplicação realiza operações simples, e inclui a rotina de tratamento de exceções de forma a controlar todo o fluxo de dados proveniente das interfaces de comunicação.

A aplicação tem o objetivo de validar a linguagem proposta para o sistema, certificando sua validade, e a implementação de uma primeira versão da aplicação que, posteriormente, pode ser modificada de modo a atender melhor as necessidades do sistema.

Foram ainda desenvolvidos vetores de teste para validação do *software* montador, desenvolvido para a tradução da linguagem em códigos de máquina interpretáveis pelo SCW.

SUMÁRIO

Página

1 INTRODUÇÃO 1

2 O SISTEMA DE COMUNICAÇÃO SEM FIO (SCW) 3

2.1 DESCRIÇÃO DO MICROPROCESSADOR 3
2.1.1 Especificações Básicas 3
2.1.2 Hardware 4
2.1.3 Software [6] 9

3 METODOLOGIA DE PROJETO 23

3.1 A APLICAÇÃO BÁSICA 23
3.2 TESTES COM O *SOFTWARE* MONTADOR 24

4 A APLICAÇÃO 27

4.1 ESPECIFICAÇÕES 27
4.2 ENTRADA DE DADOS 28
4.3 PROCESSAMENTO 28
4.4 SAÍDA DE DADOS 30
4.5 SAÍDA DE DADOS DE CONTROLE 32
4.6 ALGORITMOS ELABORADOS PARA A APLICAÇÃO 34
4.7 A APLICAÇÃO EM LINGUAGEM DE MÁQUINA 36

5 A ROTINA DE TRATAMENTO DE INTERRUPÇÕES 40

5.1 ESPECIFICAÇÕES 40
5.2 ETAPA INICIAL DO TRATAMENTO DE EXCEÇÕES 41
5.3 TESTES DE IDENTIFICAÇÃO DA OCORRÊNCIA DE EXCEÇÕES PROVENIENTES DAS INTERFACES DE COMUNICAÇÃO 43
5.4 TRATAMENTO INDIVIDUAL DAS EXCEÇÕES 43
5.4.1 Pilhas de Armazenamento Temporário 45
5.5 ESPECIFICAÇÕES PARA O TRATAMENTO DE CADA UM DOS TIPOS DE EXCEÇÃO . 49
5.5.1 Interrupções geradas pela interface ΣΔ 49
5.5.2 Interrupções geradas pelas interfaces de RF e serial 51
5.5.3 Exceções geradas pela ocorrência de overflow ou erro no endereçamento 53
5.6 ETAPA FINAL DO TRATAMENTO DAS EXCEÇÕES 54
5.7 A ROTINA DE TRATAMENTO DAS INTERRUPÇÕES EM LINGUAGEM DE MÁQUINA 58
5.8 O BLOCO PROGRAMACIONAL COMPLETO 65

6 A MONTAGEM DA APLICAÇÃO EM CÓDIGO DE MÁQUINA 66

7 OS TESTES REALIZADOS COM O MONTADOR 70

7.1 TESTES DE MICROINSTRUÇÕES 70
7.1.1 Testes das Instruções Add e Sub 71
7.1.2 Testes da Instrução Addi 71
7.1.3 Teste da Instrução Shift 72
7.1.4 Teste das Instruções And, Or e Xor 72
7.1.5 Teste da Instrução Not 72
7.1.6 Teste das instruções Lw e Sw 72

7.1.7 Teste da instrução Lui *73*
7.1.8 Teste de Instrução Slt *73*
7.1.9 Teste das Instruções Beq e Blt *73*
7.1.10 Teste das Instruções de Desvio Incondicional *74*
7.2 TESTES REALIZADOS COM AS PSEUDO-INSTRUÇÕES 74
8 DISCUSSÃO 77
9 CONCLUSÃO 78
REFERÊNCIAS 80
ANEXO I 82
ANEXO II 93
ANEXO III 105
ANEXO IV 110

LISTA DE TABELAS

Tabela **Página**

Tabela 2.1: Posições reservadas em memória para I/O 5
Tabela 2.2: Registradores presentes no sistema. Uso e conteúdo. 6
Tabela 2.3: Conjunto de microinstruções para o SCW. 10
Tabela 2.4: Ciclos de clock e tempo gasto para realização das microinstruções 12
Tabela 2.5: Pseudoinstruções aritméticas 13
Tabela 2.6: Pseudoinstruções lógicas 13
Tabela 2.7: Pseudoinstruções de Transferência 13
Tabela 2.8: Pseudoinstruções de Desvio Condicional 13
Tabela 2.9: Pseudoinstruções de Desvio Incondicional 14
Tabela 2.10: Pseudoinstrução Add 14
Tabela 2.11: Pseudoinstrução Sub 14
Tabela 2.12: Pseudoinstrução Mul 15
Tabela 2.13: Pseudoinstrução Div 15
Tabela 2.14: Pseudoinstrução Addi 15
Tabela 2.15: Pseudoinstrução Subi 15
Tabela 2.16: Pseudoinstrução Muli 15
Tabela 2.17: Pseudoinstrução Divi 15
Tabela 2.18: Pseudoinstrução Rem 16
Tabela 2.19: Pseudoinstrução Sftl 16
Tabela 2.20: Pseudoinstrução Sftr 16
Tabela 2.21: Pseudoinstrução And 16
Tabela 2.22: Pseudoinstrução Or 16
Tabela 2.23: Pseudoinstrução Nor 16
Tabela 2.24: Pseudoinstrução Xor 16
Tabela 2.25: Pseudoinstrução Not 16
Tabela 2.26: Pseudoinstrução Comp 17
Tabela 2.27: Pseudoinstrução Andi 17
Tabela 2.28: Pseudoinstrução Ori 17
Tabela 2.29: Pseudoinstrução Xori 17
Tabela 2.30: Pseudoinstrução Lw 17
Tabela 2.31: Pseudoinstrução Sw 17
Tabela 2.32: Pseudoinstrução Lb 17
Tabela 2.33: Pseudoinstrução Sb 18
Tabela 2.34: Pseudoinstrução Ld 18
Tabela 2.35: Pseudoinstrução Sd 18
Tabela 2.36: Pseudoinstrução Lui 18
Tabela 2.37: Pseudoinstrução Lwi 18
Tabela 2.38: Pseudoinstrução Mov 18
Tabela 2.39: Pseudoinstrução Mflo 19
Tabela 2.40: Pseudoinstrução Mfhi 19
Tabela 2.41: Pseudoinstrução Move 19
Tabela 2.42: Pseudoinstrução Chg 19
Tabela 2.43: Pseudoinstrução Slt 19
Tabela 2.44: Pseudoinstrução Sle 19
Tabela 2.45: Pseudoinstrução Seq 19
Tabela 2.46: Pseudoinstrução Sne 19
Tabela 2.47: Pseudoinstrução Sgt 19
Tabela 2.48: Pseudoinstrução Sge 20
Tabela 2.49: Pseudoinstrução Beq 20
Tabela 2.50: Pseudoinstrução Bne 20
Tabela 2.51: Pseudoinstrução Blt 20
Tabela 2.52: Pseudoinstrução Bgt 20
Tabela 2.53: Pseudoinstrução Slti 20
Tabela 2.54: Pseudoinstrução Seqi 20
Tabela 2.55: Pseudoinstrução Sgti 20

Tabela 2.56: Pseudoinstrução Beqi ____ 20
Tabela 2.57: Pseudoinstrução Bnei ____ 21
Tabela 2.58: Pseudoinstrução Blti ____ 21
Tabela 2.59: Pseudoinstrução Bgti ____ 21
Tabela 2.60: Pseudoinstrução J ____ 21
Tabela 2.61: Pseudoinstrução Jr ____ 21
Tabela 2.62: Pseudoinstrução Jpc ____ 21
Tabela 2.63: Pseudoinstrução Jal ____ 21
Tabela 2.64: Pseudoinstrução Jalr ____ 21
Tabela 2.65: Pseudoinstrução Jalpc ____ 21
Tabela 2.66: Pseudoinstrução Jd ____ 22
Tabela 2.67: Pseudoinstrução Jald ____ 22
Tabela 2.68: Pseudoinstrução Jalrd ____ 22
Tabela 2.69: Pseudoinstrução Jalpcd ____ 22
Tabela 4.1: Bloco programacional para a aplicação ____ 37
Tabela 5.1: Etapa inicial da rotina de tratamento de exceções em linguagem de máquina ____ 60
Tabela 5.2: Bloco de programa em linguagem de máquina para o tratamento de exceções geradas pela interface ΣΔ ____ 62
Tabela 5.3: Rotina em linguagem de máquina para tratamento das interrupções geradas pelas interfaces de RF ____ 62
Tabela 5.4: Codificação dos erros gerados na transmissão ou recepção de dados pelas interfaces ____ 63
Tabela 5.5: Constantes utilizadas no bloco programacional ____ 63
Tabela 5.6: Rotina de tratamento da ocorrência de overflow e erro de endereçamento em linguagem de máquina ____ 64
Tabela 5.7: Bloco de programa em linguagem de máquina para a etapa final do tratamento de exceções ____ 64
Tabela 6.1: Montagem da primeira etapa da aplicação ____ 66
Tabela 6.2: Primeira versão da multiplicação em linguagem de máquina ____ 67
Tabela 6.3: Sintaxe da operação de multiplicação modificada na etapa de montagem ____ 67
Tabela 6.4: Primeira versão da etapa de identificação da exceção ocorrida ____ 68
Tabela 6.5: Sintaxe modificada para atender ao código resultante da montagem ____ 68
Tabela 7.1: Códigos de Registradores e Microinstruções ____ 70
Tabela I.1 – Teste da instrução Add ____ 82
Tabela I.2 – Teste da instrução Sub ____ 82
Tabela I.3 – Teste da instrução Addi ____ 83
Tabela I.4 – Teste da instrução Shift ____ 84
Tabela I.5 – Teste da instrução And ____ 84
Tabela I.6 – Teste da instrução Or ____ 85
Tabela I.7 – Teste da instrução Xor ____ 86
Tabela I.8 – Teste da instrução Not ____ 87
Tabela I.9 – Teste da instrução Lw ____ 87
Tabela I.10 – Teste da instrução Sw ____ 88
Tabela I.11 – Teste da instrução Lui ____ 89
Tabela I.12 – Teste da instrução Slt ____ 89
Tabela I.13 – Teste da instrução Beq ____ 90
Tabela I.14 – Teste da instrução Blt ____ 91
Tabela I.15 – Teste da instrução J ____ 91
Tabela I.16 – Teste da instrução Jal ____ 92
Tabela II.1 – Teste da instrução Mul ____ 93
Tabela II.2 – Teste da instrução Div ____ 93
Tabela II.3 – Teste da instrução Subi ____ 94
Tabela II.4 – Teste da instrução Muli ____ 94
Tabela II.5 – Teste da instrução Divi ____ 94
Tabela II.6 – Teste da instrução Rem ____ 95
Tabela II.7 – Teste da instrução Sftl ____ 95
Tabela II.8 – Teste da instrução Sftr ____ 95
Tabela II.9 – Teste da instrução Comp ____ 96
Tabela II.10 – Teste da instrução Andi ____ 96
Tabela II.11 – Teste da instrução Ori ____ 96
Tabela II.12 – Teste da instrução Xori ____ 96

Tabela II.13 – Teste da instrução Lb _____ *97*
Tabela II.14 – Teste da instrução Sb _____ *97*
Tabela II.15 – Teste da instrução Ld _____ *97*
Tabela II.16 – Teste da instrução Sd _____ *98*
Tabela II.17 – Teste da instrução Mov _____ *98*
Tabela II.18 – Teste da instrução Mflo _____ *98*
Tabela II.19 – Teste da instrução Mfhi _____ *98*
Tabela II.20 – Teste da instrução Move _____ *99*
Tabela II.21 – Teste da instrução Chg _____ *99*
Tabela II.22 – Teste da instrução Sle _____ *99*
Tabela II 23 – Teste da instrução Seq _____ *99*
Tabela II.24 – Teste da instrução Sne _____ *99*
Tabela II.25 – Teste da instrução Sgt _____ *100*
Tabela II.26 – Teste da instrução Sge _____ *100*
Tabela II.27 – Teste da instrução Bne _____ *100*
Tabela II.28 – Teste da instrução Bgt _____ *100*
Tabela II.29 – Teste da instrução Slti _____ *100*
Tabela II.30 – Teste da instrução Seqi _____ *101*
Tabela II.31 – Teste da instrução Sgti _____ *101*
Tabela II.32 – Teste da instrução Beqi _____ *101*
Tabela II.33 – Teste da instrução Bnei _____ *102*
Tabela II.34 – Teste da instrução Blti _____ *102*
Tabela II.35 – Teste da instrução Bgti _____ *102*
Tabela II.36 – Teste da instrução Jr _____ *102*
Tabela II.37 – Teste da instrução Jpc _____ *103*
Tabela II.38 – Teste da instrução Jalr _____ *103*
Tabela II.39 – Teste da instrução Jalpc _____ *103*
Tabela II.40 – Teste da instrução Jd _____ *103*
Tabela II.41 – Teste da instrução Jald _____ *104*
Tabela II.42 – Teste da instrução Jalrd _____ *104*
Tabela II.43 – Teste da instrução Jalpcd _____ *104*
Tabela III.1 – Conjunto programacional completo _____ *105*
Tabela IV.1 – Bloco montado da aplicação em linguagem de montagem _____ *110*

LISTA DE FIGURAS

Figura **Página**

Figura 2.1: Estrutura básica do sistema SCW ____ *4*
Figura 2.2: Esquemático da memória do sistema SCW ____ *4*
Figura 2.3: Mapeamento dos bits das palavras de setup e status para as interfaces ____ *6*
Figura 2.4: Conteúdo do registrador $int ____ *7*
Figura 2.5: ULA de 1 Bit ____ *8*
Figura 2.6: ULA de 16 bits desenvolvida para o SCW ____ *8*
Figura 2.7: Esquemático completo da unidade de controle de sistema ____ *9*
Figura 2.8: Formato de instrução do tipo R ____ *10*
Figura 2.9: Formato da instrução do tipo I ____ *10*
Figura 2.10:Formato da instrução do tipo J ____ *10*
Figura 3.1: Esquemático da seqüência de atividades seguida na etapa de desenvolvimento ____ *24*
Figura 4.1: Esquemático completo da unidade de controle de sistema ____ *27*
Figura 4.2: Aquisição de dados originados na interface ΣΔ ____ *28*
Figura 4.3: Seqüência de atividades realizadas na etapa de processamento ____ *29*
Figura 4.4: Esquema representativo da transmissão de dados pela interface de RF ____ *31*
Figura 4.5: Esquemático da seqüência de envio de dados de controle ____ *33*
Figura 4.6: Algoritmo para envio de dados às interfaces de saída ____ *33*
Figura 4.7: Esquema seqüencial simplificado par a aplicação ____ *34*
Figura 4.8: Algoritmo para envio de dados à interface de RF ____ *35*
Figura 4.9: Algoritmo para a saída de dados de controle ____ *36*
Figura 5.1: Codificação do registrador $int ____ *41*
Figura 5.2: Etapa inicial do tratamento de exceções ____ *42*
Figura 5.3: Descrição esquemática do uso da memória ____ *45*
Figura 5.4: Descrição esquemática do controle da pilha de armazenamento temporário ____ *46*
Figura 5.5: Utilização da posição de memória #FFF3$_h$ ____ *47*
Figura 5.6: Algortimo de controle do ponteiro de escrita ____ *48*
Figura 5.7: Algortimo para controle do ponteiro de leitura ____ *49*
Figura 5.8: Algortimo para a etapa de processamento da aplicação incluindo o controle do ponteiro de leitura ____ *50*
Figura 5.9: Algortimo para armzenamento em pilha dos dados disponibilizados pela interface ΣΔ ____ *51*
Figura 5.10: Algortimo descritivo do tratamento dado às exceções geradas pelas interfaces serial e de RF ____ *53*
Figura 5.11: Esquemático da seqüência de operação para ocorrência de overflow aritmético ou erro de endereçamento ____ *54*
Figura 5.12: Esquemático da sequência de atividades para a etapa final do tratamento de exceções ____ *56*
Figura 5.13: Algoritmo completo para a rotina de tratamento de exceções ____ *57*
Figura 5.14: Detalhamento das rotinas de tratamento de exceções geradas pelas interfaces Serial e de RF dentro da Figura 5.13 ____ *58*

1 INTRODUÇÃO

Nos últimos anos a demanda por sistemas integrados para comunicações móveis tem crescido consideravelmente. Neste âmbito, a tecnologia busca sempre opções que correspondam a melhor desempenho e que possam explorar, de forma eficiente, as potencialidades de um sistema com estas características. O emprego de circuitos integrados CMOS, por possuir características de baixo consumo, alta densidade de integração e banda de operação ajustável, entre outras, é a uma boa opção. Atualmente podem ser encontradas aplicações de sistemas de comunicações em chip utilizando esta tecnologia, destinadas à telefonia celular, subsistemas de comunicação de diversos tipos, sistemas de processamento de informação, computação, etc.

Entretanto, a maior parte das estruturas existentes destina-se a aplicações específicas, com atributos vinculados a um único tipo de utilização, apesar da tecnologia hoje disponível possibilitar o desenvolvimento de sistemas configuráveis por *software*, o que amplia consideravelmente a gama de utilizações.

Com este conceito vem sendo desenvolvido o SCW (Sistema de Comunicação *Wireless*), um chip de características peculiares no sentido de associar microprocessador, interfaces de comunicação, transceptor RF e memória, num sistema monolítico. O projeto vem sendo desenvolvido em parceria por universidades brasileiras e é uma das frentes de trabalho constituintes do programa PROCAD – Programa Nacional de Cooperação Acadêmica, fomentado pela CAPES.

A proposta pioneira da integração em um único chip, dos componentes de um sistema completo de comunicação sem fio, deve ser capaz de atender às necessidades de processamento de um sistema desta complexidade. Para tal, deve ser dada especial atenção ao *software* que trabalhará neste processador para que o sistema, como um todo, atinja os objetivos propostos com bom desempenho. Foi desenvolvida pela equipe de *software* a linguagem a ser utilizada pelo microprocessador, com um *set* de instruções específico que permite uma flexibilidade de programação tal que, todas as possibilidades de aplicações idealizadas para o sistema possam ser alcançadas. Embora o *set* de instruções não seja inovador em sua composição, ele possibilita a plena operação do projeto e é simples o bastante para ser implementado numa estrutura tão restrita como a do SCW.

O *set* de instruções foi criado, em conjunto com a equipe de *hardware* para atender da melhor forma às expectativas do projeto, tendo em vista que a prioridade, num projeto destas dimensões é dada ao *hardware* em detrimento do *software*, que deve se ajustar às condições propostas.

De forma a testar a aplicabilidade e funcionalidade da linguagem criada para o microprocessador do SCW, foi elaborada neste trabalho uma aplicação na linguagem de máquina criada para o sistema. Os objetivos desta aplicação são: (1) a realização de testes de *hardware* para o primeiro protótipo; e, (2) a validação do *set* de instruções desenvolvido. Neste sentido, a aplicação não foi desenvolvida visando a implementação em chip e não deverá ser usada em escala pelo projeto, por se tratar de um programa básico de testes e verificação.

O objetivo é mostrar que, com o set de instruções criado para o SCW, é possível realizar qualquer tipo de seqüência de atividades desejada para o sistema. Para tanto, a aplicação deve tratar e controlar o fluxo de dados gerado pelas interfaces, e controlar as interrupções recebidas pelo processador.

Assim, a aplicação discutida neste trabalho realiza operações básicas de escrita e leitura de memória, processamento simples e controla de forma ampla todo o fluxo de dados entre o sistema de processamento e as interfaces de comunicação. A aplicação efetua também todo o tratamento de interrupções disparadas ao processador, tendo sido originadas externa ou internamente. Cumprindo tais especificações, a aplicação desenvolvida torna-se bem próxima de um sistema operacional simples. No entanto, seu caráter experimental objetivando, apenas, a realização de testes, torna-a uma ferramenta computacional útil para validação da estrutura de *hardware* proposta, bem como do projeto relativo à parte do *software* básico.

A linguagem criada para o SCW é baseada nos conceitos da arquitetura RISC (RISC - *Reduced Instructions Set Computer*), que prevê a utilização de um conjunto reduzido de instruções para a máquina. De forma a atender aos requisitos de universalidade para a linguagem, foi implementado um conjunto de instruções mais completo, chamado conjunto de pseudoinstruções.

As pseudoinstruções emulam um conjunto maior de instruções, construído utilizando as instruções mais básicas. Para a tradução da linguagem de máquina em códigos de máquina interpretáveis pelo processador, foi elaborado o *software* montador, desenvolvido no trabalho final de graduação da referência [8] . De forma a validar o *software*, identificando falhas e a situação de suas ocorrências, criou-se neste trabalho uma metodologia de testes específica, buscando certificar a funcionalidade do programa desenvolvido.

2 O SISTEMA DE COMUNICAÇÃO SEM FIO (SCW)

O projeto SCW almeja a construção de um chip visando a integração monolítica dos componentes básicos de um sistema de comunicações. A proposta é a implementação, em um único circuito integrado, em tecnologia CMOS (*complementary metal-oxide semiconductor*), de um sistema constituído por um transceptor de RF, operando a freqüências da ordem de 1 GHz, associado a um processador digital, dotado de memória residente e interfaces de comunicação analógicas e digitais.

O sistema de comunicação sem fio comporta aplicações, entre outras, nas áreas de Telemetria e Comunicações Móveis. Devido a sua portabilidade, a unidade de comunicação pode ser conectada a um microcomputador PC/IBM, possibilitando o acesso dessa máquina a outras por meio de um *link wireless* entre as unidades, sejam essas máquinas outros computadores (LAN – *local área network*) ou equipamentos a serem monitorados (Telemetria). A mobilidade do sistema é possível graças ao tamanho reduzido do *hardware* projetado e ao baixo consumo das unidades integrantes.

Um projeto inovador, este circuito integrado multifuncional atende de forma completa a necessidade de chips com arquiteturas flexíveis e portabilidade para as mais diversas aplicações em telecomunicações.

2.1 Descrição do Microprocessador

2.1.1 Especificações Básicas

O microprocessador associado ao projeto SCW é interligado às três interfaces de comunicação previstas, de modo a controlar todo o fluxo de dados do sistema. A função do processador abrange tanto a parte de aquisição, tratamento e envio dos dados a serem trafegados, como o gerenciamento da transmissão e controle do enlace de comunicação RF.

As restrições que o projeto apresenta, como área reduzida e baixo consumo, foram os parâmetros básicos para detalhamento da especificação. Foi desenvolvida em conjunto pelas equipes de *hardware* e *software*, a especificação descrita pelos parâmetros abaixo[4],[6]:

- Processador RISC, 16-bits e 250 MHz de *clock* com tecnologia 0.8μm CMOS;
- 3,3 V de Tensão de Alimentação;
- Unidade Lógica e Aritmética (ULA) operando em ponto fixo;
- Banco de Registradores com 16 unidades de 16-bits;
- Unidade de Memória de 8KB;
- Ponteiro para rotina de interrupção: $\#0000_h$;
- Endereços de I / O: mapeados em memória (Tabela 2.1);
- Multiplicação por *software*;
- Endereço de inicialização: $\#0001_h$.

A Figura 2.1 representa a estrutura básica do sistema SCW.

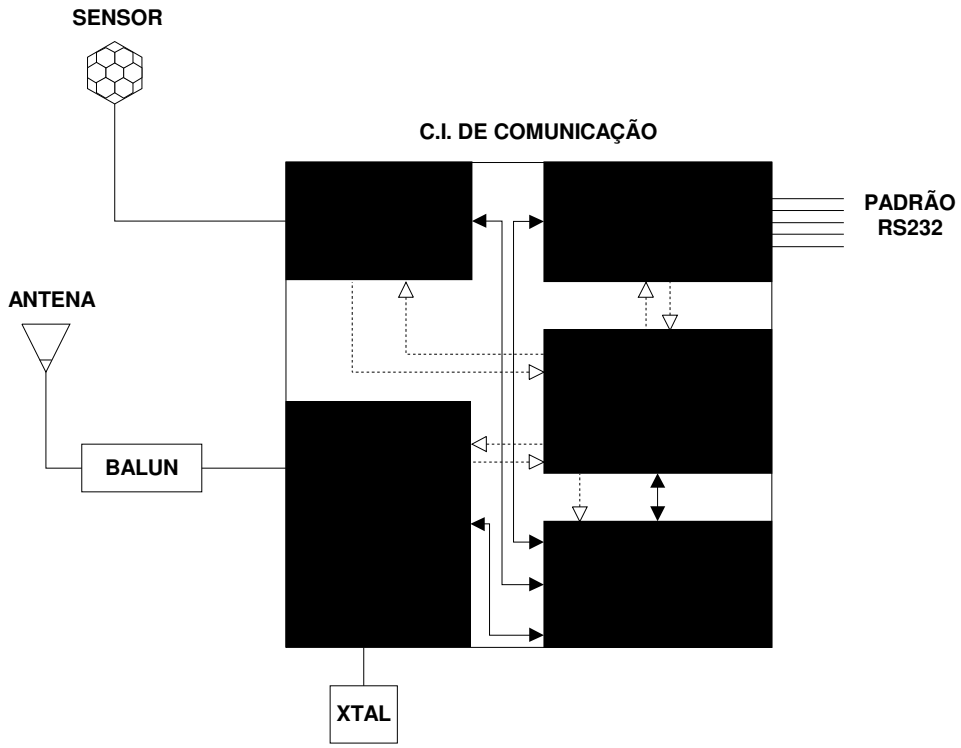


Figura 2.1: Estrutura básica do sistema SCW

2.1.2 Hardware

2.1.2.1 Memória

O sistema possui um único banco de memória para dados e instruções com capacidade total de 8KB (SRAM). O endereçamento é feito à palavra de 16 bits, sendo possíveis 4K posições de memória endereçadas por 12 bits. O sistema de endereçamento é feito por apontamento indireto, no qual todo endereço de memória é acessado por meio de um registrador que contém o endereço de destino. Assim, quando se deseja acessar uma posição específica de memória, carrega-se o valor num registrador, e por meio de uma instrução de leitura ou escrita, tem-se acesso à informação contida naquela posição de memória.

Considerando-se que o tamanho de palavra adotado para o SCW é de 16 bits, é possível endereçar até 64K posições possibilitando expansões futuras no bloco de memória inicial de 8KB sem maiores modificações na estrutura de *hardware* proposta.

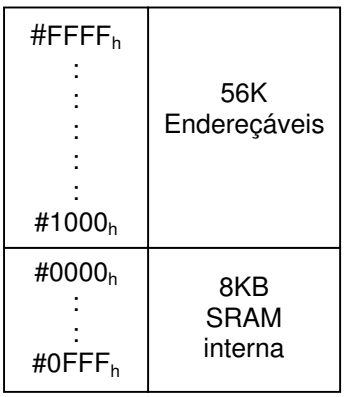


Figura 2.2: Esquemático da memória do sistema SCW

Uma parte da memória está reservada para uso das configurações do módulo de RF e interfaces. Esta parte da memória, embora seja endereçada normalmente, armazena, sempre no mesmo endereço, as configurações necessárias à operação das interfaces. Para permitir o acesso direto e dedicado da unidade de processamento, estas posições são ligadas diretamente às interfaces (*wired*). O processador, durante a transmissão, armazena dados nas posições de memória previstas para dados e, em seguida, carrega as informações de *setup* de transmissão para as interfaces nas áreas destinadas. As interfaces então seguem as instruções de configuração armazenadas e realizam a transmissão dos dados.

O processo de aquisição de informações se inicia quando a interface disponibiliza um dado, que é armazenado na área específica de dados de entrada para aquela interface. A interface então sinaliza ao processador a necessidade de armazenamento de um dado, por meio de uma interrupção. O processador, que checa periodicamente o conteúdo do registrador $int, percebe a chegada de dados e busca a informação área onde a interface os deixou e dá a destinação correta dentro da memória.

O sistema numérico adotado pelo processador utiliza números inteiros de -32767 até 32768 representados em complemento de dois.

2.1.2.2 I/O

A arquitetura de I/O do processador considera 12 posições de memória dedicadas, tratadas pela aplicação como registradores, para comunicação com as interfaces das unidades de RF, de Comunicação Serial e de conversão analógico/digital ΣΔ, que são acessadas através das instruções Lw (*Load Word*) e Sw (*Save Word*). Os registradores são mapeados por endereços de memória não abrangidos pela RAM de 8 KB. Os endereços e a destinação de cada registro estão descritos na Tabela 2.1.

Tabela 2.1: Posições reservadas em memória para I/O

Conteúdo	Unid. de RF	Unid. de Com Serial	Interface ΣΔ
Setup	$\#FFFF_h$ e $\#FFFE_h$	$\#FFFA_h$	$\#FFF6_h$
Status	$\#FFFD_h$	$\#FFF9_h$	$\#FFF5_h$
Dados (Transmissão)	$\#FFFC_h$	$\#FFF8_h$	-
Dados(Recepção)	$\#FFFB_h$	$\#FFF7_h$	$\#FFF4_h$

As informações salvas nos Registradores de Dados (Transmissão) são carregadas a medida que o processador solicita o envio. A própria operação de escrita no registrador deve solicitar a leitura do mesmo pela interface. As informações são enviadas de acordo com os parâmetros colocados nos registradores de *setup*, conforme o padrão da Figura 2.3. Após ser escrito, o dado é carregado na interface, que sinaliza, por meio do registrador de *status*, que está pronta para receber nova informação. A aquisição de dados originados nas interfaces é seguida de um sinal de interrupção para o processador, quando este está habilitado, indicando a necessidade de tratamento dos dados. O controle de interrupção envia, então, uma sinalização às interfaces indicando o tratamento de uma interrupção e bloqueando a recepção dos pedidos.

Ao final do tratamento da interrupção, os pedidos são desbloqueados e as interfaces são liberadas para encaminhar novas requisições. Este bloqueio é feito por meio do Bit 0 do registro $\#FFE3_h$, que desabilita, quando ativo, a entradas dos pedidos de interrupção no controlador. A rotina de tratamento de exceções deve limpar esse

registro, após o tratamento da interrupção corrente, habilitando novamente a ocorrência de exceções.

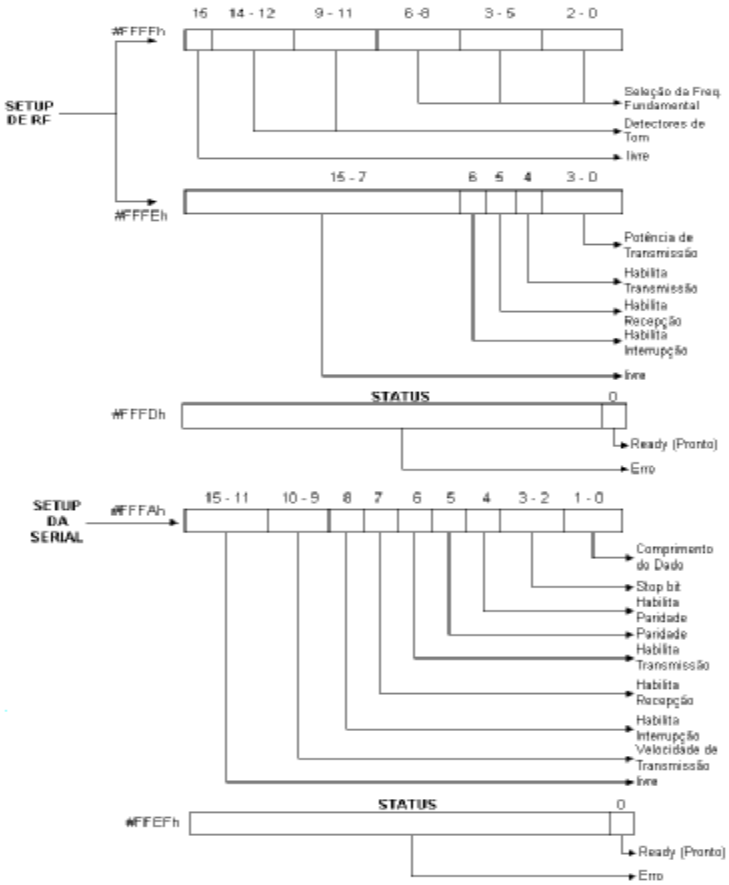


Figura 2.3: Mapeamento dos bits das palavras de *setup* e *status* para as interfaces

2.1.2.3 Registradores

Uma das especificações de projeto foi a utilização de um banco com 16 registradores de 16 bits. Foi escolhida a estratégia de definição de alguns registradores com propósito específico a serem usados no controle da execução, e alguns com flexibilidade suficiente para serem de uso geral, empregados no armazenamento de dados. Os registradores são identificados pela unidade de controle e pela ULA por seu código binário, de 0000 a 1111. A Tabela 2.2 descreve o conjunto de registradores do SCW.

Tabela 2.2: Registradores presentes no sistema. Uso e conteúdo.

Código	Símbolo	Significado	Observações
0000	$zero	Constante zero	Na verdade não é um registrador, mas apenas linhas de dados aterradas para sinalizar zero.
0001	$t0	Temporários	Usados para auxiliar o cálculo
0010	$t1		
0011	$t2		
0100	$a0	Argumento	Recebe os argumentos para cálculo
0101	$a1		
0110	$a2		
0111	$s0	Salvos	Guarda os resultados
1000	$s1		
1001	$s2		
1010	$s3		
1011	$int	Cód. Interrupção	Armazena o código de erro gerado pelo processador
1100	$gp	Apontador global	Aponta o topo da pilha geral
1101	$sp	Apontador de pilha	Aponta o topo da pilha de dados

Código	Símbolo	Significado	Observações
1110	$pc	Contador de programa	Contém a linha do programa que está sendo executada
1111	$ra	Endereço de retorno	Contém o endereço de retorno de uma sub-rotina

2.1.2.4 Interrupções

O sistema de processamento do SCW comporta 3 tipos de interrupção para interfaces de comunicação, além de duas sinalizações de erro, a saber:

- recepção de dado pela porta serial (Int = 1);
- recepção de dado pela unidade de RF (Int = 2);
- recepção de dado pela interface A/D (Int = 3);
- erro de *overflow* (ULA);
- erro de endereçamento (Memória).

As sinalizações geradas pelas interfaces de comunicação ocorrem quando da solicitação de leitura de dados enviados pelas mesmas, como forma de indicar ao processador que o dado encontra-se pronto para tratamento. Também podem ser geradas numa situação de transmissão onde a interface sinaliza ao processador o término do processo.

O erro ocasionado por *overflow* aritmético acontece quando o resultado de uma operação em complemento a dois, não pode ser comportado em 16 bits de determinado registrador. Neste caso, a unidade de controle interrompe o programa para tratamento da exceção ocorrida.

Os erros de endereçamento acontecem sempre que se tenta utilizar, como posições de armazenamento de instruções, as externas aos 8K de memória básicos do sistema desenvolvido. Isto pode acontecer com um desvio incondicional para uma posição de memória inválida ou, ainda, com um desvio condicional com *offset* referente à área de memória não utilizada para instruções.

O registrador $int contém o código de cada interrupção e o endereço da instrução executada quando aconteceu a interrupção, como mostrado no diagrama da Figura 2.4. Os dois bits menos significativos codificam os pedidos das interfaces de comunicação, conforme o código Int apresentado acima, enquanto que os bits dois e três alertam para erros no processamento de instruções. Os demais fazem o registro do endereço da instrução durante a qual ocorreu a solicitação.

Surgindo um pedido de interrupção o sistema é desviado para o endereço $\#0000_h$ da memória, no qual uma instrução de desvio incondicional faz a transferência para rotina de tratamento. Essa rotina identificará que tipo de interrupção foi gerado e procederá ao tratamento específico retomando, em seguida, a seqüência normal do programa.

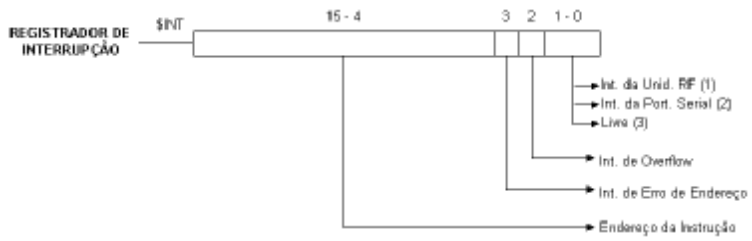


Figura 2.4: Conteúdo do registrador $int

2.1.2.5 Unidade Lógica Aritmética

A ULA é a parte do microprocessador que realiza operações aritméticas e lógicas. Qualquer instrução que demande uma alteração ou teste de um dado utiliza a ULA para sua execução.

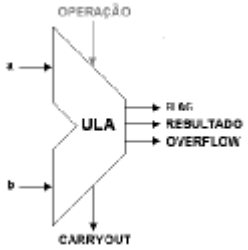


Figura 2.5: ULA de 1 Bit

A partir da análise de estruturas mais simples, tais como uma ULA de 1 bit até o nível mais complexo de uma ULA de 16 bits, construiu-se a estrutura básica da ULA que deve operar no microprocessador do SCW [5], mostrada na Figura 2.6. A ULA é responsável pela verificação e geração do sinal de *overflow* aritmético.

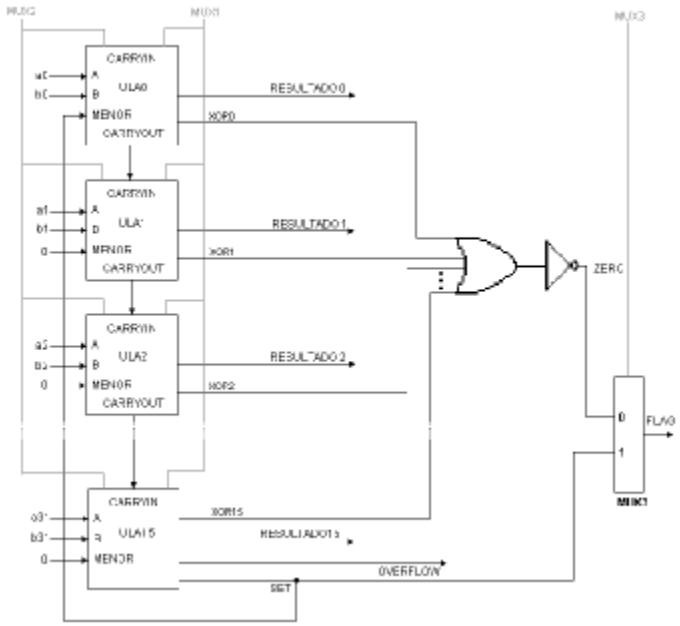


Figura 2.6: ULA de 16 bits desenvolvida para o SCW

2.1.2.6 A Unidade de Controle

A unidade de controle é responsável por comandar, controlar e verificar o funcionamento do processador. Traduz as instruções contidas em código de máquina em sinais de controle que acionam e configuram o sistema para que atenda às necessidades do *software*. Também interpreta os sinais de exceção, as interrupções e os estados de máquina para "decidir" qual a próxima instrução a ser executada. Embora todo o comando esteja na unidade de controle, o gerenciamento de alto nível permanece na aplicação, que através das instruções determina as diretrizes de funcionamento.

A Figura 2.7 representa a unidade de controle projetada pela equipe de *hardware* [4] para o sistema SCW.

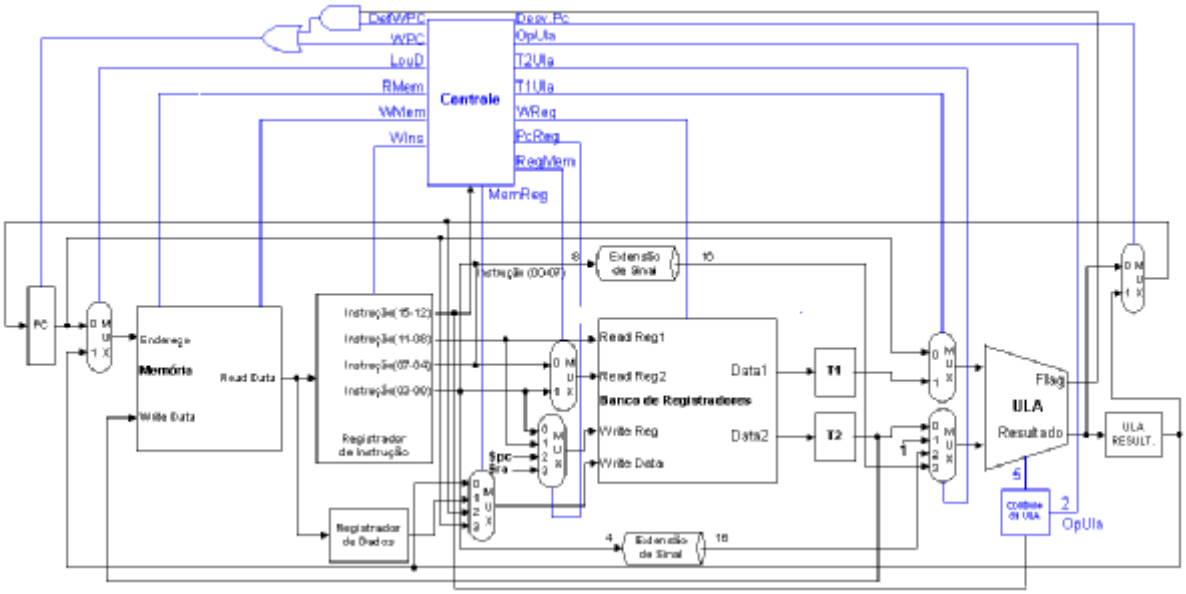


Figura 2.7: Esquemático completo da unidade de controle de sistema

2.1.3 Software [6]

No âmbito do conceito da arquitetura RISC, adotada pelo sistema, é utilizado um *set* de instruções reduzido, e a partir deste *set* são implementadas instruções mais complexas, que possam atender aos requisitos de processamento. O *set* de microinstruções realiza as operações básicas. O conjunto de instruções mais complexo é chamado de conjunto de pseudoinstruções, pelo fato de não representarem instruções válidas para o processador, que possam ser convertidas diretamente em código de máquina; mas sim conjuntos de microinstruções com tarefas específicas. Cada pseudoinstrução na verdade corresponde a uma única ou uma seqüência de microinstruções formando uma nova instrução.

A linguagem de máquina para o SCW foi desenvolvida em conjunto pelas equipes de *hardware* e *software* visando a integração do conjunto, que possibilita melhor desempenho.

2.1.3.1 As Microinstruções

O set de instruções elaborado para o SCW possui 3 formatos possíveis:

R - com três registradores;
I - com dois registradores e uma constante de quatro bits;
J - com um registrador e uma constante de oito bits.

A instrução do tipo R possui 4 campos: “Operação” que contém a codificação que identifica a instrução; “Registrador de Origem” que identifica o registrador de origem; “Registrador de Origem 2” que identifica o segundo registrador de origem; e “Registrador de Destino” que identifica o registrador de destino do resultado.

As instruções do tipo I são constituídas também de quatro campos: “Operação”, “Registrador de Origem”, “Registrador de Origem2” e “Deslocamento”. Tipicamente o campo “Deslocamento” armazena uma constante com a informação do salto, a partir do $pc, constituindo um endereço para a qual a execução deve ser transferida. O desvio pode ser realizado com *offset* de apenas 7 palavras a partir do $pc.

As instruções do tipo J possuem 3 campos, “Operação”, “Registrador de

Origem" e "Endereço". Os campos "Operação" e "Registrador de Origem" possuem 4 bits cada. Assim sendo, a instrução pode acessar um registrador com um endereço base com dezesseis bits e fornecer um deslocamento a partir da base de oito bits.

Nas Figura 2.8, Figura 2.9 e Figura 2.10 se pode ver de maneira esquemática os três formatos de instrução que o projeto utiliza.

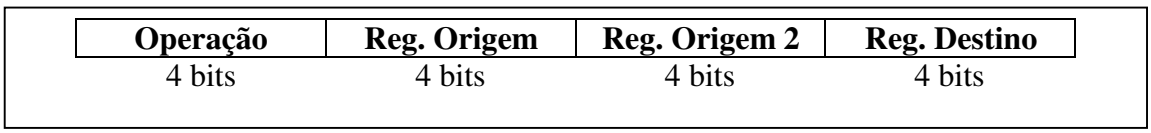


Figura 2.8: Formato de instrução do tipo R

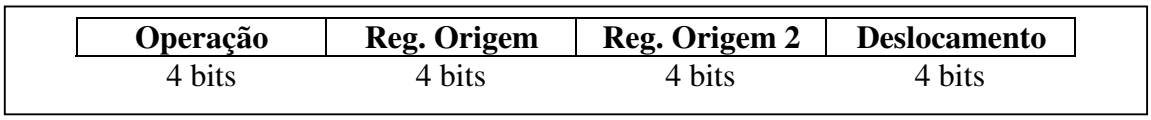


Figura 2.9: Formato da instrução do tipo I

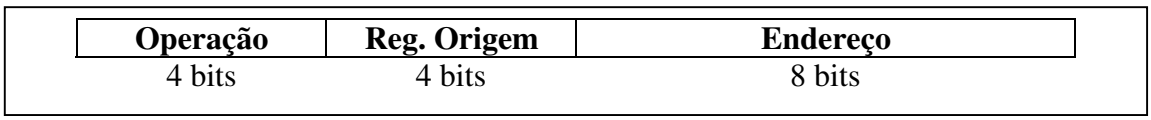


Figura 2.10:Formato da instrução do tipo J

A seleção das microinstruções a serem utilizadas leva em consideração os princípios de projeto, as limitações da arquitetura e, principalmente, as necessidades da aplicação. Todas as instruções necessárias à aplicação devem ser possíveis de serem implementadas, quer como microinstruções, quer como pseudoinstruções. Foi desenvolvido um *set* de 16 microinstruções que realizam as operações mais básicas do sistema diretamente. As microinstruções podem se divididas em quatro categorias básicas: Aritmética, Lógica, Transferência e Desvio.

A Tabela 2.3 mostra o *set* básico de microinstruções definido para o SCW.

Tabela 2.3: Conjunto de microinstruções para o SCW.

Cód	Cat	Inst	Exemplo	Significado	F
0010	Aritmética	Add	Add $s1,$s2,$s3	Adiciona $s2 a $s3 e armazena em $s1	R
0011		Sub	Sub $s1,$s2,$s3	Subtrai $s3 de $s2 e armazena em $s1	R
1000		Addi	Addi $s1,100	Adiciona $s1 a constante e armazena em $s1	J
1001	Lógica	Shift	Sft $s1,100	Desloca $s1 do valor da constante e armazena em $s1. Se o valor da constante for negativo desloca à esquerda, se for positivo desloca à direita	J
0100		And	And $s1,$s2,$s3	AND booleano bit a bit entre $s2 e $s3 e armazena em $s1	R
0101		Or	Or $s1,$s2,$s3	OR booleano bit a bit entre $s2 e $s3 e armazena em $s1	R
1010		Not	Not $s1	NOT booleano bit a bit de $s1 e armazena em $s1	J
0110		Xor	Xor $s1,$s2,$s3	XOR booleano bit a bit de $s2 e $s3 e armazena em $s1	R

Cód	Cat	Inst	Exemplo	Significado	F
0000	Transferência	Lw	Lw $s1,$s2,$s3	Carrega palavra armazenada no endereço $s2 deslocado de $s3 e salva em $s1	R
0001		Sw	Sw $s1,$s2,$s3	Carrega palavra armazenada em $s1 e salva no endereço $s2 deslocado de $s3	R
1011		Lui	Lui $s1,100	Carrega a constante nos oito bits mais significativos de $s1	J
0111	Desvio Cond.	Slt	Slt $s1,$s2,$s3	Torna $s1=1 se $s2 < $s3 senão $s1=0	R
1100		Beq	Beq $s1,$s2,5	Se $s1 = $s2 desvia para PC constante	I
1101		Blt	Blt $s1,$s2,5	Se $s1 < $s2 desvia para PC constante	I
1110	Desvio Incond.	J	J $s1,100	Desvia para o endereço $s1 deslocado da constante	J
1111		Jal	Jal $s1,100	Desvia para o endereço $s1 deslocado da constante salvando origem em $ra	J

Algumas instruções são facilmente compreendidas na tabela acima como: Add, Sub, And, Or Lw, Sw, e Slt. As demais instruções são descritas em detalhes a seguir.

Add – Instrução de soma com constante de oito bits.

Esta instrução é utilizada para armazenar uma constante de até oito bits nos bits menos significativos de um registrador. Pode também ser utilizada em conjunto com a instrução Lui para armazenar uma constante de até dezesseis bits.

Shift – Instrução de deslocamento lateral.

Usada para promover um deslocamento lateral no conteúdo do registrador indicado em "Registrador de Origem". O campo "Endereço" armazena uma constante que possui significação composta indica a direção e também a quantidade do deslocamento. Se o valor da constante for negativo o deslocamento será à esquerda em seu valor absoluto e se for positivo o deslocamento será à direita neste mesmo valor.

Lui – Instrução de carregamento de constante nos oito bits mais significativos de um registrador.

A instrução carrega esta constante nos bits mais significativos do registrador indicado preenchendo com oito zeros os bits menos significativos.

Beq e Blt – Instruções de desvio condicional.

Estas instruções comparam o conteúdo de dois registradores e desviam o programa para o endereço atual (armazenado em $pc), acrescido do valor da constante indicada se o teste resultar verdadeiro. A constante pode possuir valor negativo.

J e Jal – Instruções de desvio incondicional.

São utilizadas para promover desvios no programa independente de condições. Possuem três campos: "Operação", "Registrador de Origem" e "Endereço". O primeiro identifica a operação (J ou Jal) o segundo identifica o registrador que contém o endereço base e o terceiro contém uma constante que indica o deslocamento a partir do endereço base com até 255 palavras ou 510 bytes de distância.

O que diferencia as duas instruções é que a instrução Jal salva o endereço atual no registrador $ra, de forma a possibilitar o retorno.

As estimativas de ciclos de *clock* a serem gastos para realização das microinstruções foi realizada na etapa de desenvolvimento da linguagem para o SCW [6] e são mostradas na Tabela 2.4.

Tabela 2.4: Ciclos de *clock* e tempo gasto para realização das microinstruções

Cód	Categoria	Microinstrução	Ciclos	Tempo
0000	Aritmética	Add	4	16ns
0001		Sub	4	16ns
0010		Addi	4	16ns
0011	Lógica	Shift	4	16ns
0100		And	4	16ns
0101		Or	4	16ns
0110		Not	4	16ns
0111		Xor	4	16ns
1000	Transferência	Lw	5	20ns
1001		Sw	4	16ns
1010		Lui	4	16ns
1011	Desvio Cond.	Slt	4	16ns
1100		Beq	3	12ns
1101		Blt	3	12ns
1110	Desvio Incond.	J	3	12ns
1111		Jal	4	16ns

2.1.3.2 As Pseudoinstruções

O conjunto completo de pseudoinstruções inclui todas as microinstruções e acrescenta aquelas que não puderam ser implementadas, mas são necessárias à aplicação. Para o SCW foi definido um conjunto de sessenta e quatro pseudoinstruções distribuídas pelas mesmas cinco categorias utilizadas nas microinstruções da seguinte forma: nove aritméticas, onze lógicas, treze de transferência, dezessete de desvio condicional, dez de desvio incondicional e quatro reservadas para uso futuro.

As tabelas 2.4, 2.5, 2.6, 2.7 e 2.8 mostram as sessenta e quatro instruções (microinstruções e pseudoinstruções) desenvolvidas para o projeto SCW.

Tabela 2.5: Pseudoinstruções aritméticas

Cat	Instrução	Significado
Aritmética	Add $s1,$s2,$s3	Adiciona $s2 a $s3 e armazena em $s1
	Sub $s1,$s2,$s3	Subtrai $s3 de $s2 e armazena em $s1
	Mul $s1,$s2,$s3	Multiplica $s2 por $s3 e armazena em $s1
	Div $s1,$s2,$s3	Divide $s2 por $s3 e armazena em $s1
	Addi $s1,100	Adiciona $s1 a constante e armazena em $s1
	Subi $s1,100	Subtrai $s1 de uma constante e armazena em $s1
	Muli $s3,$s2,100	Multiplica $s2 por uma constante e armazena em $s3
	Divi $s3,$s2,100	Divide $s2 por uma constante e armazena em $s3
	Rem $s1,$s2	Armazena o resto da divisão de $s1 por $s2 em $s1

Tabela 2.6: Pseudoinstruções lógicas

Cat	Instrução	Significado
Lógica	Sftl $s1,100	Desloca a esquerda $s1 do valor da constante e armazena em $s1.
	Sftr $s1,100	Desloca a direita $s1 do valor da constante e armazena em $s1.
	And $s1,$s2,$s3	AND booleano bit a bit entre $s2 e $s3 e armazena em $s1
	Or $s1,$s2,$s3	OR booleano bit a bit entre $s2 e $s3 e armazena em $s1
	Nor $s1,$s2,$s3	NOR booleano bit a bit entre $s2 e $s3 e armazena em $s1
	Xor $s1,$s2,$s3	XOR booleano bit a bit entre $s2 e $s3 e armazena em $s1
	Not $s1	NOT booleano bit a bit de $s1 e armazena em $s1
	Comp $s1	Complemento a dois de $s1 e armazena em $s1
	Andi $s1,100	AND booleano bit a bit entre $s1 e uma constante e armazena em $s1
	Ori $s1,100	OR booleano bit a bit entre $s1 e uma constante e armazena em $s1
	Xori $s1,100	XOR booleano bit a bit entre $s1 e uma constante e armazena em $s1

Tabela 2.7: Pseudoinstruções de Transferência

Cat	Instrução	Significado
Transferência	Lw $s1,(100)$s2	Carrega em $s1 a palavra armazenada no endereço $s2 deslocado da constante.
	Sw $s1,(100)$s2	Salva a palavra armazenada em $s1 no endereço $s2 deslocado da constante.
	Lb $s1,(100)$s2	Carrega em $s1 o byte menos significativo armazenado no endereço $s2 deslocado da constante.
	Sb $s1,(100)$s2	Salva o byte menos significativo de $s1 no endereço $s2 deslocado da constante.
	Ld $s1,(100)$s3	Carrega em $s1 e $s2 duas palavras armazenadas no endereço $s3 deslocado da constante e o próximo.
	Sd $s1,(100)$s3	Salva o conteúdo de $s1 e $s2 no endereço $s3 deslocado da constante.
	Lui $s1,100	Carrega a constante nos oito bits mais significativos de $s1.
	Lwi $s1,100	Carrega uma constante de 16 bits num registrador.
	Mov $s1,$s2	Move o conteúdo de $s1 para $s2.
	Mflo $s1	Move o byte menos significativo de $s1 para o mais significativo.
	Mfhi $s1	Move o byte mais significativo de $s1 para o menos significativo.
	Move $s1, $s2	Move o conteúdo de $s2 para $s1 preenchendo $s2 com 0
	Chg $s1, $s2	Troca o conteúdo de dois registradores

Tabela 2.8: Pseudoinstruções de Desvio Condicional

Cat	Instrução	Significado
Desvio Condicional	Slt $s1,$s2,$s3	Torna $s1=1 se $s2 < $s3 senão $s1=0
	Sle $s1,$s2,$s3	Torna $s1=1 se $s2 < ou = $s3 senão $s1=0
	Seq $s1,$s2,$s3	Torna $s1=1 se $s2 = $s3 senão $s1=0
	Sne $s1,$s2,$s3	Torna $s1=1 se $s2 < > $s3 senão $s1=0
	Sgt $s1,$s2,$s3	Torna $s1=1 se $s2 > $s3 senão $s1=0
	Sge $s1,$s2,$s3	Torna $s1=1 se $s2 > ou = $s3 senão $s1=0

	Beq $s1,$s2,5	Se $s1 = $s2 desvia para PC constante
	Bne $s1,$s2,5	Se $s1 <> $s2 desvia para PC constante
	Blt $s1,$s2,5	Se $s1 < $s2 desvia para PC constante
	Bgt $s1,$s2,5	Se $s1 > $s2 desvia para PC constante
	Slti $s1,$s2,100	Torna $s1=1 se $s2 < constante senão $s1=0
	Seqi $s1,$s2,100	Torna $s1=1 se $s2 = constante senão $s1=0
	Sgti $s1,$s2,100	Torna $s1=1 se $s2 > constante senão $s1=0
	Beqi $s1,100,5	Se $s1 = constante 1 desvia para PC constante 2
	Bnei $s1,100,5	Se $s1 <> constante 1 desvia para PC constante 2
	Blti $s1,100,5	Se $s1 < constante 1 desvia para PC constante 2
	Bgti $s1,100,5	Se $s1 > constante 1 desvia para PC constante 2

Tabela 2.9: Pseudoinstruções de Desvio Incondicional

Cat	Instrução	Significado
Desvio Incondicional	J $s1,100	Desvia para o endereço $s1 deslocado da constante de oito bits
	Jr $s1	Desvia para o endereço $s1
	Jpc 100	Desvia para o endereço $pc deslocado da constante de oito bits
	Jal $s1,100	Desvia para o endereço $s1 deslocado da constante de oito bits salvando origem em $ra
	Jalr $s1,$s2,100	Desvia para o endereço $s2 deslocado da constante de oito bits salvando origem em $s1
	Jalpc 100	Desvia para o endereço $pc deslocado da constante de oito bits salvando origem em $ra
	Jd 100	Desvia para o endereço da constante de 16 bits
	Jald $s1,100	Desvia para o endereço $s1 deslocado da constante de 16 bits salvando origem em $ra
	Jalrd $s1,$s2,100	Desvia para o endereço $s2 deslocado da constante de 16 bits salvando origem em $s1
	Jalpcd 100	Desvia para o endereço $pc deslocado da constante de 16 bits salvando origem em $ra

2.1.3.3 O Montador [8]

O montador é o *software* que fará a conversão da linguagem de máquina para os códigos em binário interpretáveis pelo processador, fazendo com que a aplicação, projetada no nível de linguagem de máquina, possa ser executada pelo sistema. Em outras palavras, o montador permite que a abstração de um *set*, com sessenta e quatro instruções de trinta e dois bits, seja executada pelo SCW em dezesseis instruções de dezesseis bits.

Esta montagem é feita utilizando padrões de conversão preestabelecidos, que fazem com que para a cada pseudoinstrução, seja atribuída uma seqüência de microinstruções que realize a atividade desejada. Para o SCW tem-se as tabelas de 2.9 a 2.69 para conversão de pseudoinstruções [6]. A primeira linha representa a pseudoinstrução original e as subseqüentes representam a seqüência de microinstruções correspondente.

Tabela 2.10: Pseudoinstrução Add

Add	**$s3,**	**$s1,**	**$s2**	**Soma o valor de $s1 com $s2 e armazena em $s3**
Add	$s3,	$s1,	$s2	Operação direta da microinstrução

Tabela 2.11: Pseudoinstrução Sub

Sub	**$s3,**	**$s1,**	**$s2**	**Subtrai o valor de $s2 de $s1 e armazena em $s3**
Sub	$s3,	$s1,	$s2	Operação direta da microinstrução

Tabela 2.12: Pseudoinstrução Mul

Mul	**$s3,**	**$s1,**	**$s2**	**Multiplica o valor de $s1 por $s2 e armazena em $s3 versão 3**
Add	$s3,	$zero,	$zero	Armazena zero em $s3
Add	$t2,	$Zero,	$Zero	Armazena zero em $t2
Addi	$t2,	1		Armazena um em $t2
And	$t1,	$s1,	$t2	Calcula e booleano de $t2 e $s1 e armazena em $t1
Beq	$t1,	$Zero,	1	Se $t1 for zero salta uma linha
Add	$s3,	$s3,	$s2	Soma $s3 com $s2 três armazena em $s3
Sft	$s1,	1		Desloca $s1 um bit à direita
Sft	$s2,	-1		Desloca $s2 um bit à esquerda
Beq	$s1,	$zero,	1	Se $s1 for zero salta uma linha
J	$pc,	-6		Volta seis linhas

Tabela 2.13: Pseudoinstrução Div

Div	**$s3,**	**$s1,**	**$s2**	**Divide o valor de $s2 por $s1 e armazena em $s3**
Add	$s3,	$zero,	$zero	Armazena zero em $s3
Slt	$t1,	$s1,	$s2	Seta $t1 se $s1 for menor que $s2
Sft	$s3,	-1		Desloca $s3 um bit à esquerda
Beq	$t1,	$zero,	1	Se $t1 for zero salta uma linha
Add	$s3,	$s3,	$t1	Soma $s3 com $t1 três armazena em $s3
Sft	$s2,	1		Desloca $s2 um bit à direita
Beq	$s2,	$zero,	1	Se $s2 for zero salta uma linha
J	$pc,	-6		Volta seis linhas

Tabela 2.14: Pseudoinstrução Addi

Addi	**$s1,**	**100**	**Soma o valor de $s1 com uma constante e armazena em $s1**
Addi	$s1,	100	Operação direta da microinstrução

Tabela 2.15: Pseudoinstrução Subi

Subi	**$s1,**	**100**	**Subtrai o valor de uma constante de $s1 e armazena em $s1**
Add	$t1,	$Zero,	$Zero	Armazena zero em $t1
Addi	$t1,	100		Armazena a constante em $t1
Sub	$s1,	$s1,	$t1	Efetua a subtração

Tabela 2.16: Pseudoinstrução Muli

Muli	**$s3,**	**$s2,**	**100**	**Multiplica o valor de $s2 por uma constante e armazena em $s3**
Add	$s1,	$zero,	$zero	Armazena zero em $s1
Addi	$s1,	100		Armazena a constante em $s1
Add	$s3,	$zero,	$zero	Armazena zero em $s3
Add	$t2,	$zero,	$zero	Atribui o valor “zero” a $t2
Addi	$t2,	1		Armazena um em $t2
And	$t1,	$s1,	$t2	Calcula e booleano de $t2 e $s1 e armazena em $t1
Beq	$t1,	$Zero,	1	Se $t1 for zero salta uma linha
Add	$s3,	$s3,	$s2	Soma $s3 com $s2 três armazena em $s3
Sft	$s1,	1		Desloca $s1 um bit à direita
Sft	$s2,	-1		Desloca $s2 um bit à esquerda
Beq	$s1,	$zero,	1	Se $s1 for zero salta uma linha
J	$pc,	-6		Volta seis linhas

Tabela 2.17: Pseudoinstrução Divi

Divi	**$s3,**	**$s2,**	**100**	**Divide o valor de $s2 por uma constante e armazena em $s3**
Add	$s1,	$Zero,	$Zero	Armazena zero em $s1

Addi	$s1,	100		Armazena a constante em $s1
Add	$s3,	$zero,	$zero	Armazena zero em $s3
Slt	$t1,	$s1,	$s2	Seta $t1 se $s1 for menor que $s2
Sft	$s3,	-1		Desloca $s3 um bit à esquerda
Beq	$t1,	$zero,	1	Se $t1 for zero salta uma linha
Add	$s3,	$s3,	$t1	Soma $s3 com $t1 três armazena em $s3
Sft	$s2,	1		Desloca $s2 um bit à direita
Beq	$s2,	$zero,	1	Se $s2 for zero salta uma linha
J	$pc,	-6		Volta seis linhas

Tabela 2.18: Pseudoinstrução Rem

Rem	**$s1,**	**$s2**		**Armazena o resto da divisão de $s1 por $s2 em $s1**
Add	$s3,	$zero,	$zero	Armazena zero em $s3
Slt	$t1,	$s1,	$s2	Seta $t1 se $s1 for menor que $s2
Sft	$s3,	-1		Desloca $s3 um bit à esquerda
Beq	$t1,	$zero,	1	Se $t1 for zero salta uma linha
Add	$s3,	$s3,	$t1	Soma $s3 com $t1 três armazena em $s3
Sft	$s2,	1		Desloca $s2 um bit à direita
Beq	$s2,	$zero,	1	Se $s2 for zero salta uma linha
J	$pc,	-6		Volta seis linhas

Tabela 2.19: Pseudoinstrução Sftl

Sftl	**$s1,**	**100**	**Desloca $s1 à esquerda do valor da constante e armazena em $s1**
Sft	$s1,	-100	Operação direta da microinstrução atribuindo um valor negativo à constante

Tabela 2.20: Pseudoinstrução Sftr

Sftr	**$s1,**	**100**	**Desloca $s1 à direita do valor da constante e armazena em $s1**
Sft	$s1,	100	Operação direta da microinstrução atribuindo um valor positivo à constante

Tabela 2.21: Pseudoinstrução And

And	**$s1,**	**$s2,**	**$s3**	**AND booleano bit a bit entre $s2 e $s3 e armazena em $s1**
And	$s1,	$s2,	$s3	Operação direta da microinstrução

Tabela 2.22: Pseudoinstrução Or

Or	**$s1,**	**$s2,**	**$s3**	**OR booleano bit a bit entre $s2 e $s3 e armazena em $s1**
Or	$s1,	$s2,	$s3	Operação direta da microinstrução

Tabela 2.23: Pseudoinstrução Nor

Nor	**$s1,**	**$s2,**	**$s3**	**NOR booleano bit a bit entre $s2 e $s3 e armazena em $s1**
Or	$s1,	$s2,	$s3	Operação direta da microinstrução
Not	$s1			Operação direta da microinstrução

Tabela 2.24: Pseudoinstrução Xor

Xor	**$s1,**	**$s2,**	**$s3**	**XOR booleano bit a bit entre $s2 e $s3 e armazena em $s1**
Xor	$s1,	$s2,	$s3	Operação direta da microinstrução

Tabela 2.25: Pseudoinstrução Not

Not	**$s1**	**NOT booleano bit a bit de $s1 e armazena em $s1**

Not	$s1			Operação direta da microinstrução

Tabela 2.26: Pseudoinstrução Comp

Comp	**$s1**			**Complemento a dois de $s1 e armazena em $s1**
Sub	$s1,	$ Zero,	$s1	Subtrai $s1 de $Zero resultando no complemento a dois de $s1 e armazena em $s1

Tabela 2.27: Pseudoinstrução Andi

Andi	**$s1,**	**100**		**AND booleano bit a bit entre $s1 e uma constante e armazena em $s1**
Add	$t1,	$Zero,	$Zero	Armazena zero em $t1
Addi	$t1,	100		Armazena a constante em $t1
And	$s1	$s1,	$t1	Operação direta da microinstrução

Tabela 2.28: Pseudoinstrução Ori

Ori	**$s1,**	**100**		**OR booleano bit a bit entre $s1 e uma constante e armazena em $s1**
Add	$t1,	$Zero,	$Zero	Armazena zero em $t1
Addi	$t1,	100		Armazena a constante em $t1
Or	$s1	$s1,	$t1	Operação direta da microinstrução

Tabela 2.29: Pseudoinstrução Xori

Xori	**$s1,**	**100**		**XOR booleano bit a bit entre $s1 e uma constante e armazena em $s1**
Add	$t1,	$Zero,	$Zero	Armazena zero em $t1
Addi	$t1,	100		Armazena a constante em $t1
Xor	$s1	$s1,	$t1	Operação direta da microinstrução

Tabela 2.30: Pseudoinstrução Lw

Lw	**$s1,**	**$s2,**	**$s3**	**Carrega em $s1 a palavra armazenada no endereço $s2 deslocado do valor armazenado em $s3**
Lw	$s1,	$s2,	$s3	Operação direta da microinstrução

Tabela 2.31: Pseudoinstrução Sw

Sw	**$s1,**	**$s2,**	**$s3**	**Salva a palavra armazenada em $s1 no endereço $s2 deslocado do valor armazenado em $s3**
Sw	$s1,	$s2,	$s3	Operação direta da microinstrução

Tabela 2.32: Pseudoinstrução Lb

Lb	**$s1,**	**$s2,**	**$s3**	**Carrega em $s1 o *byte* menos significativo armazenado no endereço $s2 deslocado do valor armazenado em $s3**
Lw	$t1,	$s2,	$s3	Carrega palavra armazenada no endereço $s2 deslocado do valor armazenado em $s3 e salva em $t1
Sft	$t1,	-8		Desloca $t1 oito vezes para a esquerda
Sft	$t1,	8		Desloca $t1 oito vezes para a direita
Sft	$s1,	8		Desloca $s1 oito vezes para a direita
Sft	$s1,	-8		Desloca $s1 oito vezes para a esquerda
Add	$s1,	$s1,	$t1	Concatena os oito bits mais significativos com os oito bits menos significativos da palavra armazenada no endereço $s2 e guarda em $s1

Tabela 2.33: Pseudoinstrução Sb

Sb	**$s1,**	**$s2,**	**$s3**	**Salva o *byte* menos significativo de $s1 no endereço $s2 deslocado da constante do valor armazenado em $s3**
Add	$t1,	$Zero,	$s1	Carrega o valor armazenado em $s1 em $t1
Sft	$t1,	$Zero,	-8	Desloca o $t1 oito vezes para a esquerda
Sft	$t1,	$Zero,	+8	Desloca o $t1 oito vezes para a direita
Lw	$t2,	$s2,	$s3	Carrega palavra armazenada no endereço $s2 deslocado do valor armazenado em $s3 e salva em $t2
Sft	$t2,	$Zero,	+8	Desloca o $t2 oito vezes para a direita
Sft	$t2,	$Zero,	-8	Desloca o $t2 oito vezes para a esquerda
Add	$s1,	$t1,	$t2	Concatena os oito bits mais significativos de $s1 com os oito bits menos significativos da palavra armazenada no endereço $s2 e guarda em $s1
Sw	$s1,	$s2,	$s3	Salva a palavra armazenada em $s1 no endereço $s2 deslocado do valor armazenado em $s3

Tabela 2.34: Pseudoinstrução Ld

Ld	**$s1,**	**$s2,**	**(100)$s3**	**Carrega em $s1 e $s2 duas palavras armazenadas no endereço $s3 deslocado da constante e no próximo**
Add	$t1,	$Zero,	$Zero	Armazena zero em $t1
Addi	$t1,	100		Armazena o valor da constante em $t1
Lw	$s1,	$t1,	$s3	Carrega palavra armazenada no endereço $t1 deslocado do valor armazenado em $s3 e salva em $s1
Addi	$s3,	1		Soma um ao valor de $s3
Lw	$s2,	$t1,	$s3	Carrega palavra armazenada no endereço $t1 deslocado do valor armazenado em $s3 e salva em $s2

Tabela 2.35: Pseudoinstrução Sd

Sd	**$s1,**	**$s2,**	**(100)$s3**	**Salva o conteúdo de $s1 e $s2 no endereço $s3 deslocado da constante**
Add	$t1,	$Zero,	$Zero	Armazena zero em $t1
Addi	$t1,	100		Armazena o valor da constante em $t1
Sw	$s1,	$t1,	$s3	Salva a palavra armazenada em $s1 no endereço $t1 deslocado do valor armazenado em $s3
Addi	$s3,	1		Soma um ao valor de $s3
Sw	$s2,	$t1,	$s3	Salva a palavra armazenada em $s2 no endereço $t1 deslocado do valor armazenado em $s3

Tabela 2.36: Pseudoinstrução Lui

Lui	**$s1,**	**100**	**Carrega a constante nos oito bits mais significativos de $s1**
Lui	$s1,	100	Operação direta da microinstrução

Tabela 2.37: Pseudoinstrução Lwi

Lwi	**$s1,**	**100**	**Carrega uma constante de 16 bits num registrador**
Add	$s1,	$Zero, $Zero	Armazena o valor zero em $s1
Lui	$s1,	mais significativa	Carrega a parte mais significativa da constante em $s1
Addi	$s1,	menos significativa	Soma $s1 com a parte menos significativa da constante e armazena em $s1

Tabela 2.38: Pseudoinstrução Mov

Mov	**$s1,**	**$s2**	**Move o conteúdo de $s1 para $s2**
Add	$t1,	$s1, $Zero	Armazena o valor de $s1 em $t1
Add	$s1,	$s2, $Zero	Armazena o valor de $s2 em $s1
Add	$s2,	$t1, $Zero	Armazena o valor de $t1 em $s2

Tabela 2.39: Pseudoinstrução Mflo

Mflo	**$s1**			**Move o *byte* menos significativo de $s1 para o mais significativo**
Sft	$s1,	$Zero,	-8	Desloca o $s1 oito vezes para a esquerda

Tabela 2.40: Pseudoinstrução Mfhi

Mfhi	**$s1**			**Move o *byte* mais significativo de $s1 para o menos significativo**
Sft	$s1,	$Zero,	+8	Desloca o $s1 oito vezes para a direita

Tabela 2.41: Pseudoinstrução Move

Move	**$s1,**	**$s2**		**Move o conteúdo de $s2 para $s1 guardando zero em $s2**
Add	$s1,	$s2,	$Zero	Copia o conteúdo de $s2 para $s1
Add	$s1,	$Zero,	$Zero	Armazena zero em $s2

Tabela 2.42: Pseudoinstrução Chg

Chg	**$s1,**	**$s2**		**Move o conteúdo de $s2 para $s1 e o conteúdo de $s1 para $s2**
Add	$t1,	$s1,	$Zero	Copia o conteúdo de $s1 para $t1
Add	$s1,	$s2,	$Zero	Copia o conteúdo de $s2 para $s1
Add	$s2,	$t1,	$Zero	Copia o conteúdo de $t1 para $s2

Tabela 2.43: Pseudoinstrução Slt

Slt	**$s1,**	**$s2,**	**$s3**	**Torna $s1=1 se $s2 < $s3 senão $s1=0**
Slt	$s1,	$s2,	$s3	Operação direta da microinstrução

Tabela 2.44: Pseudoinstrução Sle

Sle	**$s1,**	**$s2,**	**$s3**	**Torna $s1=1 se $s2 < ou = $s3 senão $s1=0**
Slt	$s1,	$s3,	$s2	Torna $s1=1 se $s3 < $s2 senão $s1=0
Not	$s1			Inverte o conteúdo de $s1

Tabela 2.45: Pseudoinstrução Seq

Seq	**$s1,**	**$s2,**	**$s3**	**Torna $s1=1 se $s2 = $s3 senão $s1=0**
Add	$s1,	$Zero,	$Zero	Atribui o valor “zero” a $s1
Addi	$s1,	1		Atribui o valor “um” a $s1
Beq	$s1,	$s2,	2	Vai para o fim se $s1 for igual a $s2
Add	$s1,	$Zero,	$Zero	Armazena zero em $s1

Tabela 2.46: Pseudoinstrução Sne

Sne	**$s1,**	**$s2,**	**$s3**	**Torna $s1=1 se $s2 < > $s3 senão $s1=0**
Add	$s1,	$Zero,	$Zero	Armazena zero em $s1
Beq	$s1,	$s2,	2	Vai para o fim se $s1 for igual a $s2
Addi	$s1,	1		Atribui o valor “1” a $s1

Tabela 2.47: Pseudoinstrução Sgt

Sgt	**$s1,**	**$s2,**	**$s3**	**Torna $s1=1 se $s2 > $s3 senão $s1=0**
Slt	$s1,	$s3,	$s2	Operação direta da microinstrução

Tabela 2.48: Pseudoinstrução Sge

Sge	**$s1,**	**$s2,**	**$s3**	**Torna $s1=1 se $s2 > ou = $s3 senão $s1=0**
Slt	$s1,	$s2,	$s3	Operação direta da microinstrução
Not	$s1			Inverte o conteúdo de $s1

Tabela 2.49: Pseudoinstrução Beq

Beq	**$s1,**	**$s2,**	**5**	**Se $s1 = $s2 desvia para PC mais a constante**
Beq	$s1,	$s2,	5	Operação direta da microinstrução

Tabela 2.50: Pseudoinstrução Bne

Bne	**$s1,**	**$s2,**	**5**	**Se $s1 <> $s2 desvia para PC mais a constante**
Beq	$s1,	$s2,	2	Se $s1 = $s2 pula a próxima linha
J	$pc,	5		Desvia para Pc mais a constante

Tabela 2.51: Pseudoinstrução Blt

Blt	**$s1,**	**$s2,**	**5**	**Se $s1 < $s2 desvia para PC mais a constante**
Blt	$s1,	$s2,	5	Operação direta da microinstrução

Tabela 2.52: Pseudoinstrução Bgt

Bgt	**$s1,**	**$s2,**	**5**	**Se $s1 > $s2 desvia para PC mais a constante**
Blt	$s2,	$s1,	5	Operação direta da microinstrução

Tabela 2.53: Pseudoinstrução Slti

Slti	**$s1,**	**$s2,**	**100**	**Torna $s1=1 se $s2 < constante senão $s1=0**
Add	$s3,	$Zero,	$Zero	Armazena zero em $s3
Addi	$s3,	100		Armazena o valor da constante em $s3
Slt	$s1,	$s2,	$s3	Operação direta da microinstrução

Tabela 2.54: Pseudoinstrução Seqi

Seqi	**$s1,**	**$s2,**	**100**	**Torna $s1=1 se $s2 = constante senão $s1=0**
Add	$s3,	$Zero,	$Zero	Atribui o valor “zero” a $s3
Addi	$s3,	100		Atribui o valor da constante a $s3
Add	$s1,	$Zero,	$Zero	Atribui o valor “zero” a $s1
Addi	$s1,	1		Atribui o valor “um” a $s1
Beq	$s2,	$s3,	2	Vai para o fim se $s2 for igual a $s3
Add	$s1,	$Zero,	$Zero	Armazena zero em $s1

Tabela 2.55: Pseudoinstrução Sgti

Sgti	**$s1,**	**$s2,**	**100**	**Torna $s1=1 se $s2 > constante senão $s1=0**
Add	$s3,	$Zero,	$Zero	Atribui o valor “zero” a $s3
Addi	$s3,	100		Armazena o valor da constante em $s3
Slt	$s2,	$s1,	$s3	Operação direta da microinstrução

Tabela 2.56: Pseudoinstrução Beqi

Beqi	**$s1,**	**100,**	**5**	**Se $s1 = constante um desvia para PC mais a constante 2**
Add	$s2,	$Zero,	$Zero	Atribui o valor “zero” a $s2
Addi	$s2,	100		Armazena o valor da constante em $s2
Beq	$s1,	$s2,	5	Vai para $pc mais a constante se $s2 for igual a $s1

Tabela 2.57: Pseudoinstrução Bnei

Bnei	**$s1,**	**100,**	**5**	**Se $s1 <> constante 1 desvia para PC mais a constante 2**
Add	$s2,	$Zero,	$Zero	Atribui o valor “zero” a $s2
Addi	$s2,	100		Armazena o valor da constante em $s2
Beq	$s1,	$s2,	2	Pula a próxima linha se $s2 for igual a $s1
J	$pc,	5		Desvia para Pc mais a constante

Tabela 2.58: Pseudoinstrução Blti

Blti	**$s1,**	**100,**	**5**	**Se $s1 < constante 1 desvia para PC mais a constante 2**
Add	$s2,	$Zero,	$Zero	Atribui o valor “zero” a $s2
Addi	$s2,	100		Armazena o valor da constante em $s2
Blt	$s1	$s2	5	Operação direta da microinstrução

Tabela 2.59: Pseudoinstrução Bgti

Bgti	**$s1,**	**100,**	**5**	**Se $s1 > constante 1 desvia para PC mais a constante 2**
Add	$s2,	$Zero,	$Zero	Atribui o valor “zero” a $s2
Addi	$s2,	100+1		Armazena o valor da constante 1 mais 1 em $s2
Blt	$s1	$s2	5	Operação direta da microinstrução

Tabela 2.60: Pseudoinstrução J

J	**$s1,**	**100**	**Desvia para o endereço $s1 deslocado da constante de oito bits**
J	$s1,	100	Operação direta da microinstrução

Tabela 2.61: Pseudoinstrução Jr

Jr	**$s1**	**Desvia para o endereço $s1**
J	$s1,	0	Operação direta da microinstrução

Tabela 2.62: Pseudoinstrução Jpc

Jpc	**100**	**Desvia para o endereço $pc deslocado da constante de oito bits**
J	$pc,	100	Operação direta da microinstrução

Tabela 2.63: Pseudoinstrução Jal

Jal	**$s1,**	**100**	**Desvia para o endereço $s1 deslocado da constante de oito bits salvando origem em $ra**
Jal	$s1,	1000	Operação direta da microinstrução

Tabela 2.64: Pseudoinstrução Jalr

Jalr	**$s1,**	**$s2,**	**100**	**Desvia para o endereço $s2 deslocado da constante de oito bits salvando origem em $s1**
Jal	$s2,	100		Operação direta da microinstrução
Add	$s1,	$ra,	$Zero	Copia o valor de $ra para $s1

Tabela 2.65: Pseudoinstrução Jalpc

Jalpc	**100**	**Desvia para o endereço $pc deslocado da constante de oito bits salvando origem em $ra**
Jal	$pc,	100	Operação direta da microinstrução

Tabela 2.66: Pseudoinstrução Jd

Jd	**100**		**Desvia para o endereço da constante de 16 bits**
Add	$s1,	$Zero, $Zero	Armazena o valor zero em $s1
Lui	$s1,	mais significativa	Carrega $t1 com a parte mais significativa da constante
J	$s1,	menos significativa	Operação direta da microinstrução

Tabela 2.67: Pseudoinstrução Jald

Jald	**$s1,**	**100**	**Desvia para o endereço $s1 deslocado da constante de 16 bits salvando origem em $ra**
Add	$t1,	$Zero, $Zero	Armazena o valor zero em $t1
Lui	$t1,	mais significativa	Carrega $t1 com a parte mais significativa da constante
Add	$s1,	$s1, $t1	Soma o endereço base com $t1
Jal	$s1,	menos significativa	Operação direta da microinstrução

Tabela 2.68: Pseudoinstrução Jalrd

Jalrd	**$s1,**	**$s2,**	**100**	**Desvia para o endereço $s2 deslocado da constante de 16 bits salvando origem em $s1**
Add	$t1,	$Zero,	$Zero	Armazena o valor zero em $t1
Lui	$t1,	mais significativa		Carrega $t1 com a parte mais significativa da constante
Add	$s2,	$s2,	$t1	Soma o endereço base com $t1
Jal	$s2,	menos significativa		Operação direta da microinstrução
Add	$s1,	$ra,	$Zero	Copia o valor de $ra para $s1

Tabela 2.69: Pseudoinstrução Jalpcd

Jalpcd	**$s1,**	**100**	**Desvia para o endereço $pc deslocado da constante de 16 bits salvando origem em $s1**
Add	$t1,	$Zero, $Zero	Armazena o valor zero em $t1
Lui	$t1,	mais significativa	Carrega $t1 com a parte mais significativa da constante
Add	$t1,	$pc, $t1	Soma $pc com $t1
Jal	$t1,	menos significativa	Operação direta da microinstrução
Add	$s1,	$ra, $Zero	Copia o valor de $ra para $s1

3 METODOLOGIA DE PROJETO

3.1 A Aplicação Básica

A necessidade da criação de uma primeira aplicação que realize uma seqüência básica de operações fica clara quando se coloca o fato de que todo o projeto do SCW foi desenvolvido de forma a atender as especificações do projeto (*customized*).

O projeto do processador, bem como de toda a parte de memória e interfaces foi realizado com vistas ao sistema SCW. Também o projeto de *software* foi feito visando atender à características específicas de *hardware* do SCW. Desta maneira, enquanto não exista um meio de testes físico, devem ser efetuados testes de validação, tanto em *hardware,* quanto em *software,* de maneira sistemática para que o sistema possa ser construído de forma segura e sem falhas.

Ao elaborar-se uma primeira versão de aplicação para o microprocessador, torna-se mais plausível para programadores, que venham a desenvolver aplicações de maior complexidade, com o objetivo de utilização em escala de fabricação, o esquema de funcionamento do microprocessador e seus detalhes.

Com o propósito de testar a funcionalidade da linguagem desenvolvida para o SCW, a aplicação deveria (1) trabalhar com todas as interfaces de comunicação, (2) utilizar o maior número possível de instruções pertinentes ao *set*, sem aumentar em demasia a complexidade e o tamanho do programa, e (3) efetuar operações de processamento básicas. A aplicação desenvolvida neste trabalho deveria então controlar todo fluxo de dados entre o processador e as interfaces, realizar algum tipo de processamento sobre os dados recebidos, e trabalhar com instruções de acesso à memória.

Para atingir tais objetivos, foi realizado um estudo extensivo do projeto elaborado para o *hardware* e para o *software*, presentes nas referências [4], [6], de forma a identificar suas características principais e também estabelecer especificações básicas para a aplicação que atendessem perfeitamente aos dois projetos.

De forma a utilizar as interfaces de comunicação projetadas, foram analisadas todas as estruturas contidas no chip. Em se tratando da estrutura de *I/O* desenvolvida para o SCW, verificou-se que todas as operações de aquisição e transmissão de dados da interface para o processador eram controladas por meio do uso de interrupções. Foi necessária então a criação de uma rotina de controle deste tipo de sinalização para o processador, já que é através dela que o fluxo de informações deve ser controlado.

Assim, a aplicação foi dividida em termos de projeto em duas partes distintas no que concerne à elaboração, mas paralelas em sua operação: a etapa de tratamento de exceções e a etapa de operação dos dados, dentro da aplicação.

O paralelismo entre as duas fases se verifica quando se analisa a aquisição e a transmissão de dados para as interfaces. A rotina de tratamento de exceções deve controlar o fluxo de dados de entrada e armazená-los em memória, enquanto que a fase de processamento deve retirar informações da memória, processá-las e, em seguida, enviá-las a uma das interfaces de saída.

Tanto na criação da rotina de tratamento de exceções, quanto na rotina para processamento, em se tratando de ferramentas computacionais, foi utilizada a metodologia sistemática geral para desenvolvimento de *softwares*. Com as especificações de processamento e de funcionamento das interfaces de comunicação,

obtidas nas referências [4] e [6],foram elaboradas seqüências básicas de operação, para as duas etapas, de acordo com as atividades idealizadas para alcance dos objetivos propostos. Em seguida, foi desenvolvido, para cada uma das etapas, um algoritmo de trabalho que seguisse a série de atividades desejada. O algoritmo deveria descrever seqüencialmente os passos a serem realizados pelo programa em funcionamento, de maneira esquemática, para embasamento da aplicação em linguagem de máquina. Aspectos como acesso direto à memória, tempo de execução, possibilidade de programação modular, e principalmente inteligibilidade e facilidade de entendimento foram os parâmetros estudados para a escolha da linguagem de programação a ser utilizada: as pseudoinstruções.

Definida a linguagem de programação, foram implementados programas correspondentes à rotina de tratamento de exceções e à rotina de processamento, utilizando os recursos disponíveis com o *set* de pseudoinstruções criados para compor a linguagem de montagem.

Na etapa seguinte, já definido o bloco programacional completo (conjunto composto pela rotina de tratamento de exceções e pela rotina de processamento dos dados), passou-se à verificação da montagem em códigos de máquina no bloco de memória do processador. Este procedimento foi de fundamental importância pois, através dele, foram detectados erros de sintaxe resultantes da utilização de pseudoinstruções. Também, por meio da montagem em memória, foi possível avaliar o tamanho real do bloco computacional, permitindo a realização de estimativas de tempo de execução e a análise do desempenho do programa desenvolvido. A

Figura 3.1 mostra esquematicamente as etapas percorridas para a produção da aplicação.

Figura 3.1: Esquemático da seqüência de atividades seguida na etapa de desenvolvimento

3.2 Testes com o *Software* Montador

Todo o *set* de instruções criado para o sistema foi desenvolvido com base no conceito RISC que prevê um *set* reduzido de instruções e, a partir das mesmas, é construído um conjunto maior de instruções, que realizem tarefas específicas, emulando assim um *set* de instruções completo.

Na implementação de arquitetura RISC a complexidade das instruções é trabalhada em nível de *software*, o que criou a necessidade de um programa que convertesse a linguagem de nível mais alto (linguagem de máquina) para a linguagem de montagem verificando quaisquer erros de sintaxe, para enviar o código de máquina, devidamente compilado, para o microprocessador.

O montador [8] deve utilizar como fonte o programa em linguagem de máquina e realizar a conversão para códigos de máquina. Por realizar estes procedimentos, o montador se torna uma peça indispensável na programação de microprocessadores. Neste nível, as pseudoinstruções são transformadas em blocos de microinstruções correspondentes, que são traduzidas para código de máquina. Exemplificando, seja a instrução Add $t0,$t1,$t2. Esta instrução deverá ser convertida em código de máquina

para que o processador possa interpretá-la. Esta conversão é feita de acordo com as especificações de *software*.

Após o desenvolvimento do *software* montador foi necessária uma bateria completa de testes de verificação do seu funcionamento, de modo a permitir uma avaliação de seu desempenho. Para tal, necessitava-se verificar a resposta gerada pelo mesmo aos mais diversos tipos de emprego de instruções e sintaxe. O objetivo foi estabelecer parâmetros para sua utilização, bem como informar ao usuário quais são as possíveis causas e soluções aplicáveis dada uma determinada situação de erro ocorrida. Os resultados obtidos serão de fundamental importância no desenvolvimento da Ajuda ao *software*, onde deverão ser explicitados os erros mais freqüentes, bem como regras de uso do programa.

Buscando atender a estes objetivos, procurou-se desenvolver neste trabalho um esquema de testes que pudesse, ao mesmo tempo, indicar possíveis falhas de programação e, também, testar a implementação da tradução de linguagem de máquina para linguagem de montagem. A estratégia adotada possibilitaria, primeiramente, testar o emprego das instruções do conjunto de microinstruções que constitui a base para a construção do *set* completo. Numa segunda etapa, seriam feitos os testes com o conjunto completo de instruções em linguagem de máquina, e ainda, a verificação dos códigos de tradução gerados nestes casos.

Os testes do conjunto básico de instruções (microinstruções) permitiram verificar como o *software* montador responderia às mais variadas situações causadoras de erro a ele impostas. Nestas condições, desejava-se testar o emprego correto dos registradores, a utilização de constantes não aceitas por determinadas instruções e o uso uma instrução de determinado formato com uma sintaxe diferente da aceita. Todas estas situações poderiam conduzir o montador a erro.

Foram realizados testes exaustivos no sentido de identificar e executar todas estas situações bem como observar o comportamento do *software* nestas condições. Para os testes, foram elaborados mini-programas que carregados no montador, gerariam erros ou códigos compilados que deveriam corresponder aos códigos especificados para as situações. Executados os programas, o resultado das compilações foi comparado com o esperado. No caso de erros, foram identificadas suas causas e observadas as situações de ocorrência.

Os testes a serem realizados com as instruções do conjunto mais amplo (pseudoinstruções), deveriam permitir a observação dos códigos de máquina gerados para comparação com os resultados esperados. Para tal, foram implementados vetores de teste simples com as instruções dentro do ambiente do montador, e analisados os códigos resultantes da compilação. Os códigos obtidos quando da compilação da pseudoinstrução deveriam ser os mesmos se utilizado o bloco de microinstruções constituinte da mesma.

Os vetores de testes elaborados para as pseudoinstruções visam: (1) testar a utilização de sintaxes contendo registradores usados internamente pela instrução; (2) testar o comportamento do *software* para declaração de constantes com valores maiores que os do range aceito e (3) identificar erros de tradução. Foram criados mini-programas, de forma a atender estes objetivos, que foram executados no montador. Não foram necessários, nesta etapa, os testes extensivos de sintaxe, já que estes testes foram efetuados para as microinstruções, e os resultados esperados seriam os mesmos para os

testes para as pseudoinstruções, que é composto das microinstruções.

4 A APLICAÇÃO

4.1 Especificações

Conforme discutido anteriormente, para validar o projeto de *hardware* e *software* é necessário o desenvolvimento de uma aplicação que utilize o maior número possível de componentes de *hardware*. Para atingir este objetivo foi projetado um bloco programacional que pudesse realizar as seguintes atividades:

1. Aquisição de dados;
2. Leitura e escrita em memória;
3. Processamento destes dados;
4. Transferência dos resultados à uma interface de saída.

Para tornar a aplicação suficientemente próxima de um programa real que possa ser implementado no projeto do SCW, optou-se por utilizar a interface de saída de RF como interface padrão para envio de dados processados e, a interface ΣΔ como padrão para obtenção de dados. Assim sendo, o bloco programacional representado pela rotina de tratamento de exceções e pela aplicação, deveria retirar dados disponibilizados pela interface de entrada ΣΔ e enviá-los à memória. Em seguida, os dados deveriam ser recuperados, realizar algum tipo de processamento e, em seguida disponibilizá-los à interface de saída de RF para envio.

Durante a fase de estabelecimento das especificações para a aplicação, verificou-se a necessidade do envio de dados de controle durante a execução, de modo a permitir ao usuário o acesso às informações relativas ao andamento do programa. Para tanto, foi escolhida a interface serial. A estratégia de envio de dados de controle utilizando a interface serial possibilitaria o uso das duas interfaces de comunicação de saída de dados, permitindo o teste do módulo completo de saída do processador.

Para uma simulação completa do funcionamento do processador, os dados deveriam de alguma forma ser operados por ele. O processamento deveria ser capaz de gerar erros de endereçamento ou *overflow* aritmético, de modo a testar o funcionamento do mesmo nestas duas situações.

Nesta primeira versão da aplicação, optou-se então um processamento simples, mas que atendesse às necessidades básicas de operação e teste. Foi escolhida a realização de uma operação de multiplicação dos dados por uma constante. Este procedimento é eficiente em relação ao alcance do objetivo maior da aplicação que é o de testar a utilização das interfaces, a estrutura do processador e a linguagem de montagem desenvolvida para o SCW. Em versões posteriores de aplicações para o sistema, outros processamentos mais elaborados poderão ser efetuados, mantendo a mesma estrutura básica do programa desenvolvido para a aplicação.

A Figura 4.1 representa um esquemático da seqüência de operações a ser realizada pela aplicação.

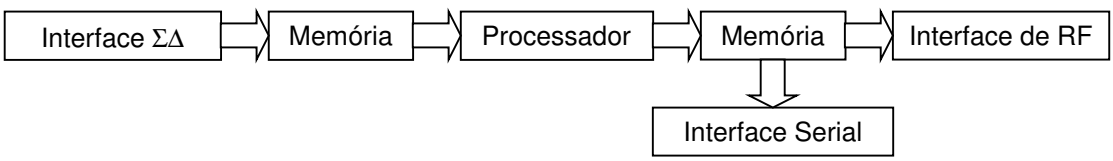


Figura 4.1: Esquemático completo da unidade de controle de sistema

4.2 Entrada de Dados

A estrutura do SCW prevê em sua utilização 3 interfaces de comunicação. As interfaces se comunicam com o processador por meio de posições de memória que são ligadas diretamente à interface e ao processador. É através destas posições de memória que a interface disponibiliza ao processador os dados obtidos, para que os mesmos possam ser armazenados em memória.

Analisando as especificações, verificou-se que o processo de aquisição de dados pelo processador deve funcionar da seguinte forma:

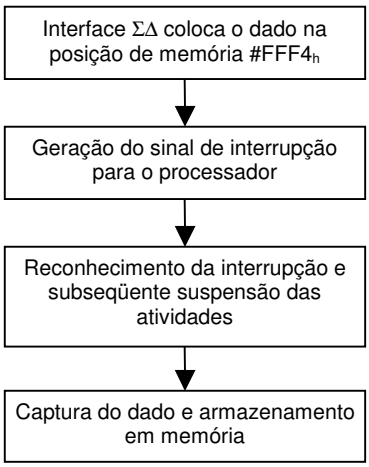
1 – A interface disponibiliza o dado a ser processado no "registrador" mapeado em memória destinado a este fim;
2 – É disparada uma interrupção ao processador, que pára suas atividades, e captura o dado;
3 – O dado é enviado à memória por meio de operações de escrita.

No caso da aplicação discutida neste trabalho, os dados são provenientes da interface ΣΔ. A interface ΣΔ, por se tratar do meio de comunicação entre o processador e um sensor de pressão ou movimento (APS), constitui uma entrada primária de informações. O sensor envia seus resultados para processamento através da interface.

Verifica-se que para a interface ΣΔ, são usadas as posições de memória dedicadas referenciadas na Tabela 2.1.

Tendo em vista que a seqüência de recebimento de informações acontece seguida de uma interrupção, o algoritmo e o bloco de programação em linguagem de máquina criado para a aquisição de dados provenientes da interface ΣΔ, será discutido em detalhes no capítulo 5, em conjunto com a rotina de tratamento de exceções. A Figura 4.2 representa esquematicamente a aquisição dos dados por parte do processador.

Figura 4.2: Aquisição de dados originados na interface ΣΔ



4.3 Processamento

A etapa de processamento dentro da aplicação desenvolvida neste trabalho é a responsável pela operação dos dados originados nas interfaces, e conseqüentemente pela geração de resultados. Nesta etapa, recuperados os dados salvos em memória, alguma ação deve ser realizada sobre os mesmos, de forma a testar a funcionalidade do projeto.

A aplicação requeria um processamento que fosse simples, dado que os objetivos primeiros são o teste e a validação, e que permitisse ao mesmo tempo avaliar as funções do *set* de instruções escolhido.

Escolheu-se a operação de multiplicação, por ser a operação aritmética dentro do *set* com maior possibilidades de geração de erros de *overflow*. Desta maneira, seria possível com a realização deste processamento básico testar a sinalização da ocorrência de operações com resultados maiores que 16 bits.

Ainda visando a simplicidade, escolheu-se uma constante como outro parâmetro para efetuar a operação de multiplicação. Se fosse escolhido um número não previamente definido para a realização da multiplicação, se tornaria difícil avaliar o comportamento das saídas subseqüentes e analisar possíveis erros.

Em se tratando da utilização de linguagem de máquina para implementação da aplicação, o armazenamento de constantes é feito de maneira bruta, apenas guardando em determinada posição de memória, não referenciada pelo programa como uma instrução, o valor a ser utilizado posteriormente. Quando do *startup* do processador, as constantes devem ser carregadas juntamente com o bloco programacional dentro da memória do SCW.

Com relação ao valor que a constante de multiplicação deveria assumir, foram analisadas algumas opções. A multiplicação pelo valor zero seria totalmente ineficiente, no sentido de que as saídas observadas no andamento do programa assumiriam sempre valor zero, impossibilitando a identificação de prováveis erros ou ainda de problemas ocorridos na execução do programa. O valor 1 também se mostrou ineficaz pois não geraria nunca o erro de *overflow* aritmético. Observou-se também que valores maiores que 2048 ($\#0000100000000000_b$ = $\#0800_h$) para a constante de multiplicação fariam com que a possibilidade de ocorrência de *overflow* aumentasse muito, descaracterizando o resultado.

O valor escolhido para a constante foi então 255 ($\#0000000011111111_b$ = $\#00FF_h$), que ocasionaria a sinalização de *overflow* para casos em que o valor dado a ser multiplicado fosse superior a $2^8 - 1$ ou 257. Esta constante deve ser carregada com o programa e seu valor, mantido.

O dado adquirido da interface ΣΔ e a constante de multiplicação, anteriormente armazenados em memória devem ser carregados em registradores, para em seguida efetuar-se a multiplicação. O processamento é realizado de acordo com o algoritmo seqüencial da Figura 4.3 .

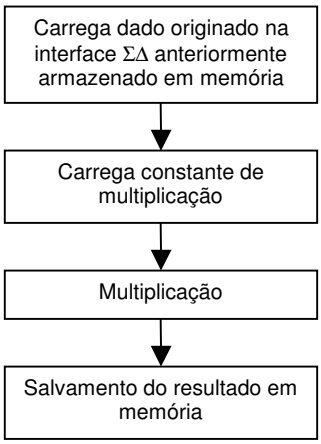


Figura 4.3: Seqüência de atividades realizadas na etapa de processamento

Dado que as posições de memória onde serão armazenados os dados disponibilizados pela interface ΣΔ serão definidas quando do tratamento da interrupção correspondente ao recebimento de dados, o algoritmo para o processamento de dados de entrada poderá ser definido com mais detalhes, quando da apresentação da rotina de tratamento de exceções, discutida no capítulo 5.

4.4 Saída de Dados

Após o processamento dos dados disponibilizados pela interface de entrada, os dados resultante devem ser encaminhados à interface de saída. A interface de saída padrão é a interface de RF, que no caso do SCW transmitirá ao meio os resultados de processamento, através do transceptor de RF.

Para tal é necessária a configuração completa da interface, utilizando parâmetros como potência de transmissão e freqüência de operação. Os parâmetros são transmitidos pelo processador à interface por meio de posições de memória dedicadas ao *setup* de transmissão.

As interfaces de saída (de RF e serial) possuem também um tipo de sinalização que indica o estado da mesma, que pode ou não estar pronta para transmitir um determinado dado disponibilizado pelo processador. Esta sinalização é feita através do bit 0 da posição de memória dedicada ao *status* da interface. As posições em memória dedicadas à interface de RF são mostradas na Tabela 2.1.

O procedimento de transmissão de dados através da interface de RF deve ser realizado continuamente pois a cada dado processado corresponde um envio para a interface. Isto sugeriu o uso de uma subrotina para disponibilizar o dado à interface, que pudesse ser chamada durante a seqüência do programa. Uma vantagem da utilização de subrotinas é a de se possibilitar a reutilização dos módulos já elaborados em outros programas, sem que se comprometa o desempenho e a flexibilidade que são compromissos nesta primeira aplicação.

As especificações para a interface de RF ainda não estão completamente definidas, desta maneira não se pôde nesta etapa trabalhar com palavras de *setup* reais, que possuam informações com relação à transmissão de dados. Trabalhou-se então com uma configuração arbitrária da interface, atribuindo às palavras de *setup* não correspondentes à uma transmissão efetiva de dados. No entanto, os valores utilizados no desenvolvimento desta aplicação podem ser modificados de forma a atender às futuras especificações, tão logo seja completado o projeto da interface de RF. Para tornar ainda mais eficiente o processo de envio que deve ser implementado através da chamada de uma subrotina dentro do programa, as palavras de *setup* deveriam tratadas como constantes carregadas quando da chamada ao procedimento.

O processo de transmissão deve então ser iniciado com a verificação do *status* da interface de saída. Caso a interface encontre-se pronta, o dado é colocado na posição de memória $\#FFFC_h$ (transmissão de dados para interface de RF) e em seguida a interface deve ser configurada. São então carregadas as palavras de *setup* para transmissão. A interface de RF utiliza duas palavras de *setup*, nas posições $\#FFFE_h$ e $\#FFFF_h$, conforme a Figura 2.3.

A verificação do estado da interface de RF deve ser realizada através do bit 0 da posição $\#FFFD_h$, que é a posição prevista para o armazenamento do *status* da unidade. Se a interface encontra-se habilitada a receber o dado, o mesmo é subseqüentemente enviado pelo procedimento anteriormente descrito. Analisando a situação em que a

interface não se encontra disponível para recebimento no momento da solicitação por parte do processador, algumas possibilidades foram estudadas. O processador poderia neste caso aguardar em *stand by* pela permissão de envio, até que a mesma acontecesse. A adoção deste procedimento seria demasiado ineficiente, pelo fato de que, com o processador aguardando a liberação da interface para envio, outras atividades deixariam de ser executadas como, por exemplo, o tratamento das solicitações de aquisição de dados feitas por outras interfaces.

Como a aplicação tem a finalidade primeira de testar as especificações propostas tanto na parte de *hardware* quanto na parte de *software*, o tratamento escolhido para lidar com esta circunstância é o de envio de uma informação de erro à uma das interfaces, como maneira de sinalizar ao ambiente externo ao processador a ocorrência desta situação. Deveria ser então elaborado um esquema de codificação, onde os dados enviados à interface pudessem, de alguma forma, determinar o erro corrente.

Observa-se que a definição dos códigos de erro que podem ser gerados como números de 16 bits, acarreta a possibilidade de tratar um destes códigos como o resultado de um processamento sendo encaminhado a interface de saída. Porém os códigos de erro são transmitidos somente às interface serial. A interface de saída para os dados processados pela aplicação é a interface de RF. Deste modo, para a aplicação discutida, onde a interface Serial foi escolhida para envio de códigos de erro, elimina-se a hipótese de os dados processados serem confundidos com códigos de erro. Esta alternativa se adequa bem aos objetivos deste trabalho, que são o teste e a validação.

Em recebendo o dado, a interface gera então uma palavra de verificação de comunicação, que é armazenada num endereço em memória e é acessível ao processador, como os dados em trânsito e a palavra de *setup*. Ao final da rotina de transmissão de dados, deve então ser realizado um teste referente ao *status* desta comunicação, de modo a detectar possíveis erros. Este teste deve ocorrer dentro da rotina de tratamento de exceções, quando da geração da interrupção proveniente da interface para transmitir ao processador o resultado do processo de envio.

A Figura 4.4 representa esquematicamente a ação de transmissão de dados de saída à interface de RF.

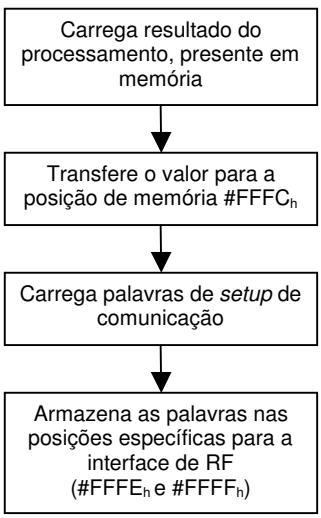


Figura 4.4: Esquema representativo da transmissão de dados pela interface de RF

4.5 Saída de Dados de Controle

Verificou-se que deveria ser feita a transmissão de dados de controle ao ambiente externo ao processador, como forma de sinalização externa de erros e de levar ao ambiente externo ao processador informações sobre o andamento da execução. A interface escolhida para envio de dados de controle foi a interface serial. Como dados de controle a serem enviados pela interface serial ao ambiente exterior ao processador foram escolhidos os conteúdos dos registradores $int e $pc.

A escolha do registrador $int foi feita com base em seu conteúdo, que guarda o endereço da última interrupção ocorrida, além das informações relativas ao código de interrupção. Já a opção pelo registrador $pc torna-se necessária pelo fato de que uma das informações importantes com relação à operação do programa, a qual se deseja ter acesso, é a posição da instrução executada. Por meio dela, pode-se avaliar situações em que ocorra um *loop* infinito dentro do programa.

Considerando que nesta primeira versão o programa teria um tamanho reduzido em relação a versões posteriores pela simplicidade do processamento realizado, optou-se pela transferência dos dados de controle duas vezes durante a execução do programa. No entanto, para aplicações posteriores de maior complexidade deve ser estudada a possibilidade de se realizar um número maior de transferências dos dados de controle dentro da seqüência programacional, de maneira a atender às necessidades de controle que são específicas, variando de programa para programa.

Seguindo a proposta de utilização de subrotinas para os procedimentos mais efetuados durante a execução, foi criada também uma subrotina para transferência de dados de controle para a interface de saída serial.

A transferência de dados realizada para interface serial é muito similar à realizada para a RF. Porém, cada uma das interfaces possui "registradores" específicos mapeados em memória para configuração da transmissão e para disponibilização dos dados a serem enviados. A Tabela 2.1 contém as posições de memória dedicadas à interface serial.

Assim como na transmissão através da interface de RF, a primeira etapa no processo de transmissão de dados de controle, seria a aquisição do dado a ser enviado à interface. Após o *setup*, o dado seria então colocado na posição específica para envio e a interface devidamente configurada, faz a leitura do dado a ela enviado.

No caso de erro decorrente da tentativa não concretizada do envio de dados à interface, seria necessária também aqui a transmissão de um código identificador que deveria ser enviado para interface serial. Novamente analisando a situação descrita na sessão 4.4, em que se solicita o envio de dados processados à interface, e a mesma não se encontra pronta. A transmissão para interface serial poderia parecer neste momento uma péssima escolha. No entanto, recorrendo às especificações da aplicação desenvolvida para o SCW observa-se que um dos seus objetivos é o teste de *hardware* quando da execução do programa, possibilitando o diagnóstico de partes defeituosas. A verificação de lacunas de dados na saída da interface serial poderia ser percebida por meio de uma monitoração simples, e indicaria algum problema em seu funcionamento, dado que a aplicação a utiliza para envio de dados de controle ao ambiente exterior. A ausência destes dados implicaria em defeito nesta interface.

Após a aquisição do dado, as interfaces de saída devem gerar uma palavra de *status* de comunicação que deve ser armazenada em memória para reconhecimento do

processador. Também este procedimento é executado pela interface serial. Após a geração deste *status*, o processador é avisado por meio de uma interrupção da presença do resultado da última tentativa de envio. A execução é então suspendida e o processador passa ao tratamento da situação.

A Figura 4.5 representa a seqüência de atividades para o envio de dados à interface serial, dentro da rotina de envio de dados de controle.

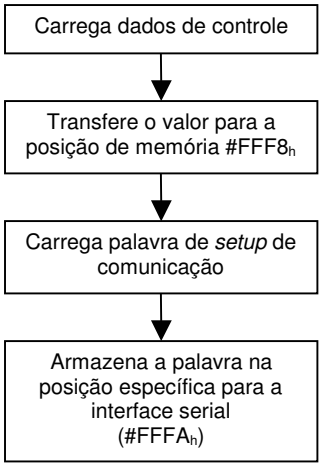


Figura 4.5: Esquemático da seqüência de envio de dados de controle

Na Figura 4.6 verifica-se a seqüência detalhada a ser seguida pelas rotinas de transmissão de dados, seja o destino a interface de RF ou a interface serial. Nos dois casos deve ser realizado o teste de status antes da tentativa de envio. Em caso positivo, o dado é colocado na posição de memória dedicada ao envio. Em caso negativo, deve ser transmitido pela interface serial o código descritivo do erro ocorrido.

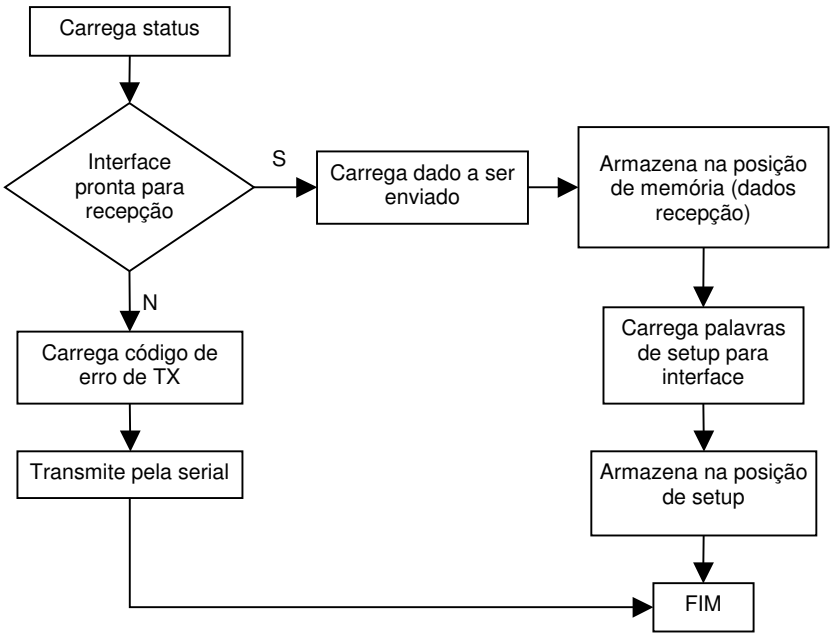


Figura 4.6: Algoritmo para envio de dados às interfaces de saída

Para a rotina de envio de dados de controle, seguiu-se o mesmo princípio. Foi utilizada uma subrotina de transferência de dados de controle, que seria chamada pelo programa principal duas vezes ao longo de sua execução. Tendo ainda em vista que a subrotina de envio de dados de controle envolveria duas transmissões de dados para a

interface serial, sendo eles os registradores $int e $pc, deveriam ser realizadas dentro da subrotina de controle, duas chamadas à subrotina de transferência de dados para a interface serial.A estratégia adotada neste trabalho para o controle eficiente do fluxo de dados entre o processador e as interfaces, foi o armazenamento e leitura dos dados em memória usando-se dois ponteiros independentes. O procedimento elaborado para controle dos ponteiros é detalhado na sessão 5.4.1 deste trabalho.

Todo o controle dos ponteiros, bem como testes para verificação da validade das posições a serem usadas para leitura e escrita é realizado pela rotina de tratamento de interrupções, é descrito no capítulo 5. Uma vez que o controle é realizado quando da entrada de dados, resta apenas fazer as modificações dos ponteiros de leitura quando da retirada do dado para processamento, processo realizado durante a aplicação. Assim, quando da leitura de um dado em memória, à aplicação cabe apenas a tarefa de ajustar o ponteiro, passando à posição posterior em pilha, e controlar o tamanho, fazendo com que o ponteiro não ultrapasse os limites da pilha específica para cada interface. Estes procedimentos devem ser realizados na primeira etapa da aplicação e são parte integrante do procedimento de retirada dos dados da pilha para posterior processamento.

4.6 Algoritmos Elaborados para a Aplicação

A Figura 4.7 representa o algoritmo básico para a aplicação desenvolvida.

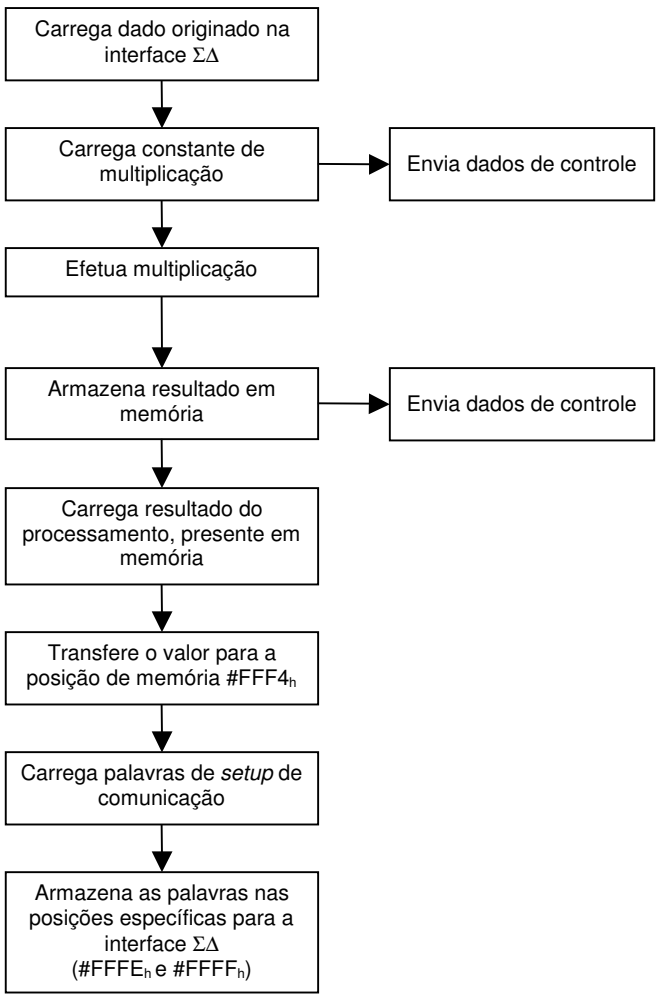


Figura 4.7: Esquema seqüencial simplificado par a aplicação

Tendo sido retirado o dado de entrada da memória, passa-se então ao processamento, que é realizado por meio de uma operação de multiplicação por uma constante. Em seguida, armazena-se o resultado em memória e o controle de execução pode ser transferido para a rotina de envio de dados para a interface RF, que deve transmitir o resultado da multiplicação.

Os dados de controle são enviados duas vezes durante a execução da aplicação: uma vez antes do processamento, ou seja, após a recuperação do dado da memória e depois do processamento, onde pode ser gerada uma interrupção por *overflow* aritmético.

De forma a aprofundar ainda mais o algoritmo elaborado presente na Figura 4.7, nas Figura 4.8 e Figura 4.9 encontram-se os algoritmos detalhados para o envio de dados de controle e o algoritmo detalhado de transmissão de dados pela interface de RF.

Figura 4.8: Algoritmo para envio de dados à interface de RF

Nas Figura 4.7, Figura 4.8 e Figura 4.9 é possível observar a dinâmica envolvida no processo realizado pela aplicação. Primeiramente carrega-se o dado proveniente da interface ΣΔ. Em seguida, carrega-se a constante de multiplicação. Neste momento deve ser feita a transferência de dados de controle para a interface serial. Passa-se então à subrotina de dados de controle, descrita pelo hachurado ⊠ na Figura 4.9. É carregado o primeiro dado a ser transmitido que é o conteúdo do registrador $pc, e em seguida é chamada a subrotina de envio de dados para a interface serial, indicada pelo hachurado ⊠ . Neste momento a interface serial realiza, se possível, a operação de transmissão e transfere o controle novamente à rotina de envio de dados de controle, que então carrega o registrador $int e faz nova chamada à rotina de transmissão de dados para a interface serial. Após a transmissão do registrador $int, o controle é

retomado pela rotina de dados de controle, que deve fazer o retorno à execução normal do programa.

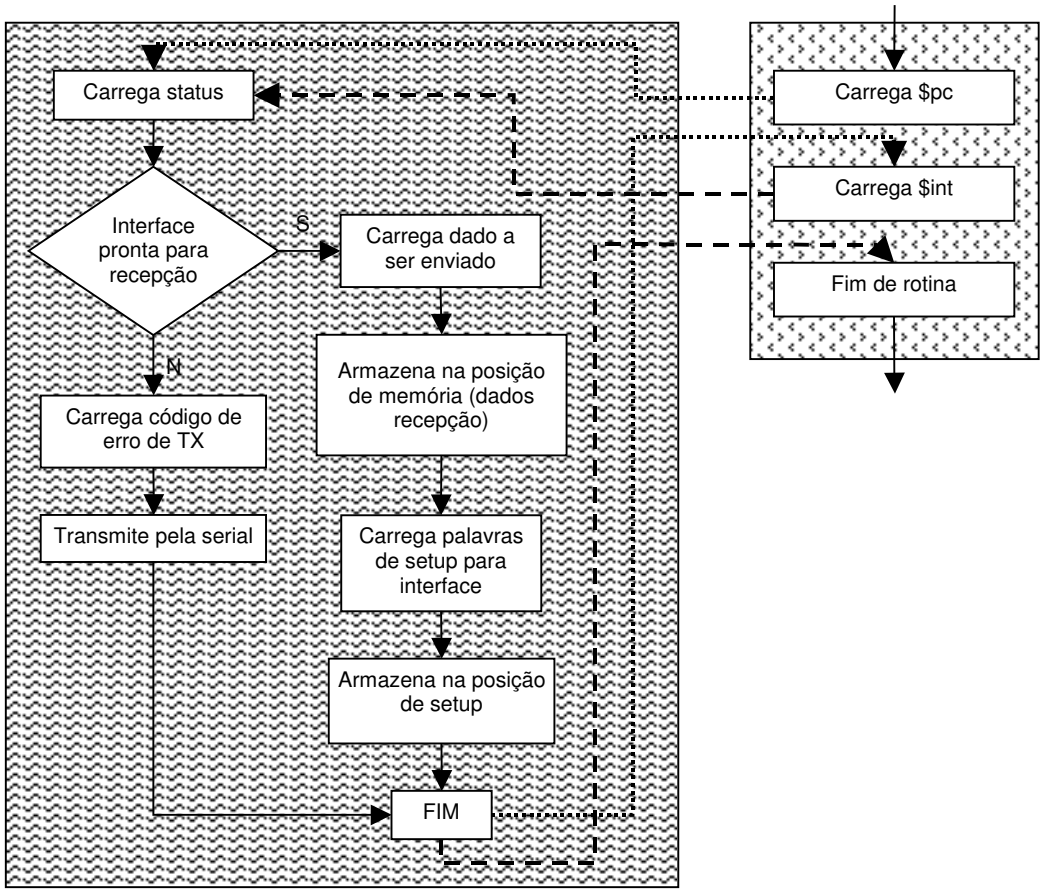


Figura 4.9: Algoritmo para a saída de dados de controle

Em seguida é realizada a etapa de processamento dentro da aplicação, que é representada pela operação de multiplicação. O resultado gerado é então armazenado em memória. É feita nova chamada à rotina de envio de dados de controle, que transmitirá novamente por meio da interface serial os conteúdos dos registradores $pc e $int.

Após o retorno da segunda chamada à rotina de envio de dados de controle, o programa prepara-se para transmitir o resultado obtido do processamento para a interface de RF. A seqüência que deve ser executada para transmissão do dado é descrita na Figura 4.8. Depois de executar os procedimentos relativos à transmissão via RF, o programa deve retornar à primeira instrução do programa, e conseqüentemente ao seu início para novo processamento.

4.7 A Aplicação em Linguagem de Máquina

Com os algoritmos definidos, passou-se a programação em linguagem de máquina da aplicação. Novamente, a utilização da pseudolinguagem por sua maior simplicidade de inteligibilidade levou à impossibilidade da definição de endereços em memória, tanto para as constantes quanto para os desvios incondicionais. Foi desenvolvido o bloco de programa descrito pela Tabela 4.1

Tabela 4.1: Bloco programacional para a aplicação

Label	Instrução	Comentários
	J Interrupção	Realiza salto incondicional para posição em que se encontra a rotina de tratamento de interrupções.
Posição início aplicação	Lwi $t0, pl_sd	Carrega dado proveniente da pilha ΣΔ
	Lw $t1, $t0, $zero	
	Lw $t2, $t1, $zero	
	Lwi $s0, proc	Armazena o dado na posição de processamento
	Sw $t2, $s0, $zero	
	Lwi $t2, fim_de_pilha_SD	Testa se a posição do ponteiro é válida. Se chegou à última posição, recoloca o ponteiro no topo da pilha. Se não, ajusta o ponteiro, incrementando-o e armazenando-o novamente
	Beq $t1,$t2, 5	
	Addi $t1,0001h	
	Sw $t1,$t0,$zero	
	Lwi $a1, Dados_Controle	
	J $a1, 0	
	Lwi $t1, $início_de_pilha	
	Sw $t1,$t0,$zero	
Dados_Controle	Lwi $s1 $TX_controle	Vai para rotina de envio de dados de controle
	Jal $s1,0	
	Lwi $t0, const_mul	Carrega o dado e realiza a multiplicação por uma constante
	Lw $a0, $t0, $zero	
	Lwi $t0, proc	
	Lw $a1, $t0, $zero	
	Mul $s1, $a0, $a1	
	Lwi $s2, result	Armazena o resultado
	Sw $s1, $s2, $zero	
	Lwi $s1 $TX_controle	Vai para rotina de envio de dados de controle
	Jal $s1,0	
	Lwi $s0, result	Coloca o resultado na posição que vai ser utilizada pela rotina de envio para RF que fica armazenada em memória como constante dado_RF
	Lw $s1, $s0, $zero	
	Lwi $s0, dado_RF	
	Sw $s1, $s0, $zero	
	Lwi $s1, $TX_RF	Chama rotina de envio de dados para RF
	Jal $s1, 0	
	Lwi $s0, Posição_início_aplicação	Retorna ao início para novo processamento
	Jal $s0, 0	
TX_controle	Lwi $a0, Guarda_$ra	Armazena $ra para retorno da subrotina
	Sw $ra, $a0, $zero	
	Lwi $t0, dado_Serial	Armazena $pc no endereço para envio de dados para Serial
	Sw $pc,$t0, $zero	
	Lwi $t0, $TX_Serial	Chama subrotina de envio de dados para Serial
	Jal $t0, 0	
	Lwi $t0, dado_serial	Armazena $int no endereço para envio de dados para Serial
	Sw $int,$t0, $zero	
	Lwi $t0, $TX_Serial	Chama subrotina de envio de dados para Serial
	Jal $t0, 0	
	Lwi $t0, Guarda_$ra	Reestabelece o valor de $ra antes da chamada ao procedimento de controle
	Lw $ra, $t0, $zero	
	J $ra,0	Retorna à execução normal do programa
TX_Serial	Lwi $t1, FFF9h	Carrega *Status* da interface Serial
	Lw $t0,$t1, $zero	
	Andi $t0,0000 0000 0000 00001	Isola o bit menos significativo (*ready*), para testar o estado da interface.
	Beq $t1,$zero, Ready_Serial	Se a interface está pronta, vai para Ready_Serial
	Lwi $t0, Cód_Erro_TX_Serial	Se não está pronta, envia código de erro para a interface serial
	Lw $t0, $t0, $zero	
	Lwi $t1, Dado_Serial	
	Sw $t0,$t1,$zero	
	J TX_Serial	
Ready_Serial	Lwi $s0,dado_serial	Carrega dado a ser enviado para a interface Serial
	Lw $s1,$s0,$zero	
	Lwi $s2, FFF8h	Armazena o dado no registrador apropriado mapeado em memória
	Sw $s1, $s2, $zero	
	Lwi $t0, setup_TX_serial	Carrega palavra de *setup* de transmissão Serial
	Lw $t1, $t0, $zero	
	Lwi $t2, FFFAh	Armazena setup no registrador apropriado mapeado em memória
	Sw $t1, $t2, $zero	
	J $ra,0	Retorna à execução normal do programa

Label	Instrução	Comentários
TX_RF	Lwi $t1, FFFD	Carrega palavra de *status* de RF
	Lw $t0,$t1, $zero	
	Andi $t0,0000 0000 0000 00001	Isola o bit menos significativo (ready), para testar o estado da interface.
	Beq $t0,$zero, Ready_RF	Se a interface está pronta, vai para Ready_RF
	Lwi $t0, Código_de_Erro_RF	Se não está pronta, envia código de erro para a interface serial
	Lw $t0, $t0, $zero	
	Lwi $t1,Dado_Serial	
	Sw $t0,$t1,$zero	
	J TX_Serial	
Ready_RF	Lwi $s0, dado_RF	Carrega o dado a ser transmitido pela interface de RF
	Lw $s1, $s0, $zero	
	Lwi $s2, FFFC	Armazena o dado no registrador apropriado mapeado em memória
	Sw $s1, $s2, $zero	
	Lwi $t0, setup_TX_RF1	Carrega palavra de setup de transmissão de RF
	Lw $t1, $t0, $zero	
	Lwi $t2, FFFE	Armazena setup no registrador apropriado mapeado em memória
	Sw $t1, $t2, $zero	
	Lwi $t0, setup_TX_RF2	Carrega palavra de setup de transmissão de RF
	Lw $t1, $t0, $zero	
	Lwi $t2, FFFF	Armazena setup no registrador apropriado mapeado em memória
	Sw $t1, $t2, $zero	
	J $ra	Retorna a execução do programa

Uma dificuldade inicialmente observada foi a especificação do endereço $\#0000_h$ para conter o desvio incondicional para a rotina de tratamento de exceções e o endereço $\#0001_h$ para a primeira instrução a ser executada pelo programa. A destinação de apenas uma posição de memória para a colocação da instrução de desvio incondicional tornaria impossível o arranjo descrito pelas especificações. Isto por que, as instruções de desvio incondicional contidas no *set* só podem ser realizadas à constantes de 8 bits ou ainda à registradores. Como a constante de 8 bits pode não comportar o tamanho do desvio que se almeja realizar. Em versões posteriores, a aplicação discutida neste trabalho pode aumentar em muito seu tamanho, e assim a rotina de tratamento de interrupções começaria após a posição $\#00FF_h$, que é o máximo desvio que se pode realizar com a instrução J. A opção mais natural seria a de se usar um registrador com o endereço da rotina nele contido e efetuar o desvio tendo o mesmo como base. No entanto, no momento da ocorrência de uma interrupção, o programa pode estar executando qualquer uma das instruções nele contidas, e assim estar utilizando os registradores, com valores diferentes do endereço da rotina de tratamento de interrupções.

A solução encontrada para o problema foi a utilização da posição destinada ao desvio para realização dentro da memória a um novo desvio. Já que o conteúdo do mesmo não poderia ser modificado durante o programa, pela restrição de sua utilização imposta pela rotina de tratamento de controle, esta seria a opção mais eficiente para contornar a dificuldade encontrada.

Considerou-se também a hipótese de se utilizar as pseudoinstruções do *set* relativas ao desvio incondicional, que possibilitariam o uso de constantes de tamanho superior a 8 bits. Porém, avaliando as pseudoinstruções verificou-se que as microinstruções que as compunham eram às vezes numerosas, e seu emprego nesta parte do processo acarretaria em mudanças drásticas nas especificações de *hardware* que deve trabalhar com o primeiro endereço de execução do programa como sendo $\#0001_h$.

Pode ser observado que a transferência dos dados de controle é realizada depois da leitura do dado e após o armazenamento do resultado do processamento. No intuito de realizar uma passagem segura de parâmetros para as subrotinas de transmissão de dados, algumas posições em memória foram reservadas de modo a conter sempre o

dado a ser enviado dentro da subrotina. Para isto, observa-se que o dado a ser transmitido é sempre armazenado na posição específica em memória antes de se transferir o controle ao subprocedimento de transmissão. Após o processamento, o resultado é transmitido à interface de RF utilizando a subrotina específica, e em seguida o programa é reiniciado, através do desvio incondicional para a primeira posição do programa. Após o desenvolvimento da aplicação, passou-se à criação das subrotinas de transmissão de dados de controle.

A subrotina de transmissão de dados de controle desenvolvida neste trabalho, realiza então duas chamadas à subrotina de envio de dados à interface serial. Uma primeira dificuldade encontrada foi o retorno correto do subprocedimento chamado. Uma vez que era chamada a subrotina de controle, o registrador $ra assumia então o valor de retorno da subrotina de controle ao programa principal. Desejando-se realizar outra chamada a procedimento dentro desta subrotina, o registrador de retorno $ra seria ocasionalmente perdido, pois quando da chamada ao procedimento de envio à serial, o registrador $ra assumiria o valor de retorno dentro da subrotina de controle. De forma a contornar esse problema e realizar eficientemente a chamada ao procedimento desejada dentro da subrotina de controle, o salvamento do registrador $ra foi necessário. Em seguida, após o envio dos dados para a interface serial, o registrador $ra é então recuperado da memória, e a instrução de desvio incondicional referente ao seu conteúdo pode ser utilizada para retorno seguro e correto ao programa principal.

Seguindo o mesmo princípio descrito antes, o dado a ser enviado para a interface deve ser armazenado em posição específica da memória para em seguida ser chamado o subprocedimento. Esta estratégia foi usada tanto na chamada ao envio de dados à serial quanto no envio de dados à RF. Assim, no caso da rotina de envio de dados de controle, dentro da subrotina, antes da chamada ao envio, os registradores cujo conteúdo deve ser transmitido, são levados à posição de memória específica, neste caso dado_serial, antes de se passar à rotina de transmissão.

Em seguida, passou-se a elaboração das rotinas de transmissão de dados. O primeiro procedimento a ser efetuado deve ser o teste para verificação do *status* da interface. O teste deve ser feito com o último bit, que indica a situação da interface: se está ou não pronta para receber as informações. Estando pronta, passa-se ao processo de transmissão propriamente dito. É realizado o carregamento do dado que se deseja enviar, anteriormente armazenado em memória sob forma de constante. A informação deve ser carregada da posição específica em memória, e em seguida enviada à posição pré-estabelecida para comunicação com a interface, de acordo com a Tabela 2.1. Em seguida deve ser obtida a palavra de *setup* de comunicação habilitando a transmissão, e o armazenamento desta palavra na posição de memória respectiva, de acordo com a interface para a qual se deseja transmitir. Após a inserção da palavra de *setup*, a interface a reconhece e captura o dado presente na posição de memória e gera após finalizar o processo de recepção uma interrupção ao processador, indicando se a transmissão foi feita com sucesso ou se houve erro durante a mesma. A aplicação em criada em pseudoinstruções é mostrada na Tabela 4.1.

5 A ROTINA DE TRATAMENTO DE INTERRUPÇÕES

Um dos tipos de controle que o microprocessador deve efetuar é o reconhecimento e tratamento das exceções e das interrupções, eventos que, a exemplo dos desvios, mudam o fluxo normal de execução das instruções dentro de um determinado programa. Uma exceção é um evento extra-seqüencial, resultante de algum processamento, ou numa última análise, da execução de alguma instrução pelo processador.

Uma interrupção, diferentemente das exceções, é um evento gerado externamente, não originado pelo processador e sim pelas interfaces de comunicação. É a maneira pela qual os dispositivos de entrada e saída (I/O) se comunicam com o processador, indicando a ocorrência de algum evento. Muitas arquiteturas e muitos autores não distinguem interrupções e exceções, muitas vezes usando o termo interrupção para referir-se a ambos os tipos de eventos.

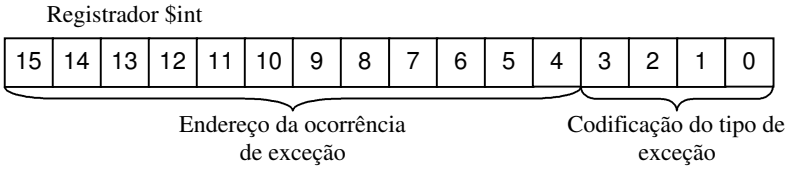
A convenção adotada neste trabalho será a mesma utilizada pela arquitetura MIPS, onde a palavra exceção denota qualquer mudança no fluxo de execução, sendo ela gerada pelo processador ou para ele. O termo interrupção designará apenas eventos gerados externamente, ou seja, pelas interfaces de comunicação.

Conforme anteriormente considerado, o registrador $int possui em sua composição campos para indicação do endereço gerador da exceção bem como bits de codificação para identificação do evento gerador da mesma. O objetivo do armazenamento do endereço é fazer com que o processador possa, ao final da rotina de tratamento retomar suas atividades do ponto exato onde aconteceu a interrupção.

5.1 Especificações

São previstos pelo processador do SCW cinco tipos de exceção possíveis [4], sendo 3 tipos de interrupção para interfaces de comunicação, além de duas sinalizações de erro. Toda a comunicação realizada entre a interface e o processador é realizada por meio das interrupções. As interrupções geradas pelas interfaces são as de recepção de dados provenientes das interfaces ΣΔ, de RF ou serial, e as geradas pelas interfaces de saída, de RF e Serial, para indicação ao processador de que o dado por ele transmitido foi recebido. As interrupções solicitadas pelas interfaces são originadas externamente ao processador. As exceções geradas pelo próprio processador são a de erro de *overflow* e a de erro de endereçamento, ambas utilizadas para sinalização da ocorrência destes erros.

De forma a construir o código identificador do tipo de exceção ocorrida usando os quatro bits menos significativos do registrador $int, optou-se então por sinalizar diretamente a ocorrência de *overflow* aritmético e erro de endereçamento, usando dois bits dedicados do registrador $int, e para as três interfaces, propôs-se a codificação, usando os outros dois bits restantes[6]. A codificação proposta é ilustrada com a Figura 5.1



Bit 3	Bit 2	Bit 1	Bit 0	Código referente
1	0	0	0	Erro de endereçamento
0	1	0	0	*Overflow* aritimético
0	0	0	0	Não utilizado
0	0	0	1	RF
0	0	1	0	Serial
0	0	1	1	ΣΔ

Figura 5.1: Codificação do registrador $int

Foi necessário se considerar um endereço de memória para o qual a execução do programa deve ser transferida, no acontecimento de uma exceção, de forma a iniciar o tratamento da mesma. O endereço definido foi o $\#0000_h$. Assim, quando da ocorrência de uma exceção, o contador de programa é desviado para a posição de memória $\#0000_h$ onde deve ser incluída uma instrução de desvio incondicional ao endereço inicial da rotina referente ao tratamento de exceções. Essa rotina determinará então o tipo de interrupção e procederá ao tratamento específico, retomando em seguida, a seqüência normal do programa. Uma das vantagens deste arranjo é que a rotina de tratamento de exceções pode ter tamanho e localização variados dentro da memória, permitindo uma maior flexibilidade ao programador.

Do ponto de vista da unidade de controle, o tratamento das exceções se dará por meio de duas operações: (1) verificação da ocorrência de interrupção ao final de cada ciclo de instrução e, em caso positivo, é feita a construção da instrução $int com o endereço da instrução causadora e o tipo de interrupção gerada, e (2) a transferência da execução do programa para o endereço designado à rotina de tratamento. Para efetuar essas tarefas, foi incluído um módulo de apoio definido como controlador de interrupções [4], que concentra as atividades de comunicação e preparação dos dados para armazenamento. O módulo de apoio é responsável pelo *check* contínuo da ocorrência de interrupção e pela execução das tarefas posteriores, como montagem do conteúdo do registrador $int com o endereço em memória da unidade causadora e código de exceção.

5.2 Etapa inicial do tratamento de exceções

Foi especificado que, quando da ocorrência de uma interrupção, a unidade de controle do microprocessador deve bloquear a recepção de pedidos. Isto é feito por meio do bit 0 do endereço de memória $\#FFF3_h$, que quando habilitado, impede a solicitação de pedidos, assegurando que o processador só interprete uma interrupção após o tratamento da última ocorrida.

Quando do acontecimento de uma exceção, seja ela externa ou interna, existem alguns procedimentos a serem executados inicialmente, independente do tipo de exceção gerado. O primeiro deles deve ser habilitar o bit 0 do endereço $\#FFF3_h$ de modo a não permitir que o processador receba outros pedidos de interrupção por parte das

interfaces ou o desvio para tratamento de erros de processamento (*overflow* aritmético e erro de endereçamento) enquanto não tiver terminado de tratar a exceção acontecida.

A intenção é que o programa retome seu funcionamento normal depois do tratamento das exceções. Para tal, é necessário assegurar que o retorno seja feito para o endereço correto e garantir que todos os registradores tenham o seu conteúdo mantido após o tratamento das exceções. Este procedimento garante a execução normal do programa, onde quer que tenha havido a interrupção, sem erros após o tratamento de uma exceção. Então deveria ser elaborado um mecanismo destinado a salvar os registradores em memória e recuperá-los após o tratamento.

Quando da ocorrência de exceções, é importante que se possa, de alguma forma, ter acesso ao tipo de exceção e ao endereço em que a mesma aconteceu do lado externo ao processador. No caso de futuramente ser projetado um sistema operacional elaborado para o SCW, a sinalização das exceções será de fundamental importância para seu funcionamento correto, e esta sinalização precisa ser feita para o ambiente externo ao processador. Uma maneira de se informar o tipo de interrupção ocorrida é fazer com que seja disponibilizado o conteúdo do registrador $int por meio de uma das interfaces, que são o canal de comunicação do processador com o que quer que possa ser conectado a ele. A interface de saída para o envio das informações referentes à interrupção corrente ao ambiente externo ao processador deve ser a interface serial. Este passo deve ser executado para todos os cinco tipos de exceção.

De posse do conteúdo do registrador $int, é possível a determinação do tipo interrupção, permitindo que sejam implementadas rotinas específicas para o tratamento dos dados nas cinco situações possíveis. Assim, de acordo com o teste a seqüência da rotina é desviada para o tratamento de cada caso.

Na Figura 5.2 um esquemático seqüencial das atividades a serem realizadas no início da rotina de tratamento para os cinco tipos de interrupção.

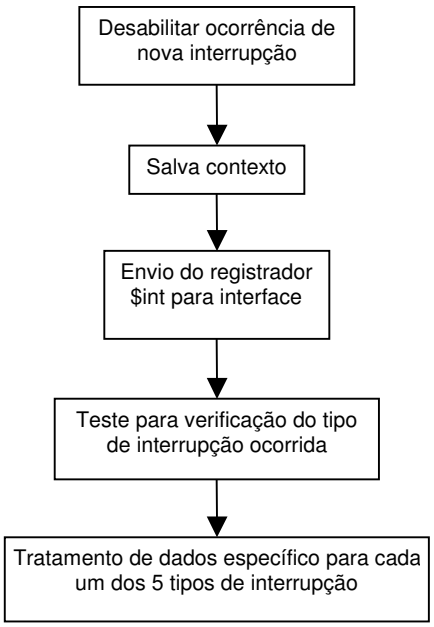


Figura 5.2: Etapa inicial do tratamento de exceções

5.3 Testes de identificação da ocorrência de exceções provenientes das interfaces de comunicação

O tratamento de cada uma das exceções inclui tarefas específicas. No caso da rotina de tratamento para interrupções provenientes das interfaces de comunicação, primeiramente devem ser realizados testes para identificar a ocorrência de uma interrupção proveniente da solicitação de envio de um dado ao processador, ou ainda, se a interrupção foi originada pela situação de sinalização de fim de recebimento de dados enviados pelo processador. As duas situações são distintas e para cada uma deve ser implementada uma seqüência específica de atividades. No caso da interrupção gerada para envio de dados da interface para o processador, o dado deve ser capturado e enviado para a memória. Isto feito, deve ser armazenada no endereço específico para tal, a palavra de *setup* de I/O correspondente à habilitação de nova transmissão pela interface.

No caso da ocorrência de interrupção gerada pelas interfaces de saída, para sinalização de recebimento correto ou não do dado anteriormente enviado a elas, deve ser verificado primeiramente se a interrupção indica erro de transmissão ou recebimento correto do dado por parte da interface. Para determinar qual das situações é a corrente, devem ser feitos testes com o registrador de *status* da respectiva interface para que a seqüência de ações seja a correta para cada um dos casos.

5.4 Tratamento individual das exceções

A etapa subseqüente aos testes de identificação é o tratamento individual dos casos de exceção. Considerando as possibilidades de se receber o sinal de interrupção proveniente das interfaces tanto para indicação do recebimento de um novo dado quanto para indicar o final de uma transmissão, foi necessária a análise das situações de recepção e envio em todos os casos pertinentes.

O processo de aquisição de dados se inicia com a ocorrência da interrupção gerada pela interface, que posiciona o dado a ser lido pelo processador no endereço específico em memória para tal. Caso o dado não seja subseqüentemente tratado, a interface pode tentar disponibilizar um novo dado, posicionando outro dado no endereço e sobrescrevendo o anterior. Fazendo referência às especificações propostas de *hardware* presentes na referência [4], verifica-se que não existe qualquer tipo de *buffer* para o recebimento de dados, e que os mesmos ficam disponíveis até que outros sejam enviados, sobrescrevendo os anteriores. De modo a evitar a perda de informações neste processo, foi então estudada uma estratégia de armazenamento temporário dos dados em memória via *software*.

Em linguagem de alto nível, se trabalha comumente com as estruturas denominadas pilhas, que nada mais são do que *arrays* de armazenamento de dados em memória. Estas estruturas são adequadas para a implementação da bufferização via *software*, pois tratam as posições de memória como espaços dedicados e indexados para guardar dados.

Numa primeira versão da rotina de tratamento de interrupções e da aplicação, em conjunto com as constantes utilizadas pelos programas, foi verificado que o espaço em memória utilizado efetivamente giraria em torno de 1KB. A memória do primeiro

protótipo SCW possuirá 4K posições de armazenamento, sendo expansível com a colocação de memórias externas até 64K [6]. Desta forma, haveria 3,5K posições de memória livres no primeiro protótipo. Esse espaço pode então ser usado para armazenamento temporário dos dados recebidos pelas interfaces, de forma segura, eficiente e sem comprometer o funcionamento da aplicação.

Nesta primeira versão da aplicação, é previsto o recebimento de dados apenas através da interface ΣΔ. Assim sendo, todo o espaço livre não utilizado pelo bloco programacional, pode ser preenchido com os dados recebidos da interface ΣΔ, que posteriormente serão processados. No entanto, em versões posteriores em que se deseje trabalhar com as interfaces Serial e de RF como entradas de dados, considerando seu caráter bidirecional, será necessária a divisão do espaço livre entre as três unidades de comunicação. A divisão em três pilhas de armazenamento temporário garantirá a eficiência no controle do fluxo de dados para a situação em que as três interfaces funcionem como entradas de dados e tornará o processamento independente das características de temporização das três interfaces.

Caso fosse considerada a situação em que as três interfaces enviassem dados ao processador, deveria ser realizado um estudo no sentido de determinar que interface demandaria maior espaço em memória. Isto poderia ser feito analisando as taxas de transmissão das interfaces e o tempo de processamento requerido para efetuar todas as operações correspondentes à aplicação implementada. A interface serial encontra-se em fase de projeto, mas tem como base o protocolo RS-232 [4]. As taxas de transmissão para este padrão são determinadas pelo uso da interface. Já as interfaces de RF e ΣΔ encontram-se ainda em desenvolvimento ([1] e [3]). Tratam-se de interfaces customizadas e desenvolvidas especificamente para o projeto, sem referências anteriores na literatura que permitam uma estimativa aproximada do tempo médio de envio de informações. Porém, após a especificação completa destas unidades em termos de projeto deve ser possível caracterizar as interfaces em termos de velocidade e taxa de transmissão, permitindo assim uma boa estimativa a respeito do tamanho das pilhas de armazenamento necessárias a cada uma das interfaces de comunicação.

Como a aplicação é bastante simples na sua concepção, é possível que futuramente o conjunto de programação (aplicação + rotina de tratamento de exceções) a ser inserido em memória ocupe mais posições e conseqüentemente, tenha seu tamanho aumentado. Esta hipótese não é descartada e foi considerada no desenvolvimento da estratégia de armazenamento temporário em pilhas de posições em memória, que deve poder ser empregada para o caso em que as três interfaces transmitam informações.

Um dos argumentos usados para a escolha da utilização de pilhas é a flexibilidade possibilitada aos futuros programadores pela não limitação física do espaço a ser utilizado para armazenamento. Dependendo do tamanho do conjunto de programação e da temporização entre processamento e recebimento de dados pelas interfaces, os espaços utilizados para cada interface podem ser modificados sem maiores complicações. A especificação nesta etapa de projeto de uma pilha de 3,5K para armazenamento de dados não implica, de maneira alguma, na fixação destes tamanhos para outros programas que venham a ser desenvolvidos oportunamente. A flexibilidade é permitida a partir do momento em que se trabalha com a linguagem de baixo nível, que possibilita acesso direto ao código, e mudanças conforme seja necessário.

Um esquemático do conjunto de memória, com a pilha de armazenamento ΣΔ delimitada é apresentado na Figura 5.3.

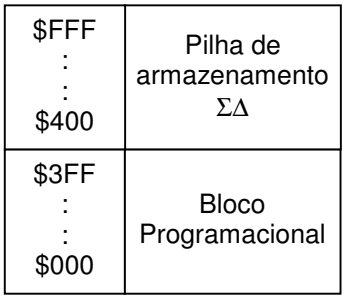


Figura 5.3: Descrição esquemática do uso da memória

5.4.1 Pilhas de Armazenamento Temporário [10]

As pilhas podem ser do tipo LIFO (*Last In*, *First Out*) ou FIFO (*First In*, *First Out*). As estruturas LIFO são aplicáveis quando a ênfase é dada à ação de leitura dos dados da pilha, em detrimento da escrita. Já as estruturas FIFO priorizam a escrita em pilha, e conseqüentemente em memória. O conceito de LIFO não é adequado para implementar a estrutura desejada de armazenamento temporário dos dados, pois considerando a situação em que o processamento demanda mais tempo do que o envio de dados, este tipo de estrutura ocasionaria perda de informações.

Para controle das pilhas foram implementados dois ponteiros atuando concomitantemente para escrita e leitura da pilha. O conceito de ponteiros é amplamente utilizado em linguagens de alto nível. Porém seu emprego em linguagem de baixo nível implica no detalhamento do processo pois é necessário o controle manual e explícito da operação da pilha. Nas linguagens de programação de alto nível este controle é feito de maneira automática, permitindo que o usuário não necessite operar a estrutura diretamente.

De forma a projetar um controle eficiente dos ponteiros de escrita e leitura, foram analisadas todas as situações possíveis no que concerne ao fluxo de dados entre o processador e as interfaces. Uma das características do sistema de processamento e interfaces é o paralelismo observado entre a recepção e transmissão de dados, que não ocorrem de forma seqüencial, e sim aleatória e independentemente. O sistema de controle de ponteiros deveria então ser capaz de atuar conforme a demanda, seja ela por leitura dos dados da pilha para posterior processamento, seja por escrita das interfaces.

Foi observado que poderiam ocorrer situações em que o tratamento dos dados pelo processador pudesse ser mais lento do que o recebimento de dados enviados pelas interfaces, gerando assim a inserção do dado numa posição de memória na pilha já ocupada, e conseqüentemente a perda do dado anterior.

A Figura 5.4 apresenta um diagrama explicativo do funcionamento dos dois ponteiros implementados no projeto, aqui chamados ponteiro de escrita em memória (Pe) e ponteiro de leitura de memória (Pl).

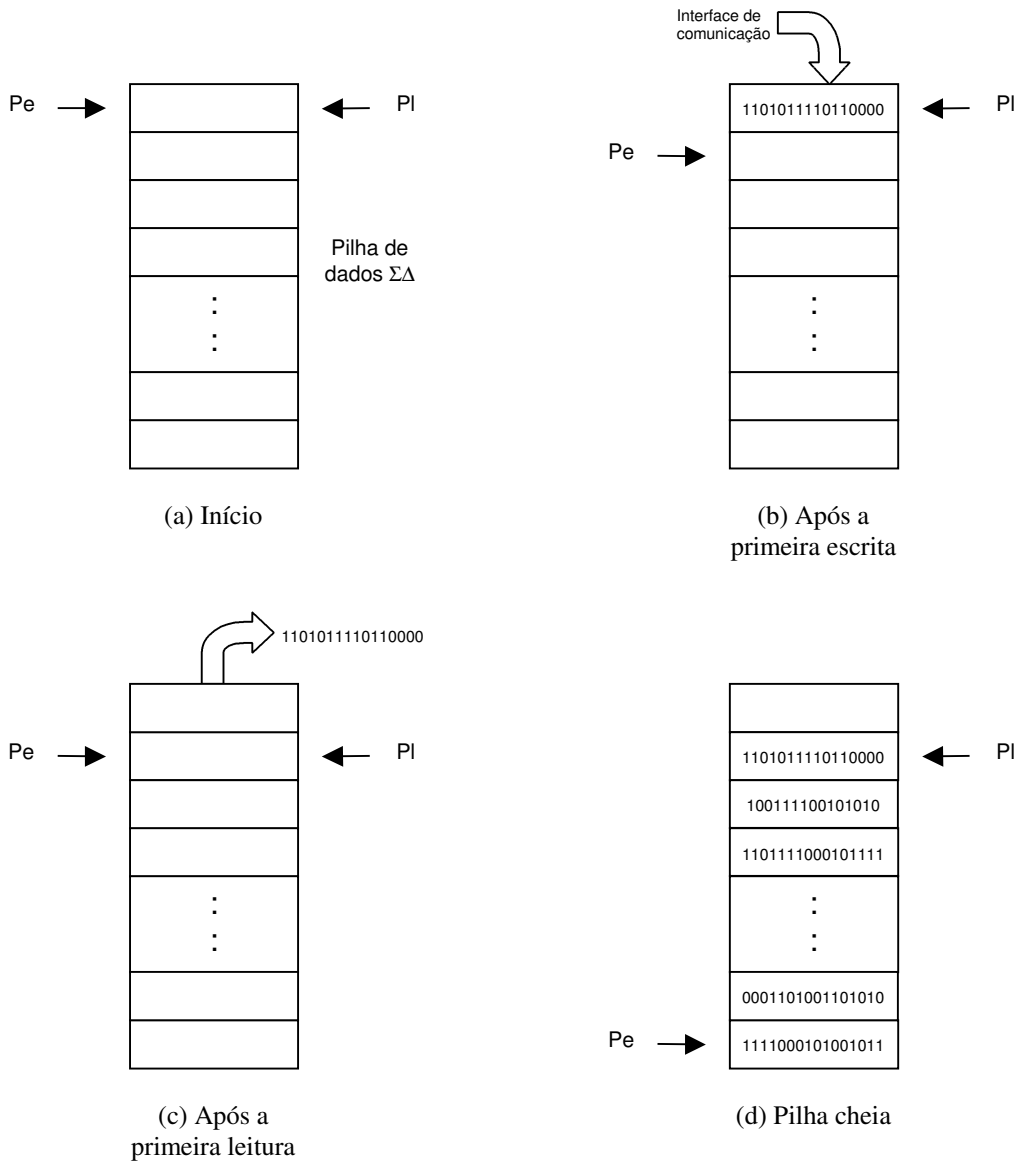


Figura 5.4: Descrição esquemática do controle da pilha de armazenamento temporário

Conforme se observa, num primeiro momento, os dois ponteiros encontram-se na primeira posição de memória reservada para a interface. Se um dado é enviado pela interface em questão, precisa ser escrito em memória. O endereço de escrita é o previsto pelo ponteiro Pe. Após o armazenamento, este ponteiro é incrementado, passando à situação descrita pela figura Figura 5.4(b). Se outro dado não é armazenado, inicia-se o processamento com a retirada do dado escrito anteriormente, por meio do endereço presente no ponteiro Pl, que em seguida é também incrementado (figura Figura 5.4 (c)). Observa-se neste momento que o ponteiro de escrita (Pe) deverá estar adiantado em relação ao ponteiro de leitura (Pl).

A eficiência do procedimento adotado pode ser melhor vista quando se analisa o caso em que a pilha encontra-se cheia e o existem dados ainda não processados. A situação é ilustrada pela figura Figura 5.4 (d). Para que a pilha estivesse pronta para o recebimento seguro de um novo dado, deveria ser primeiro esvaziada, ou seja, todos os dados deveriam ser processados para que pudesse ocorrer novo envio da interface para a

pilha. Isto acarretaria morosidade excessiva para o processamento de um conjunto de dados, e em perda de muitos outros, pois os dados enviados pela interface que não podem ser armazenados em memória são por ela sobrescritos quando há a necessidade de um novo envio.

Os ponteiros de leitura e escrita devem então ser utilizados de forma conjunta, de modo a permitir que quando se retire um dado para processamento usando o ponteiro de leitura, a posição fique automaticamente vaga para o armazenamento de um novo dado proveniente da interface, e o ponteiro de escrita aponte para esta posição.

De forma a atender esta especificação, foi então implementado um esquema onde o ponteiro de escrita corra livremente as posições da pilha destinadas à determinada interface até que o final da pilha seja alcançado. Algum tipo de controle deve então ser realizado para que se possa, com o enchimento da pilha, garantir que o ponteiro de escrita não ultrapassasse o ponteiro de leitura. Se isto ocorresse, a interface sobrescreveria dados existentes em memória.

Verificou-se que neste controle não poderia ser usado um registrador, tendo em vista que o valor deveria ser mantido para as interrupções subseqüentes, de forma independente do andamento do programa. A forma de controle escolhida foi a utilização de um bit de controle em memória. A posição deste bit poderia ser arbitrada, desde que a posição escolhida não fosse em momento algum usada pelo conjunto programacional ou ainda pela pilha de armazenamento.

A posição de memória #FFF3$_h$ que já era anteriormente utilizada pelo módulo de controle, para habilitar ou não a ocorrência de um a interrupção [6] foi escolhida também para a sinalização de pilha cheia. A sinalização é feita através do bit de controle de ponteiro (BCP), que seria representado pelo segundo bit menos significativo armazenado. Quando do enchimento de uma pilha pela primeira vez, o ponteiro de escrita deve ser controlado em função do ponteiro de leitura, utilizando o BCP relativo.

Nesta primeira aplicação desenvolvida, trabalhou-se apenas com a interface ΣΔ atuando no sentido de transmitir dados ao processador. Em aplicações futuras, conforme a demanda o mecanismo desenvolvido para controle da pilha e dos ponteiros de armazenamento ΣΔ também pode ser empregado para o armazenamento temporário das outras interfaces. Para tal, basta definir as pilhas de armazenamento e definir os bits de controle de ponteiro relativos a cada uma das interfaces utilizadas. A Figura 5.5 representa a utilização dos bits da posição de memória #FFF3$_h$.

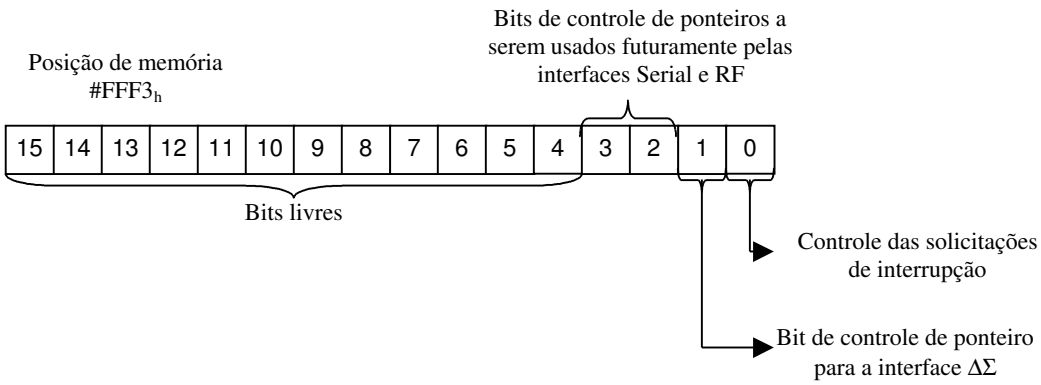


Figura 5.5: Utilização da posição de memória #FFF3$_h$

Deve, ainda, ser considerada a possibilidade de encher-se a pilha e todas as posições de armazenamento anteriores ao ponteiro de leitura, ou seja, ocupar todo o

espaço efetivo destinado ao recebimento de dados pela interface ΣΔ. Esta é uma limitação imposta pelo parâmetro de temporização entre processamento e recebimento de dados.

Neste caso, algumas alternativas foram estudadas para tratar os dados recebidos nestas condições. A primeira delas seria interromper todo o recebimento de dados enquanto os armazenados não fossem devidamente retirados da pilha e processados. Esta opção foi descartada por ocasionar uma grande perda em desempenho do programa. Isto por que, se a disponibilização do dado pela interface fosse totalmente bloqueada para adiantamento dos trabalhos de processamento e, conseqüentemente liberação de espaços em pilha, muitos dados seriam perdidos já que as interfaces de comunicação transmitem continuamente. Optou-se então por ignorar os dados recebidos na condição de pilha cheia. Caso não haja possibilidade de reter a informação proveniente da interface em memória sem que uma outra ainda não processada seja perdida, o processador apenas habilita o reconhecimento da sinalização para aquisição de novo dado, sem que o dado anterior tenha sido guardado.

O ajuste dos ponteiros é feito por meio de um incremento simples em seu conteúdo após ter sido efetuada uma leitura ou uma escrita. Ao final da pilha os ponteiros são deslocados para o topo por meio de testes realizados com seu conteúdo. Na Figura 5.6 observa-se o algoritmo desenvolvido para implementação da idéia dos ponteiros no caso da aquisição de dados.

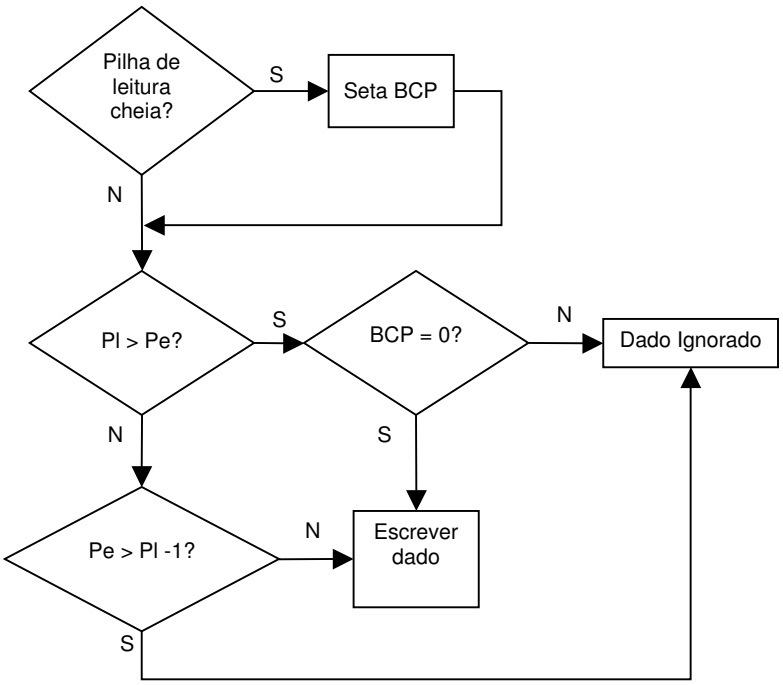


Figura 5.6: Algoritmo de controle do ponteiro de escrita

Também para a leitura de dados da pilha, é necessário o controle do ponteiro de leitura. As operações de recuperação de informações armazenadas em pilha são realizadas são mais simples, devido à consideração de que o ponteiro de leitura corra livremente as posições de memória, e só precise ser ajustado quando do final da pilha, onde o ponteiro deve ser levado ao início da mesma. O algoritmo representativo das operações de controle do ponteiro para realização de leitura de dados da pilha é mostrada na Figura 5.7

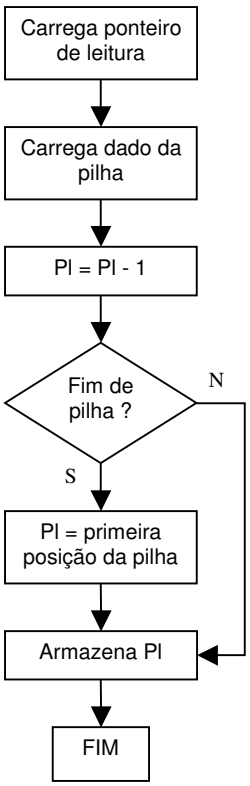


Figura 5.7: Algoritmo para controle do ponteiro de leitura

5.5 Especificações para o tratamento de cada um dos tipos de exceção

Na etapa seguinte, foram desenvolvidas rotinas de tratamento individuais para os cinco tipos de exceção.

5.5.1 Interrupções geradas pela interface ΣΔ

No caso da rotina de tratamento da interrupção gerada pela interface ΣΔ, existe um dado disponível ao processador para captura e armazenamento, já que o fluxo de dados considerado para esta interface é apenas de entrada. A seqüência de operações deve então visar o armazenamento deste dado no espaço reservado em memória, utilizando o esquema de pilhas, e conseqüentemente, o ponteiro de escrita. O processo de aquisição de dados ocorre da seguinte forma: a interface coloca o dado na posição dedicada em memória e sinaliza com uma interrupção, avisando ao processador que um novo dado está sendo enviado. O processador por sua vez, no recebimento da interrupção, pára suas atividades, busca o dado, escrevendo-o em memória, e em seguida sobrescreve a palavra de *setup* anteriormente armazenada por outra, habilitando então a interface para colocação do próximo dado na posição. A Figura 5.9 descreve o algoritmo criado com base nestas informações para o tratamento dos dados provenientes de uma interrupção gerada pela interface ΣΔ.

O primeiro teste realizado verifica se o ponteiro de escrita atingiu o final da pilha (EOS – *End of stack*). Se sim, o BCP (bit de controle de ponteiro) é testado para verificação se a próxima posição de memória a ser ocupada por um dado é válida, ou seja, se não foram feridas as especificações de que o ponteiro de escrita não ultrapasse o de leitura. Se a posição não é válida, a rotina, então, habilita uma nova transmissão e segue para a etapa final do tratamento de exceções, desprezando o dado recebido caso este não possa ser armazenado em memória.

Se a próxima posição de escrita é válida, o dado transmitido pela interface é armazenado em memória e o ponteiro de escrita é ajustado, ou pelo incremento de seu conteúdo, ou pela inserção do primeiro endereço da pilha, caso o apontamento atual seja realizado para a última posição. E após o armazenamento do dado, segue-se a etapa final da rotina de tratamento de exceções. A seqüência de tarefas a ser realizada para implementação eficiente do uso de ponteiros na aquisição de dados é descrita na Figura 5.9.

Assim como a aquisição de dados provenientes da interface ΣΔ requer que controle do ponteiro de escrita em pilha seja concomitante ao armazenamento do dado, a retirada do dado da pilha para posterior processamento, dentro da aplicação, também deveria ter associada a ela o controle do ponteiro de escrita em pilha. De forma a atender às necessidades do esquema de utilização de ponteiros, o algoritmo apresentado no capítulo 4 referente à etapa de processamento dos dados (Figura 4.3) foi modificado, de forma a incluir o controle do ponteiro de leitura. A Figura 5.8 representa o novo algoritmo.

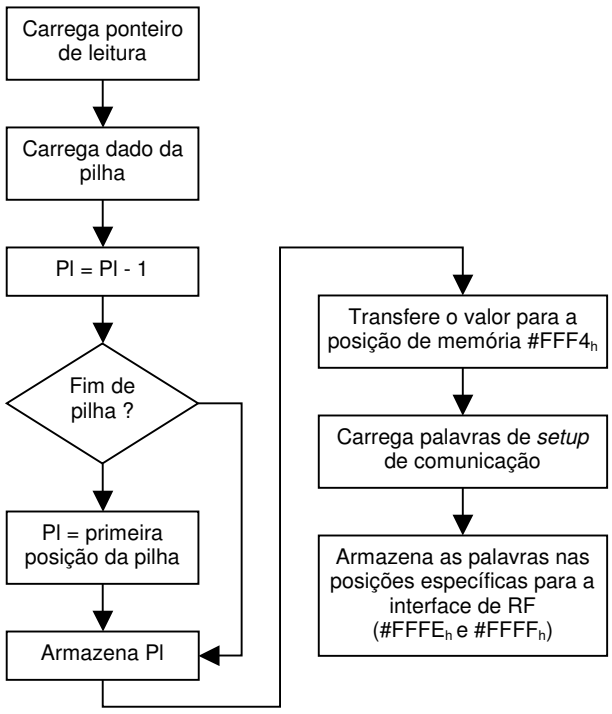


Figura 5.8: Algoritmo para a etapa de processamento da aplicação incluindo o controle do ponteiro de leitura

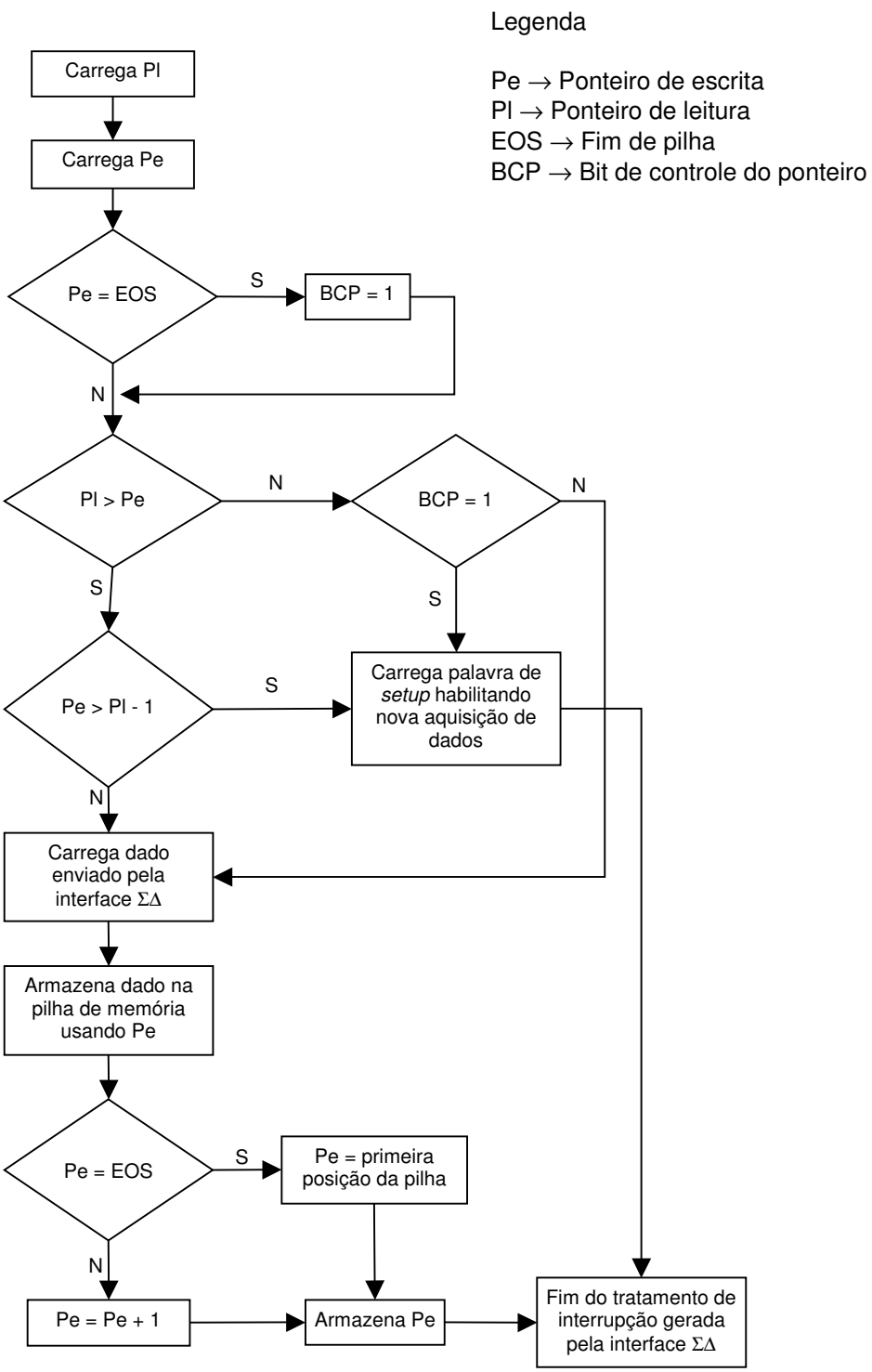


Figura 5.9: Algoritmo para armazenamento em pilha dos dados disponibilizados pela interface ΣΔ

5.5.2 Interrupções geradas pelas interfaces de RF e serial

Ainda que o objetivo da aplicação desenvolvida seja trabalhar apenas com as interfaces de entrada e saída considerando que as mesmas propiciem fluxo de dados unidirecionais, as interfaces serial e de RF foram especificadas em projeto com caráter bidirecional. Neste trabalho foi optou-se por considerar as interfaces de RF e serial como bidirecionais, como forma de tornar a aplicação o mais próxima possível de

programas que venham a ser utilizados em escala quando da fabricação do chip. Desta forma, são criados subsídios para a implementação do controle completo do fluxo de dados para as três interfaces. Para aplicações posteriores onde seja necessária a consideração do caráter bidirecional das duas interfaces, basta modificar o tratamento de interrupções aqui proposto, de forma a armazenar os dados recebidos em memória para posterior processamento.

Para as interrupções geradas pelas interfaces serial e de RF, deve ser primeiro determinada a causa desta interrupção que, considerando o fluxo de dados bidirecional, pode ocorrer no sentido da interface para o processador, bem como o inverso: pode ser gerada para aquisição de dados pelo processador ou ainda para sinalização do resultado de uma transmissão anterior do mesmo.

A identificação da circunstância que ocasionou a interrupção pode ser feita através do teste da palavra de *setup* corrente, verificando se a palavra corresponde à habilitação da transmissão ou da disponibilização de dados por parte da interface.

No caso da interrupção gerada pela interface para sinalização do resultado de uma transmissão anterior do processador, deve ser checada primeiramente a palavra de *status* referente à interface. O "registrador" de *status* é o utilizado pela interface para indicação de erro na transmissão (15 bits mais significativos) e para indicação do estado da interface (bit 0), que pode estar pronta para um novo recebimento ou não. As palavras de *status* para as interfaces Serial e de RF são mostradas na sessão 2.1.2.2, Figura 2.3.

Na verificação da ocorrência de um erro de transmissão, deveria ser determinada a seqüência de atividades a ser realizada. A primeira opção analisada foi a tentativa de retransmissão do dado. Este procedimento obrigaria a parada das atividades do processador, para que se evitasse a solicitação de transmissão de um novo dado enquanto o último não tivesse atingido seguramente seu destino. Neste caso, o desempenho do programa seria bastante prejudicado. Ainda, chegar-se-ia a uma situação na qual os dados seriam capturados pelo processador quando da sinalização da interface de entrada e não se observaria nenhuma saída de dados processados, e no ambiente externo ao microprocessador e não se saberia o que ocasionou este conjunto de circunstâncias. Optou-se, então, pelo envio de algum tipo de informação à interface de saída, para indicação da ocorrência do erro como tratamento primário dos erros de transmissão gerados no envio de informações do processador às interfaces.

O envio destes dados deveria preencher a lacuna de informações de saída caso fosse configurado o erro. A interface escolhida foi a serial por ser a interface padrão de saída de dados de controle para a aplicação desenvolvida. Assim, se asseguraria que um erro de transmissão poderia ser percebido externamente.

Caso fosse verificado que a sinalização da interface indica, não a ocorrência de erro, mas, a transmissão com sucesso, a interface deveria ser configurada de modo a habilitar uma nova transmissão de dados processados. Esta habilitação é feita através da posição reservada em memória para armazenamento do *status* da interface. A habilitação é feita setando o bit menos significativo que indica, ao processador, que nova transmissão pode ser feita.

Para a interrupção gerada pela aquisição de dados originados nas interfaces serial e de RF para a aplicação aqui descrita, optou-se também pelo envio de um código através de uma das interfaces para sinalização externa da situação em questão. Novamente, foi escolhida a interface serial.

A seqüência de ações para os casos de interrupções geradas pelas interfaces Serial e de RF é mostrada na Figura 5.10.

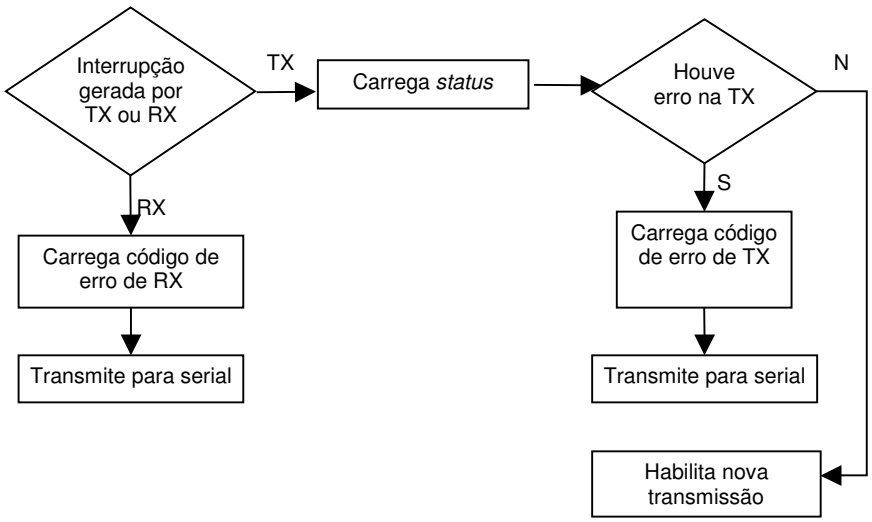


Figura 5.10: Algoritmo descritivo do tratamento dado às exceções geradas pelas interfaces serial e de RF

5.5.3 Exceções geradas pela ocorrência de *overflow* ou erro no endereçamento

Na ocorrência de *overflow* aritmético, ou erro no endereçamento, as atividades a serem realizadas devem ser: (1) o envio do conteúdo do registrador $int para a interface serial, como forma de controle por parte do usuário da execução do programa; (2) envio de código de ocorrência de *overflow*/erro de endereçamento para interface de RF .

Na verificação de uma interrupção gerada por erro de *overflow,* duas alternativas foram estudadas com relação ao tratamento a ser dado ao dado. A primeira seria tentar novamente a operação que gerou o *overflow*. No entanto, esta alternativa poderia ser bastante ineficiente no sentido que a operação que implicou num *overflow* teria grandes possibilidades de gerar o mesmo erro, se realizada com os mesmos dados. Isto levaria o programa a um *loop* infinito.

A segunda alternativa seria o descarte das informações que, quando processadas, levaram ao *overflow*.

Como discutido, um dos objetivos deste trabalho é implementar uma rotina de tratamento de exceções básica, não definitiva, que possibilite a validação da linguagem proposta para o SCW e das especificações propostas para o *hardware*. Nesse ponto, não existe preocupação com a utilização posterior dos dados provenientes de uma interrupção. A maior preocupação é que a rotina de tratamento possua funcionalidade suficiente para permitir o teste da linguagem utilizada e a validação da estrutura de *hardware*.

Assim sendo, para a ocorrência de *overflow* aritmético, foi adotado como procedimento o descarte dos dados que geraram as exceções, por ser o tratamento mais simples e mais compatível com a situação dada. Porém, esta escolha não implica em fixação deste tipo de tratamento, que pode ser modificado em versões posteriores sem maiores modificações na estrutura sugerida neste trabalho. O descarte neste caso deve ser feito apenas reiniciando a execução do programa, ou seja, fazendo com que $pc retorne ao endereço $\#0001_h$.

Para a situação de erro de endereçamento, foram feitas as mesmas considerações usadas na análise do caso de *overflow*. Caso se tentasse repetir o desvio que ocasionou a busca num endereço inválido de memória, as chances de se obter o mesmo erro seriam grandes. Neste caso, optou-se também pelo descarte do dado gerador do erro e pela sinalização externa da ocorrência do mesmo por meio de algum tipo de código. A seqüência de atividades a ser realizada em caso de *overflow* ou erro de endereçamento pode ser observada na Figura 5.11.

5.6 Etapa final do tratamento das exceções

Após o tratamento dos dados recebidos pelas interfaces ou da ocorrência de erros internos, pode-se passar à etapa final da rotina de tratamento de exceções. Esta etapa será composta: (1) pela habilitação da ocorrência de nova interrupção, desabilitando o bit zero da posição de memória a ser usada pela unidade de controle para bloqueio das interrupções (#$FFF3_h$); (2) pela restauração dos conteúdos dos registradores salvos em memória quando da execução da etapa inicial do tratamento e (3) pela recuperação do valor de $pc e subseqüente retorno à posição posterior de memória na qual aconteceu a interrupção.

A seqüência de realização destas operações é muito importante tendo em vista que, após a recuperação do contexto (registradores salvos), qualquer modificação em seus conteúdos pode causar erros no programa. Depois de uma análise profunda das conseqüências de posicionamento seqüencial incorreto destas atividades, algumas conclusões foram obtidas. Primeiro, o retorno dos registradores colocados em memória acarretaria a perda do conteúdo de, pelo menos um, pois todo o endereçamento do sistema SCW se realiza por meio de apontamento indireto, ou seja, qualquer endereçamento à memória deve ser feito utilizando uma instrução de carregamento do endereço num registrador e, em seguida, uma instrução de carregamento da palavra contida no endereço presente no mesmo registrador.

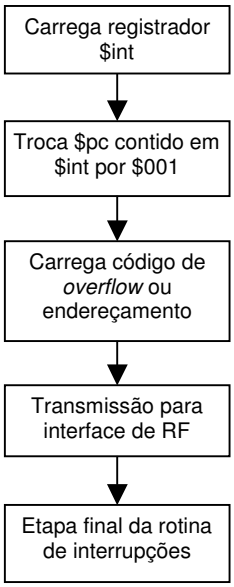


Figura 5.11: Esquemático da seqüência de operação para ocorrência de overflow aritmético ou erro de endereçamento

Foi observado também que habilitar a ocorrência de nova interrupção deveria ser a última tarefa executada pela rotina de tratamento de exceções. No entanto, neste caso, o registrador $pc teria seu conteúdo modificado pelas instruções subseqüentes, relativas à habilitação de nova interrupção, e o programa não retornaria à posição de memória adequada. Uma opção estudada foi a utilização do registrador $ra para armazenamento do endereço de retorno da rotina de tratamento de interrupções. Esta hipótese foi descartada pelo fato de que a interrupção pode acontecer em meio a uma subrotina dentro do programa principal. Se o conteúdo do registrador $ra fosse modificado pela rotina de tratamento de dados, no retorno ao programa, mais especificamente a uma subrotina, a mesma se perderia, ocasionando num *crash* do programa ou talvez num *loop* infinito.

Avaliando as conseqüências de uma inversão de atividades e a realização da tarefa de ajuste do registrador $pc antes da escrita na palavra responsável pela habilitação das interrupções (bit 0 da posição $\#FFF3_h$), verificou-se que, levando em conta o fato de que o ajuste do $pc seria realizado com apenas algumas instruções de carregamento de conteúdos de memória, o tempo gasto para este ajuste seria pequeno em relação ao tempo necessário para a disponibilização do próximo dado pela interface. Considerando a utilização de cinco instruções para restabelecer o conteúdo do $pc, numa média máxima de cinco ciclos por instrução, seriam necessários 25 ciclos.

O *clock* do processador especificado trabalha à 250MHz, então o tempo gasto no processo de devolução do conteúdo do registrador $pc seria de aproximadamente $0{,}075 \times 10^{-6}$s, enquanto que o tempo para envio de uma palavra de 8 bits pela interface serial, considerando a taxa padrão [4], serial de $1{,}25 \times 10^{-4}$ s. Assim sendo, a próxima interrupção poderia acontecer no mínimo, após 12,5 ms após a habilitação das interrupções por parte da unidade de controle, tempo suficiente para executar as instruções relativas ao ajuste do registrador $pc e o retorno ao programa.

Outra opção analisada foi utilizar um dos registradores no recebimento do endereço para volta do programa após o tratamento de exceções e se realizar o desvio incondicional final relativo ao conteúdo deste registrador ao invés de $pc ou $ra, conforme havia sido primeiramente estudado. Observou-se que esta era a escolha mais eficiente, tendo em vista que não se limitaria a taxa de transmissão de nenhuma das interfaces e tampouco se perderia em desempenho do programa, apesar de tornar um dos registradores inflexível.

Assim, optou-se pela seqüência descrita pela Figura 5.12.

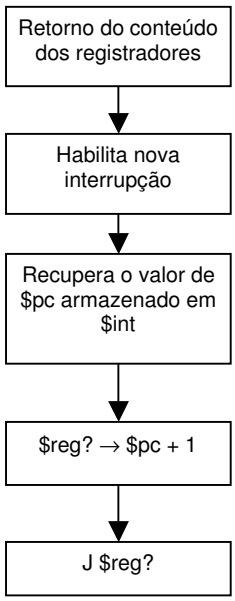


Figura 5.12: Esquemático da seqüência de atividades para a etapa final do tratamento de exceções

Depois da análise detalhada dos algoritmos elaborados para cada uma das etapas a serem percorridas pela rotina de tratamento de exceções, construiu-se o algoritmo final, mostrado na Figura 5.13. Observa-se que aqui são colocadas de forma simplificada todas as atividades a serem realizadas pela rotina de tratamento de exceções, desde o salvamento inicial do contexto até a finalização, com o desvio para o endereço de retorno.

Os tratamentos das interrupções geradas pelas interfaces de RF e serial são melhor descritos pela Figura 5.14. Nestas figuras, todas as transmissões feitas à interface serial são realizadas por meio de chamadas à subrotina desenvolvida para transmissão de dados pela interface.

O algoritmo representativo do tratamento de interrupções geradas pela interface ΣΔ foi mostrado anteriormente na Figura 5.7. Os esquemáticos seqüenciais correspondentes ao tratamento de exceções geradas por *overflow* aritmético ou erro de endereçamento encontram-se dispostos na Figura 5.11.

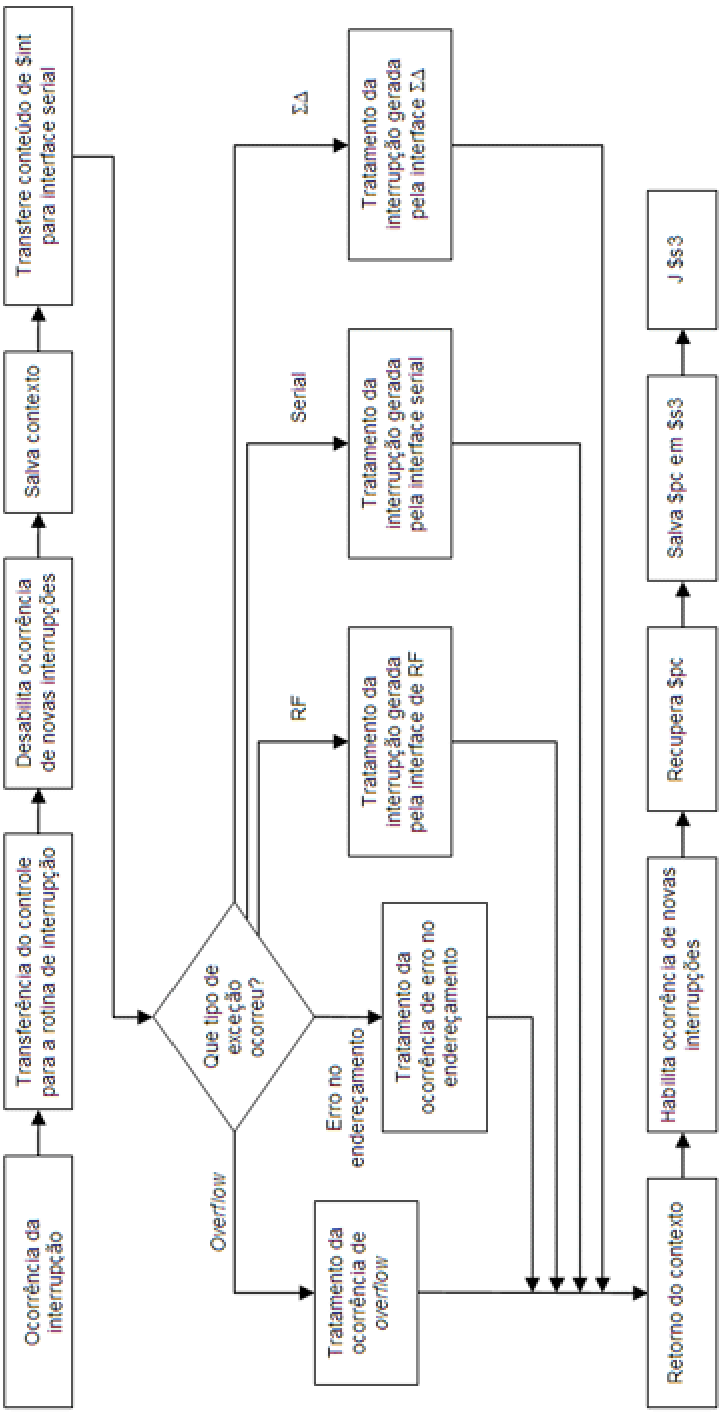


Figura 5.13: Algoritmo completo para a rotina de tratamento de exceções

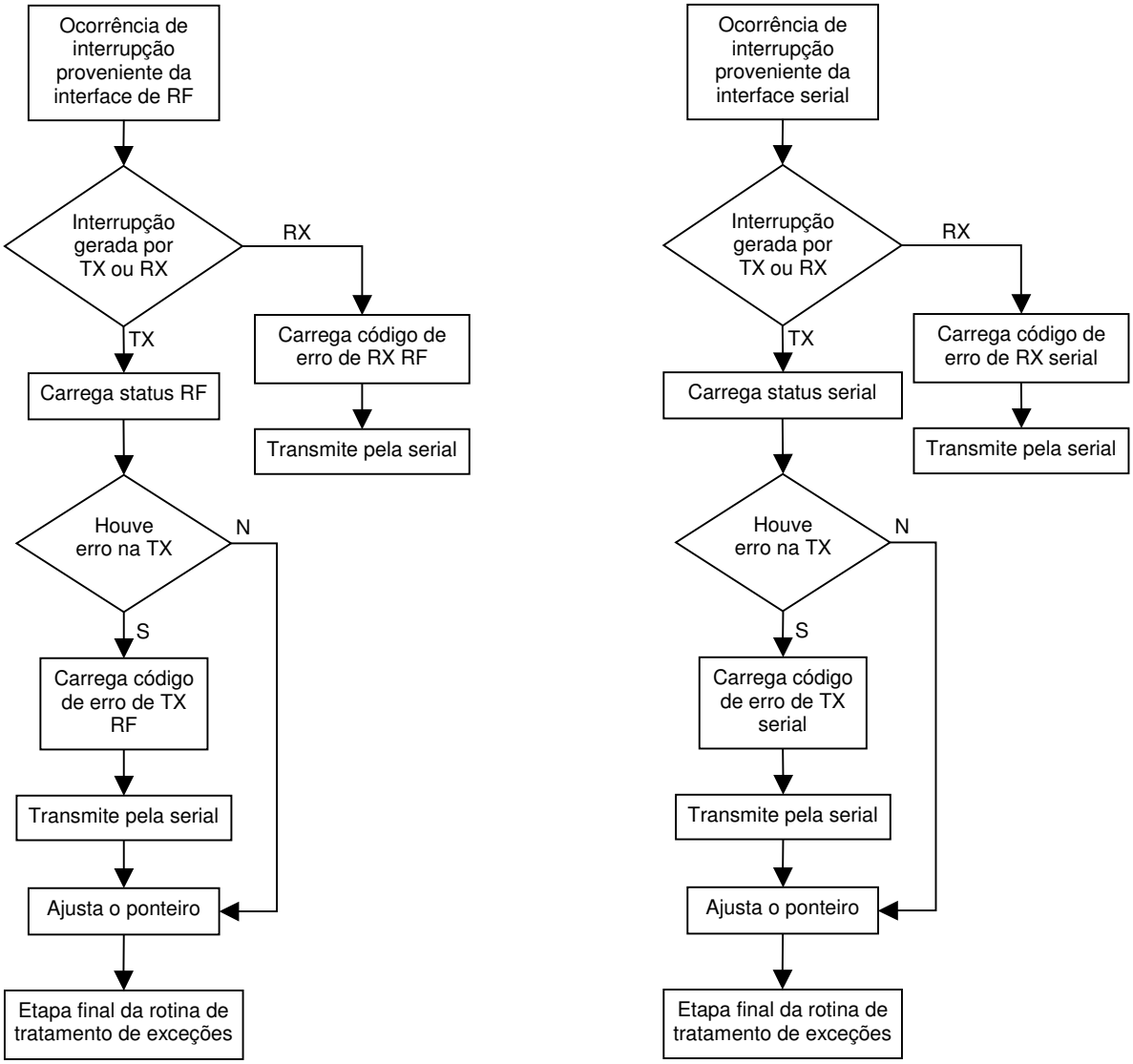


Figura 5.14: Detalhamento das rotinas de tratamento de exceções geradas pelas interfaces Serial e de RF dentro da Figura 5.13

5.7 A rotina de tratamento das interrupções em linguagem de máquina

Considerando o algoritmo completo desenvolvido para a rotina de tratamento de exceções, foi possível a construção do programa em linguagem máquina da rotina de tratamento de exceções. Numa primeira etapa, optou-se por usar a linguagem de máquina como base, ao invés, da de montagem, tendo em vista a maior simplicidade de operação e maior inteligibilidade. Porém, a linguagem de máquina, por ter suas instruções constituídas de outras microinstruções, não possibilita a montagem direta em posições de memória e, conseqüentemente, o cálculo de desvios e endereçamentos. Então, optou-se por construir os programas fazendo uso de *labels* para indicar os

endereços necessários, bem como para cálculo dos *offsets* para desvios condicionais.

No desenvolvimento da primeira etapa da rotina de tratamento de dados, que é a etapa onde se desabilita a ocorrência de novas interrupções, bem como na etapa de salvamento do contexto, observou-se que o uso de pseudoinstruções deveria ser evitado. Isto, porque, as pseudoinstruções, quando da montagem em memória, são desmembradas em microinstruções que podem trabalhar com registradores outros que não explicitados na sintaxe da pseudoinstrução. Na primeira etapa do programa, o uso de pseudoinstruções poderia ser desastroso no sentido de que, se não fossem bem estudadas as pseudoinstruções inseridas e seus conteúdos, algum dos registradores pode ter seu conteúdo modificado antes da etapa de salvamento do contexto, o que ocasionaria em erro posterior no funcionamento

do programa. Para a evitar este tipo de problema, optou-se por trabalhar apenas com as microinstruções na etapa inicial da rotina, permitindo que o tratamento com registradores pudesse ser visível ao usuário.

Verificou-se que um dos problemas na primeira etapa da rotina de tratamento, na qual é realizado o salvamento do contexto, refere-se ao armazenamento numa pilha específica de todos os 15 registradores utilizados pelo microprocessador (O registrador $zero tem valor fixo, não sendo necessário seu armazenamento).

O endereçamento de memória se faz pelo método de apontamento indireto, ou seja, para se endereçar qualquer posição de memória carrega-se antes o endereço de destino num registrador. Verifica-se que não é possível salvar em memória os 15 registradores, sem que se perca o conteúdo de ao menos um, que deve ser usado no endereçamento. O questionamento subseqüente natural é que registrador utilizar. No caso da rotina de tratamento de exceções aqui descrita, foi utilizado um dos registradores salvos ($s3), por ser a única classe de registradores com 4 disponíveis. Mas esta escolha não fixa de maneira alguma a utilização deste registrador. É possível que em aplicações futuras, ou em versões posteriores da rotina de tratamento de exceções, o programador opte pela utilização de qualquer outro registrador, não sendo necessária nenhuma mudança na estrutura da rotina de tratamento. Apenas deve ser levado em conta, na escolha, o fato de que o registrador selecionado, seja ele temporário ou, por exemplo, o ponteiro de pilha ($sp), terá seu conteúdo perdido quando da ocorrência de uma interrupção. O objetivo é manter a flexibilidade da rotina de tratamento criada, de modo a permitir que o programador utilize todos os recursos possíveis da maneira que julgar mais adequada, sem que se perca em desempenho.

Em seguida, analisando a etapa de envio de dados de controle referentes à interrupção para a porta serial, algumas opções foram apontadas. Primeiramente a inclusão da seqüência de execução completa da rotina de transferência de dados para a interface serial dentro da rotina de tratamento de exceções. A segunda opção seria se chamar, através de uma instrução de desvio incondicional, a subrotina de envio de dados para a interface serial implementada dentro do contexto da aplicação proposta.

As duas alternativas seriam viáveis, tendo em vista que a inclusão da rotina de envio de dados dentro da rotina de tratamento de exceções não aumentaria de forma significativa as posições de memória ocupadas pela mesma e nem tornaria o programa mais lento e menos eficiente. A vantagem de se utilizar uma subrotina para o envio de dados para a interface serial é tornar o programa modular, com blocos reutilizáveis por programadores que, futuramente, devam desenvolver aplicações para o SCW, e também este princípio foi adotado para a construção da aplicação.

A flexibilidade que se deseja alcançar com a utilização de blocos programacionais reutilizáveis já é plenamente obtida através da utilização da programação em linguagem de máquina. Não fixando endereços ou registradores, o programador pode fazer uso da liberdade de recursos. Desta maneira, julgou-se mais interessante a utilização da chamada à subrotina de transferência de dados dentro do conjunto de tratamento de dados. Além da redução no tamanho do programa, esta escolha representa, também, maior maleabilidade para o programador.

Na Tabela 2.1, observa-se a etapa inicial da rotina de tratamento de interrupções que é descrita pelo esquemático da Figura 5.2.

Tabela 5.1: Etapa inicial da rotina de tratamento de exceções em linguagem de máquina

Label	Instrução	Comentários
Interrupção	Lui $s3, FF	Desabilita interrupção setando o bit 0 do endereço $FFEB
	Addi $s3, EB	
	Sw $s3, $s3, $zero	
	Lui $s3, MSBSalv1	Carrega primeiro endereço de salvamento do contexto
	Addi $s3, LSBSalv1	
	Sw $ra, $s3, $zero	Guarda os 13 registradores de contexto usando $s3 como base
	Addi $s3, 0001	
	Sw $sp, $s3, $zero	
	Addi $s3, 0001	
	Sw $t0, $s3,$zero	
	Addi $s3, 0001	
	Sw $t1, $s3, $zero	
	Addi $s3, 0001	
	Sw $t2, $s3, $zero	
	Addi $s3, 0001	
	Sw $a0, $s3, $zero	
	Addi $s3, 0001	
	Sw $a1, $s3,$zero	
	Addi $s3, 0001	
	Sw $a2, $s3, $zero	
	Addi $s3, 0001	
	Sw $s0, $s3, $zero	
	Addi $s3, 0001	
	Sw $s1, $s3, $zero	
	Addi $s3, 0001	
	Sw $s2, $s3, $zero	
	Addi $s3, 0001	
	Sw $int, $s3, $zero	
	Addi $s3,0001	
	Sw $gp,$s3,$zero	
	Lwi $t1,dado_serial	Transfere conteúdo de $int para saída serial
	Sw $int,$t1,$zero	
	Lwi $s1,TX_Serial	
	Jal $s1,0	
	Andi $t0,$int,$000000000000 1000_b	Testes sucessivos com o conteúdo de $int para identificar que tipo de interrupção aconteceu e transferir controle para a rotina de tratamento adequada
	Bgt $t0,$zero, 9(Endereçamento)	
	Andi $t0,$int, 0000000000000100_b	
	Bgt $t0,$zero, 8(*Overflow*)	
	Andi $t0,$int,$0000 0000 0000 0 0 01_b	
	Bgt $t0, $zero, 7(RF)	
	And $t0, $int, $0000 0000 0000 0 0 10_b	
	Bgt $t0, $zero, 6(Serial)	
	And $t0, $int, $0000000000001 0 11_b	
	Bgt $t0, $zero, 5 ($\Sigma\Delta$)	
	Jd IntEndereçamento	Desvios para rotinas de tratamento individuais
	Jd zero,IntOverflow	
	Jd zero,IntRF	
	Jd zero,IntSerial	

Nestas primeiras versões dos programas em linguagem de máquina não foram utilizados endereços para constantes ou desvios. Os endereços em memória serão calculados quando da montagem em microinstruções.

Observou-se nesta etapa que o contexto (conteúdo dos registradores) a ser salvo para correto funcionamento da rotina tratamento de interrupções possuiria 13 registradores a serem salvos. O registrador $s3 por ter sido escolhido para emprego nos endereçamentos à memória iniciais dentro da rotina, não precisaria ser armazenado em memória, pois teria seu conteúdo modificado ao longo da etapa final da rotina de tratamento de exceções.

O registrador $pc também não precisaria ser salvo, por ter seu conteúdo contido no registrador $int no campo "Endereço". Quando do retorno ao programa seria necessário, apenas, isolar os 12 bits referentes ao campo no registrador $int e posicioná-los no registrador $pc.

Para enviar o registrador $int à interface serial, o dado deve primeiro ser carregado na posição que será posteriormente utilizada pela subrotina de transmissão de dados para a interface. E em seguida, é realizado o salto para execução da subrotina. Estes passos são verificados nas três últimas linhas da Tabela 2.2.

A etapa que se segue, após o envio do registrador $int para a interface serial, é a de testes relativos ao tipo de interrupção ocorrido. Para tal utilizou-se a

Figura 3.1 onde são observadas as codificações utilizadas para identificação da unidade geradora da exceção através do conteúdo do registrador $int. Os testes foram realizados usando a instrução representativa do operação de E lógico. A utilização deste esquema permite testar o conteúdo de apenas um único bit. Optou-se pela realização dos testes pela instrução Andi por ter o formato adequado para maior inteligibilidade do programa, e por possibilitar a realização da operação de E lógico bit a bit com relação a uma constante de 16 bits. Dado que esta, como as outras operações lógicas realizadas pelo set de instruções elaborado pelo SCW, armazena o resultado da operação num registrador, após a realização da operação lógica, o conteúdo deste registrador pode ser testado a fim de verificar a ocorrência do valor procurado. O teste e desvio deveria ser realizado usando instruções de desvio condicional, usando os registradores que se deseja comparar.

O resultado da operação de E lógico utilizada para teste só poderia gerar valor 0 ou valor maior 1 para o registrador de armazenamento. Por este motivo, foi escolhida a pseudo-instrução Bgt (*branch if greater than*). No entanto as instruções de desvio do *set* criado possibilitam saltos de até 7 palavras. As rotinas para as quais os desvios deveriam ser feitos no caso da determinação do tipo de exceção ocorrido, seriam superiores a 7 posições. Assim, implementou-se o esquema do teste e desvio para uma instrução de desvio incondicional ao endereço da rotina de tratamento individual procurada. O desvio incondicional, por ser uma instrução do tipo J possibilita o salto para até 2^8 posições.

Com o conhecimento da exceção ocorrida, pode-se então passar ao tratamento específico dos dados a ser efetuado para cada tipo de interrupção.

O tratamento a ser realizado para dados recebidos é descrito pelo algoritmo na Figura 5.9 e por meio dele, foi criado o bloco de programa presente na Tabela 5.2 que representa a rotina para interrupção gerada pela interface ΣΔ.

Tabela 5.2: Bloco de programa em linguagem de máquina para o tratamento de exceções geradas pela interface ΣΔ

Label	Instrução	Comentários
IntSD	Lwi $t0,pe_SD	Carrega os dois ponteiros (leitura e escrita) para testes e verificação da possibilidade de escrita em pilha
	Lw $t0,$t0,$zero	
	Lwi $t1,pl_SD	
	Lw $t1,$t1,$zero	
	Lwi $s1, ultima posição da pilha	Testa se a próxima posição a ser escrita é a última da pilha
	Beq $t0, $s1, 2	
	Jd not_EOS_SD	Pula para próximo teste
	Lwi $s0,FFF3	Caso tenha sido alcançado o EOS, habilita BCP, usado para sinalização
	Lwi $s1,0000 0000 0000 0010	
	Sw $s1,$s0,$zero	
Not_EOS_SD	Bgt $t1, $t0, 3	Se pl_SD > pe_SD, testa $FFF3
	Lwi $s0, FFF3	Se pl_SD <pe_SD e BCP=0 escreve dado. Se BCP=1, posição inválida.
	Lw $s0,$s0,$zero	
	Beq $s0,$zero, 7	
	Jd etapa_final	Finaliza tratamento
	Beqi $s0, 0000, 5	Se pl_SD > pe_SD e BCP=1 escreve dado
	Lwi $a0, 0001	Testa se pl_SD - 1 < pe_SD
	Sub $a1, $t1, $a0	
	Blt $t0, $a1, 2	
	J etapa_final	Finaliza tratamento
Wr_SD	Lwi $s1, FFF4h	Carrega dado enviado pela SD
	Lw $s1, $s1, $zero	
	Lwi $t0, pe_SD	Carrega ponteiro de escrita SD
	Lw $t0, $t0, $zero	
	Sw $s1, $t0, $zero	Armazena dado enviado na pilha usando o ponteiro
	Bnei $t0, ultima_posição_da_pilha, 4	Se a última posição utilizada para escrita foi a última da pilha, retorna o ponteiro ao topo da pilha
	Lwi $t2, primeira_posição_da_pilha	
	Sw $t2, $t1, $zero	
	Jd etapa_final	
	Addi $t0, 0001	Se não é a última posição, ajusta o ponteiro de escrita, somando 1 ao seu conteúdo
	Lwi $t1, pe_SD	
	Sw $t0, $t1, $zero	
	Jd etapa_final	Finaliza tratamento

Ainda para tratamento das exceções originadas nas interfaces de RF, foi gerado um outro bloco de programa, presente na Tabela 5.3 que teve como base preliminar o algoritmo presente na Figura 5.14.

Tabela 5.3: Rotina em linguagem de máquina para tratamento das interrupções geradas pelas interfaces de RF

Label	Instrução	Comentários
IntRF	Lwi $t1,FFFD	Teste para identificar interrupção gerada de transmissão ou recepção
	Lw $t1,$t1,$zero	
	Andi $t1,0000 0000 0001 0000	
	Beq $t1,$zero, RX_RF	Se interrupção gerada por recepção, envia código para Serial
	Lwi $t1, FFFD	Carrega palavra de status
	Lw $t1,$t1,$zero	
	And $t1, 0000 0000 0000 0010	Testa se houve erro, usando os 15 bits mais significativos do status. Se um deles é diferente de 0, houve erro.
	Beq $t1,$zero, ErroTX_RF	Se houve erro, desvia para rotina adequada
Not_erro_RF	Lwi $t0,0000 0000 0000 0001	Carrega ready = 1 no status da interface, significando que ela deve estar pronta para nova transmissão
	Lwi $t1, FFFD	
	Sw $t0,$t1,$zero	
ErroTX_RF	Lwi $t0, Cód _Erro_TX_RF	Carrega código de erro de TX RF e armazena no dado a ser usado pela interface de transmissão para serial e chama a subrotina de envio de dados para serial
	Lw $t0, $t0, $zero	
	Lwi $t1, Dado_Serial	
	Sw $t0,$t1,$zero	
	Jal TX_Serial	
	J etapa_final	Finaliza tratamento
RX_RF	Lwi $t0, Cód _Erro_RX_RF	Carrega código de erro de RX RF
	Lw $t0,$t0,$zero	

Label	Instrução	Comentários
	Lwi $t1, Dado_Serial	Armazena o código de erro na posição especificada e chama a subrotina de envio de dados para a Serial
	Sw $t0,$t1,$zero	
	Jal TX_Serial	
	Jd etapa_final	Finaliza tratamento

No blocos de programação descritos tanto para a interface ΣΔ quanto para as interface de RF apresentados nas tabelas 5.2 e 5.3, verifica-se a utilização de algumas constantes tais como pe_SD, pl_SD, última_posição_da_pilha e primeira_posição_da_pilha. Estas constantes deverão ser armazenadas em memória quando do carregamento inicial do programa pela interface serial, e terão endereços específicos, que serão utilizados no programa montado final, para referência às mesmas constantes.

Nesta etapa, foi necessário decidir o posicionamento das constantes dentro da memória entre o conjunto programacional. A possibilidade de se colocar as constantes no início da memória não foi considerada, dado que a especificação do projeto almeja a inicialização da execução das instruções e, conseqüentemente do programa, a partir da posição $\#0002_h$. As opções de localização das constantes entre a aplicação básica e a rotina de tratamento de exceções ou ao final do bloco programacional eram indiferentes. Optou-se então pela colocação do bloco de constantes entre a aplicação básica e a rotina de tratamento de interrupções.

Neste momento faz-se necessária a criação da codificação das informações a serem enviadas como erros para as respectivas interfaces de comunicação. Para criação deste código partiu-se de números comuns. Considerando que nesta primeira aplicação a ênfase é dada mais ao conjunto programacional, e as constantes aqui definidas podem ser modificadas em versões posteriores de programas, foram arbitrados números quaisquer. A Tabela 5.4 mostra a codificação adotada com suas respectivas interpretações.

Tabela 5.4: Codificação dos erros gerados na transmissão ou recepção de dados pelas interfaces

Valor	Erro Correspondente
0000000000000001	Erro de TX para RF
0000000000000010	Erro de RX para RF
0000000000000011	Erro de TX para Serial
0000000000000000	Erro de RX para Serial
1000000000000001	Erro de *Overflow*
1000000000000000	Erro de Endereçamento

Na Tabela 5.5 são apresentadas algumas das constantes utilizadas na rotina de tratamento de exceções juntamente com uma descrição de seu conteúdo.

Tabela 5.5: Constantes utilizadas no bloco programacional

Constante	Valor a ser assumido	Interpretação
Pl_AD	Ponteiro leitura pilha A/D	Posições de memória que contém outras posição de memória dentro das pilhas respectivas
Pe_AD	Ponteiro escrita pilha A/D	
Cód_Erro_TX_RF	Vide tabela 5.4	Código para erro na transmissão à RF
Cód_Erro_TX_Serial		Código para erro na transmissão à Serial
Cód_Erro_RX_RF		Código para erro na recepção RF
Cód_Erro_RX_Serial		Código para erro na recepção Serial
Cód_Erro_Overflow		Código de erro de *Overflow*
Cód_Erro_Endereçamento		Código de erro de endereçamento
Proc	Dado a ser processado	Dado proveniente da A/D
Const_mul	Constante para multiplicação	Constante para multiplicação
Result	Resultado	Resultado do processamento
Dado_RF	Dado de saída RF	Dado a ser usado pela rotina de envio RF
Dado_serial	Dado de saída serial	Dado a ser usado para rotina de envio Serial
Setup_TX_Serial	Setup de transmissão Serial	Palavra de setup de transmissão Serial

Constante	Valor a ser assumido	Interpretação
Setup_TX_RF1	Setup de Transmissão RF	1ª Palavra de setup de transmissão RF
Setup_TX_RF2	Setup 2 de transmissão RF	2ª Palavra de setup de transmissão RF
Salv1	Pilha de salvamento do contexto	Posições de memória reservadas para o salvamento do contexto na ocorrência de uma interrupção
Salv2	Pilha de salvamento do contexto	
Salv3	Pilha de salvamento do contexto	
Salv4	Pilha de salvamento do contexto	
Salv5	Pilha de salvamento do contexto	
Salv6	Pilha de salvamento do contexto	
Salv7	Pilha de salvamento do contexto	
Salv8	Pilha de salvamento do contexto	
Salv9	Pilha de salvamento do contexto	
Salv10	Pilha de salvamento do contexto	
Salv11	Pilha de salvamento do contexto	
Salv12	Pilha de salvamento do contexto	
Salv13	Pilha de salvamento do contexto	
Salv14	Pilha de salvamento do contexto	

Para a ocorrência de *overflow* aritmético ou erro no endereçamento, foi utilizado o algoritmo descrito pela Figura 5.11 e o bloco de programa obtido é visto na Tabela 5.6

Tabela 5.6: Rotina de tratamento da ocorrência de *overflow* e erro de endereçamento em linguagem de máquina

Label	Pseudoinstrução	Comentários
Overflow	Lwi $t0, salv14	Carrega $int salvo em memória
	Lw $t1, $t0, $zero	
	Andi $t1, $0000 0000 0001 1111	Seta $pc dentro de $int = 0001
	Sw $t1, $t0, $zero	Armazena $int na mesma posição
	Lwi $a0, código_*overflow*	Carrega código para ocorrência de overflow e envia para interface RF
	Lwi $a1, dado_RF	
	Sw $a0, $a1, $zero	
	J TX_RF	
	J etapa_final	Finaliza tratamento
Endereçamento	Lwi $t0, salv14	Carrega $int salvo em memória
	Lw $t1, $t0, $zero	
	Andi $t1, $0000 0000 0000 1111	Seta $pc dentro de $int = 0001
	Sw $t1, $t0, $zero	Armazena $int na mesma posição
	Lwi $a0, código_endereçamento	Carrega código para ocorrência de endereçamento e envia para interface RF
	Lwi $a1, dado_RF	
	Sw $a0, $a1, $zero	
	J TX_RF	

Analisando o problema do retorno correto dos registradores bem como o da seqüência correta de operações para a finalização da rotina de tratamento de exceções, verificou-se que o registrador $s3, usado na primeira etapa para salvamento, poderia ser também utilizado. Assim, a flexibilidade seria de certa maneira mantida pela restrição de apenas um registrador, e seria garantido o retorno seguro dos conteúdos dos registradores após a finalização da rotina de tratamento de interrupções.

Para a etapa final, criou-se o bloco de programa da Tabela 5.7.

Tabela 5.7: Bloco de programa em linguagem de máquina para a etapa final do tratamento de exceções

Label	Pseudoinstrução	Comentários
Etapa_Final	Lwi $s3, $salv1	Devolve o conteúdo dos 14 registradores para retorno à execução normal do programa
	Sw $s3, $ra, $zero	
	Addi $s3, $0001	
	Sw $s3, $sp, $zero	
	Addi $s3, $0001	
	Sw $s3, $t0, $zero	
	Addi $s3, $0001	
	Sw $s3, $t1, $zero	
	Addi $s3, $0001	
	Sw $ s3, $t2, $zero	
	Addi $s3, $0001	

Label	Pseudoinstrução	Comentários
	Sw $s3, $a0, $zero	Devolve o conteúdo dos 14 registradores para retorno à execução normal do programa
	Addi $s3, $0001	
	Sw $ s3, $a1, $zero	
	Addi $s3, $0001	
	Sw $ s3, $a2, $zero	
	Addi $s3, $0001	
	Sw $s3, $s0, $zero	
	Addi $s3, $0001	
	Sw $s3, $s1, $zero	
	Addi $s3, $0001	
	Sw $s3, $s2, $zero	
	Addi $s3, $0001	
	Sw $s3, $s3, $zero	
	Addi $s3, $0001	
	Sw $ s3, $int, $zero	
	Addi $s3, $0001	
	Sw $ s3, $pc, $zero	
	Addi $s3, $0001	
	Sw $s3, $int, $zero	
	Lui $s3, FF	Habilitar ocorrência de nova interrupção zerando o bit zero do endereço FFEB usando $int que terá sempre o último bit 0 depois do tratamento de dados
	Addi $s3, EB	
	Sw $int, $s3, $zero	
	Lwi $s3, $int	Recupera o valor de $pc armazenado no $int
	Lw $s3, $s3, $zero	
	Andi $s3, $1111 1111 1111 0000	
	Sft $s3, 4	Desloca o resultado do andi pois $pc só tem 12 bits
	Addi $s3, 1	Coloca $pc na posição anterior +1
	J $s3	Volta para execução normal do programa

Construídos todos os blocos de programa relativos às etapas inicial, de tratamento individual e final, construiu-se então a versão completa do programa que realizará o tratamento de interrupções. São então apresentadas as rotinas de tratamento de exceções provenientes das três interfaces de comunicação (Serial, ΣΔ e RF) bem como o tratamento realizado para a ocorrência de interrupções provenientes da execução de operações pelo processador (*overflow* aritmético e erro de endereçamento).

Nesta etapa foi verificada a necessidade da definição dos códigos de erro de transmissão e recepção a ser utilizado pelas interfaces para informação ao ambiente exterior do tipo de problema ocorrido. A especificação inicial do projeto do SCW não prevê a utilização de códigos de erros, desta maneira foi criado uma seqüência de códigos que poderá ou não ser empregada quando da construção do sistema operacional.

No Anexo III é apresentada a rotina de tratamento de exceções completa.

5.8 O bloco programacional completo

Definidas as rotinas de tratamento de interrupções bem como a aplicação proposta completa, foi possível a montagem do bloco programacional completo, juntamente com as constantes utilizadas, realizado primeiramente em linguagem de máquina. A montagem completa é mostrada no Anexo III.

Desenvolvida a rotina completa em linguagem de máquina para tratamento das exceções para o SCW, pode ser feita a montagem em posições de memória, bem como cálculo correto dos desvios. Para tal deve ser utilizada a linguagem de montagem. O processo de montagem em códigos de máquina e seus resultados são apresentados no capítulo 6.

6 A MONTAGEM DA APLICAÇÃO EM CÓDIGO DE MÁQUINA

Definido o bloco programacional, foi efetuada então a montagem. A ferramenta computacional normalmente responsável pelo processo de montagem em memória dos códigos gerados pelo montador é o carregador, que não se encontra desenvolvido ainda. Posteriormente, este procedimento deverá ser feito automaticamente, quando da montagem de um programa.

Na inexistência de ferramenta que faça ligação às posições de memória, e tendo em vista o fato de que o montador desenvolvido faz apenas a tradução das pseudoinstruções para o nível ainda mais baixo que é o das microinstruções e gera o código correspondente em hexadecimal que montará o bloco em linguagem de máquina a ser executado pelo processador.

Neste caso, procedeu-se então à montagem manual do conjunto programacional. Os resultados obtidos na montagem são mostrados no Anexo IV. Para tal, foram utilizadas as tabelas 2.9 a 2.69 , onde são descritas todas as pseudoinstruções bem como sua representação em linguagem de máquina. Após o final da montagem foi possível a definição de todos os endereços pertinentes ao programa, bem como estabelecimento dos desvios e constantes. Ainda, usando o *software* montador, foi possível a obtenção dos códigos em base hexadecimal representativos da linguagem de máquina, que serão carregados pela interface serial quando do startup do programa. A tabela completa obtida por meio da tradução manual do conjunto de programação completo, contém a pseudoinstrução original, o conjunto de microinstruções descritivo, o código de máquina gerado pelo uso destas microinstruções, a posição em memória na qual os códigos deverão ser montados e o número de ciclos gastos na realização da instrução.

Nesta etapa alguns resultados importantes foram obtidos. Na Tabela 6.1 é apresentada a primeira etapa da aplicação montada.

Tabela 6.1: Montagem da primeira etapa da aplicação

Endereço	Código	Microinstrução	Pseudoinstrução	Ciclos gastos
$000		Lui $s3,MSBInterrupção	Lui $s3,MSBInterrupção	3
$001		J $s3,LSBInterrupção	J $s3,Interrupção	4
$002		Add $t0,$zero,$zero	Lwi $t0, pl_SD	4
$003		Lui $t0, MSBPl_SD		4
$004		Addi $t0,LSBPl_SD		5
$005		Lw $t1,$t0,$zero	Lw $t1, $t0, $zero	5
$006		Lw $t2,$t1,$zero	Lw $t2, $t1, $zero	4
$007		Add $s0,$zero,$zero	Lwi $s0, proc	4
$008		Lui $s0, MSBProc		4
$009		Addi $s0,LSBProc		4
$00A		Sw $t2,$s0,$zero	Sw $t2, $s0, $zero	4
$00B		Add $t2,$zero,$zero	Lwi $t2,	4
$00C		Lui $t2, MSB fim_de_pilha_SD	fim_de_pilha_SD	4
$00D		Addi $t2, LSBfim_de_pilha_SD		4
$00E		Beq $t1,$t2,5	Beq $t1,$t2,5	3
$00F		Addi $t1,1	Addi $t1, 0001h	4
$010		Sw $t1,$t0,$zero	Sw $t1, $t0, $zero	4
$011		Add $a1,$zero,$zero	Lwi $a1, Segue	4
$012		Lui $a1,MSBSegue		4
$013		Addi $a1,LSBSegue		4
$014		J $a1,0	J $a1, 0	3
$015		Add $t1,$zero,$zero	Lwi $t1, $início_de_pilha	4
$016		Lui $t1,parte mais significativa do inicio da pilha		4
$017		Addi $t1, menos significativa do início da pilha		4
$018		Sw $t1,$t0,$zero	Sw $t1, $t0, $zero	4
$019		Add $s1,$zero,$zero	Lwi $s1 $TX_controle	4
$01A		Lui $s1,MSBTX_Controle		4
$01B		Addi $s1,LSB_TXControle		4
$01C		Jal $s1,0	Jal $s1,0	4

Endereço	Código	Microinstrução	Pseudoinstrução	Ciclos gastos
$01D		Add $t0,$zero,$zero	Lwi $t0, const_mul	4
$01E		Lui $t0,MSBConst_Mul		4
$01F		Addi $t0,LSBConst_Mul		4
$020		Lw $a0,$t0,$zero	Lw $a0, $t0, $zero	5
$021		Add $t0,$zero,$zero	Lwi $t0, proc	4
$022		Lui $t0,00h		4
$023		Addi $t0, A7h		4
$024		Lw $a1,$t0,$zero	Lw $a1, $t0, $zero	5
$025		Add $s1,$zero,$zero	Mul $s1, $a0, $a1	4
$026		Add $t2,$zero,$zero		4
$027		Addi $t2,1		4
$028		And $t1,$a0,$t2		4
$029		Beq $1,$zero,1		3
$02A		Add $s1,$s1,$a1		4
$02B		Sft $a0,1		4
$02C		Sft $a1,-1		4
$02D		Beq $a0,$zero,1		3
$02E		J $pc -6		3

Verifica-se na tabela que o programa aumenta consideravelmente seu tamanho, montado em microinstruções. Isto era esperado, dado que todas as pseudoinstruções são constituídas de blocos de instruções mais básicas. Neste exemplo, a montagem de 22 pseudoinstruções resultou em um programa com em 46 microinstruções.

Algumas das instruções utilizadas em pseudolinguagem no programa inicial, tiveram que ter sua sintaxe reformulada em termos de registradores. As pseudoinstruções em geral trabalham em seu conteúdo com registradores temporários, de forma a possibilitar a realização da tarefa requerida. Neste caso, tais registradores não poderiam ser usados como campos da pseudoinstrução, sob a pena de não se conseguir efetuar corretamente as operações. Este é o exemplo do bloco de programa presente na Tabela 6.2, onde a operação de multiplicação havia sido feita primeiramente com relação aos registradores $s1, $t2 e $t1. Durante a montagem verificou-se que a operação de multiplicação trabalhava internamente com os registradores temporários. Foi necessária então a troca dos registradores, tanto dentro da operação quanto nas etapas anteriores de carregamento dos dados, pelos registradores salvos ou de argumento. O bloco resultante é visto na Tabela 6.3.

Tabela 6.2: Primeira versão da multiplicação em linguagem de máquina

Label	Pseudoinstrução	Comentário
	Lwi $t0, const_mul	Carrega o dado e realiza a multiplicação por uma constante
	Lw $t1, $t0, $zero	
	Lwi $t0, proc	
	Lw $t2, $t0, $zero	
	Mul $s1, $t1, $t2	

Tabela 6.3: Sintaxe da operação de multiplicação modificada na etapa de montagem

Label	Pseudoinstrução	Comentário
	Lwi $t0, const_mul	Carrega o dado e realiza a multiplicação por uma constante
	Lw $a0, $t0, $zero	
	Lwi $t0, proc	
	Lw $a1, $t0, $zero	
	Mul $s1, $a0, $a1	

Outro aspecto importante é que todos desvios tiveram que ser recalculados. Este trabalho deveria ser feito por um *software* adequado, que misturasse a função de tradução do montador com a função de ligar todas as posições de memória, bem como calcular os *offsets*. Como este software ainda não foi desenvolvido, também nesta etapa foram recalculados os desvios utilizados nas versões em pseudoinstruções, de forma a

fazer com que o programa pudesse realizar os saltos para as instruções desejadas. Um exemplo é visto nas tabelas 6.4 e 6.5. Neste caso, à instrução original de desvio incondicional, que realiza saltos para até 7 palavras, teve ter ser associada uma instrução de desvio incondicional que realiza desvios maiores, de até 8 bits.

Tabela 6.4: Primeira versão da etapa de identificação da exceção ocorrida

Label	Pseudoinstrução	Comentário
	Andi $t0,$int,$000000000000 1000_b	Testes sucessivos com o conteúdo de $int para identificar que tipo de interrupção aconteceu e transferir controle para a rotina de tratamento adequada
	Bgt $t0,$zero, 9	
	Andi $t0,$int, 0000000000000100_b	
	Bgt $t0,$zero, 8	
	Andi $t0,$int,$0000 0000 0000 0 0 01_b	
	Bgt $t0, $zero, 7	
	Andi $t0, $int, $0000 0000 0000 0 0 10_b	
	Bgt $t0, $zero, 6	
	Andi $t0, $int, $0000000000001 0 11_b	
	Bgt $t0, $zero, 5	
	Jd IntEndereçamento	Desvios para rotinas de tratamento individuais
	Jd zero,IntOverflow	
	Jd zero,IntRF	
	Jd zero,IntSerial	

Tabela 6.5: Sintaxe modificada para atender ao código resultante da montagem

Label	Microinstrução	Pseudoinstrução
$0F1	Add $t1,$zero,$zero	Andi $t0,$int,$000000000000 1000
$0F2	Addi $t1,8	
$0F3	Addi $s1,$s1,$t1	
$0F4	Bgt $t0,$zero,	Bgt $t0, $zero, 9(Endereçamento)
$0F5	Lui $t1,01	
$0F6	J $zero,0B	
$0F7	Lui $s1,$01	
$0F8	J $s1,07	
$0F9	Add $t1,$zero,$zero	Andi $t0,$int, 0000000000000100_b
$0FA	Addi $t1,4	
$0FB	Addi $s1,$s1,$t1	
$0FC	Bgt $t0,$zero,2	Bgt $t0, $zero, 8(Overflow)
$0FD	Lui $t1,01	
$0FE	J $zero,	
$0FF	Lui $s1,$01	
$100	J $s1,0A	
$101	Add $t1,$zero,$zero	Andi $t0,$int,$0000 0000 0000 0 0 01_b
$102	Addi $t1,1	
$103	Addi $s1,$s1,$t1	
$104	Bgt $t0, $zero, 7(RF)	Bgt $t0, $zero, 7(RF)
$105	Add $t1,$zero,$zero	And $t0, $int, $0000 0000 0000 0 0 10_b
$106	Addi $t1,2	
$107	Addi $s1,$s1,$t1	
$108	Bgt $t0, $zero, 6(Serial)	Bgt $t0, $zero, 6(Serial)
$109	Add $t1,$zero,$zero	And $t0, $int, $0000000000001 0 11_b
$10A	Addi $t1,11	
$10B	Addi $s1,$s1,$t1	
$10C	Bgt $t0, $zero, 5 (SD)	Bgt $t0, $zero, 5 (SD)
$10D	Add $s1,$zero,$zero	Jd IntEndereçamento
$10E	Lui $s1,MSBIntEndere	
$10F	J $s1, LSBIntEndereço	
$110	Add $s1,$zero,$zero	Jd zero,IntOverflow
$111	Lui $s1,MSBIntEndere	
$112	J $s1, LSBIntEndereço	
$113	Add $s1,$zero,$zero	Jd zero,IntRF
$114	Lui $s1,MSBIntEndere	
$115	J $s1, LSBIntEndereço	
$116	Add $s1,$zero,$zero	Jd zero,IntSerial
$117	Lui $s1,MSBIntEndere	
$118	J $s1, LSBIntEndereço	

Realizada a montagem completa, a etapa seguinte foi de cálculo do número de ciclos gastos em cada etapa de execução. Para tal utilizou-se a Tabela 2.4 que contém o número de ciclos necessários à execução de cada uma das microinstruções.

Com estas informações, pôde-se fazer uma análise sucinta do desempenho do programa em termos de velocidade de execução. Dado que a rotina de tratamento de exceções trabalha paralelamente à aplicação, sendo requisitada e executada a cada interrupção ocorrida, estabeleceu-se como parâmetro para a análise, que a aplicação transcorra sem a ocorrência de nenhuma interrupção, simulando a operação completa de processamento. Tanto na rotina de tratamento de exceções quanto na aplicação básica são utilizados testes e desvios, que podem alterar o número de ciclos gastos conforme sua execução. Como a análise aqui realizada objetiva a estimativa do tempo de execução do programa de forma ilustrativa, optou-se por se trabalhar com a situação hipotética de que todas as instruções do bloco sejam realizadas, sem levar em conta ocasionais desvios. Desta maneira, avalia-se o pior caso de processamento, e o tempo de execução real da aplicação, caso algum dia seja utilizada num protótipo, deve ser menor do que o aqui obtido.

Computando todos os ciclos gastos na aplicação, chegou-se ao número de 700 ciclos para a aplicação, correspondendo ao tempo de execução de 2.8 μs. Dado que a taxa de transmissão para a interface de saída de RF ainda não encontra-se definida, a análise deve ser feita com relação a interface Serial. Considerando a taxa padrão [4], da interface serial, chega-se ao valor de $1{,}25 \times 10^{-4}$ s, para o envio de uma palavra de 16 bits. Assim sendo, comprova-se a necessidade da utilização de uma pilha para armazenamento de dados disponibilizados pelas interfaces. Em outras versões da aplicação, pode-se trabalhar no sentido de diminuir o programa, e conseqüentemente o tempo de execução.

Este estudo deve ser refeito quando da definição completa das interfaces de comunicação, de forma a dimensionar a aplicação e as pilhas de armazenamento para melhor atender aos requisitos do sistema.

7 OS TESTES REALIZADOS COM O MONTADOR

Com o propósito de testar toda a potencialidade do montador desenvolvido [8], assim como verificar os erros ainda não detectados, elaborou-se uma série de vetores de teste. A validação do *software* montador só pode ser realizada com testes extensivos de sintaxe e utilização de todos os comandos por ele suportados. Neste sentido, foram criados os vetores de teste que constituem-se, basicamente, de testes de conteúdo e de formato das instruções. Visando a simplificação, foram elaborados neste trabalho vetores de testes específicos para as 16 micro-intrsuções e a partir dos exames feitos para as mesmas e, foram criados os testes para as outras 48 pseudoinstruções. Assim foi possível a validação do *set* completo de instruções por meio da reutilização dos resultados obtidos para as microinstruções nos testes relacionados às pseudoinstruções.

7.1 Testes de microinstruções

Para cada uma das 16 microinstruções foi realizado um primeiro conjunto de verificações referentes à utilização dos registradores. Para as instruções do tipo R, que trabalham com três registradores (RegOrigem, RegOrigem2, RegDestino), os testes foram ser realizados com todos os registradores ocupando cada um dos três campos. Este teste visa o reconhecimento correto do registrador, bem como, a verificação da universalidade da instrução, com relação à utilização com quaisquer dos registradores.

No caso das instruções do tipo I e do tipo J, que utilizam 2 e 1 registrador, o teste foi feito da mesma maneira, com cada um dos 16 registradores ocupando os campos de registrador de origem e, assim, verificou-se como o *software* responderia a estas situações. O esperado era que a montagem fosse realizada sem maiores problemas, neste caso.

Outro teste realizado dentro dos vetores de teste elaborados neste trabalho, ainda, no sentido do tratamento feito pelo *software* montador para cada tipo de instrução é o do teste com todos os formatos. O objetivo neste caso é mais uma vez testar como o montador responde a um erro proposital, ou seja, uma troca do formato das instruções. Exemplificando, para uma instrução do tipo R, como a instrução Sub, os testes deviam acontecer usando a instrução como se fosse uma instrução do tipo J e, como se fosse uma instrução do tipo I. Cada um dos dois formatos mencionados usa um número diferente de registradores da instrução do tipo R, e era necessário saber como o programa se comporta nas duas situações.

Para as instruções de desvio, foram também realizados testes para valores de *offset* maiores do que os suportados pela instrução; ou no caso de desvios incondicionais, endereços inválidos de memória, que deveriam ocasionar num erro de compilação do montador.

Sabe-se que as instruções e os registradores são mapeados conforme seu código binário. Desta maneira, a conversão das instruções em código de máquina é imediata, e é obtida pela Tabela 7.1.

Tabela 7.1: Códigos de Registradores e Microinstruções

Código	Registrador	Código	Microinstrução
0000	$zero	0010	Add
0001	$t0	0011	Sub
0010	$t1	1000	Addi
0011	$t2	1001	Shift
0100	$a0	0100	And

Código	Registrador	Código	Microinstrução
0101	$a1	0101	Or
0110	$a2	1010	Not
0111	$s0	0110	Xor
1000	$s1	0000	Lw
1001	$s2	0001	Sw
1010	$s3	1011	Lui
1011	$int	0111	Slt
1100	$gp	1100	Beq
1101	$sp	1101	Blt
1110	$pc	1110	J
1111	$ra	1111	Jal

Definido a estrutura do conjunto de vetores de teste, pôde ser elaborado um *set* de pequenos programas destinados ao teste de todas as 16 microinstruções. Os resultados mostrados nas seções de 7.1.1 a 7.1.10 e nas tabelas presentes no Anexo I,por meio de tabelas, que permitem observar o teste realizado através da instrução presente na primeira coluna, o resultado obtido em linguagem de montagem pelo montador e o código numérico do erro originado, caso ocorra algum.

7.1.1 Testes das Instruções Add e Sub

No caso da instrução Add, foram realizados os testes de universalidade de registradores. Os testes de formato foram realizados com os formatos das instruções do tipo I e J e, ainda, com a utilização de constantes como campos referenciados para instrução. O teste realizado para a instrução Sub foi idêntico ao da instrução Add, por possuírem ambos o mesmo formato de instrução e trabalharem similarmente. Foram obtidos os resultados das tabelas apresentadas nas Tabelas I.1 e I.2.

7.1.2 Testes da Instrução Addi

A instrução Addi é do tipo J, ou seja, possui um registrador (RegOrigem) e uma constante de 8 bits. Como existe apenas um registrador, basta testar o funcionamento desta instrução, com cada um dos 15 registradores no campo RegOrigem, para certificar a universalidade da instrução no que concerne ao uso dos registradores. Quanto aos testes de formato, alguns outros foram incluídos. Por se tratar de uma instrução que trabalha com uma constante de 8 bits, alguns testes foram realizados com relação à base desta constante. Utilizou-se, por exemplo, a base binária, a base decimal e a base hexadecimal, verificando o resultado obtido. Foi necessária a inclusão de outro teste, com uma constante que vá além dos 8 bits reservados. Os resultados encontram-se na tabela transcrita na Tabela I.3.

Verificou-se que para a instrução Addi, assim como para todas as outras, a constante deve ser colocada logo após a vírgula. A adição de espaço em branco entre as duas leva ao erro #07 – Valor de *offset* utilizado inválido ou com espaço entre a vírgula e o número. Esta regra deve ser seguida, invariavelmente, para todas as instruções inseridas dentro de programas do montador. Verificou-se através da realização dos testes com a constante $\#20_h$ que os valores devem ser inseridos na base decimal e o *software* faz a conversão para o valor em hexadecimal. Com a utilização de números na base hexadecimal, verificou-se a ocorrência de um fenômeno interessante. As letras são interpretadas como zeros e os números traduzidos normalmente, se vierem antes de letras. Isto significa que o montador neste caso preenche o conteúdo da constante com zeros assim que uma letra é encontrada. Números em base binária são interpretados como decimais.

Verificou-se que os erros mostrados ao usuário muitas vezes não condizem com o contexto de operação. Em se tratando da instrução Addi, os erros não deveriam utilizar a palavra *offset*, pois a mesma se aplica em casos de instruções indicativas de desvio condicional, que claramente não é o caso da instrução Addi.

7.1.3 Teste da Instrução Shift

Os testes realizados para a instrução Shift foram similares aos realizados para a instrução Addi. Novamente foram feitos testes com os registradores a serem utilizados pela instrução, e aqui também, testes com relação ao formato e à base utilizada para as constantes foram necessários.

A instrução Shift por realizar deslocamentos, tem que ter seu valor de constante menor ou igual a 16, pois uma palavra pode ser deslocada de no máximo 16 posições, considerando os 4 bits disponíveis para a constante. O deslocamento deve ser digitado no formato decimal.

Por meio dos testes de formato, verificou-se mais uma vez que a sintaxe usada na redação das mensagens de erro mostradas ao usuário não condiz com a instrução usada, uma vez que na instrução de deslocamento Shift não é usado um *offset*, e sim, uma constante que armazena a quantidade de deslocamentos que deve ser realizada.

7.1.4 Teste das Instruções And, Or e Xor

Os testes realizados com as instruções lógicas And, Or e Xor são bem similares aos realizados com as instruções Add e Sub, pelo fato de também serem instruções que trabalham com 3 registradores em seu conteúdo. Os resultados são mostrados nas Tabelas I.5, I.6 e I.7, mostradas no Anexo I.

As instruções apresentaram o comportamento desejado e foram corretamente traduzidas para os códigos de máquina. Para esta verificação foi utilizada a Tabela 7.1 comparando os resultados obtidos e os esperados.

7.1.5 Teste da Instrução Not

A instrução Not é a única microinstrução lógica que trabalha com o conteúdo de apenas um registrador. Por isso, seus vetores de teste são diferentes, em sua concepção, dos vetores utilizados para as demais instruções lógicas. Os vetores e resultados são mostrados na Tabela I.8 no Anexo I.

Neste caso, verifica-se que a instrução Not $s1,10 gerou um resultado diferente do esperado, que seria um erro, já que a instrução Not funciona apenas com um registrador.

Ainda, a instrução Not 1000 gerou o resultado A000 que representaria, neste caso, a utilização da instrução Not com o registrador $zero que é representado pelo código #0000 em binário. Verifica-se, então, a constituição de um erro do montador, tendo em vista que o único formato que deveria ser aceito pela instrução Not seria a sua utilização com um registrador, e não com constantes conforme se observou na última instrução dos testes de formato realizados.

7.1.6 Teste das instruções Lw e Sw

As instruções Lw e Sw foram testadas no âmbito das duas classes de testes desenvolvidas neste trabalho: testes de funcionamento dos registradores e testes de mudanças no formato. Os resultados obtidos são mostrados no Anexo I.

Verificou-se o resultado desejado comprovando assim, a eficácia do *software* na

tradução das microinstruções de escrita e leitura em memória em códigos de máquina hexadecimais. As tabelas obtidas encontram-se no Anexo I.

7.1.7 Teste da instrução Lui

Em se tratando de uma instrução que realiza o carregamento de uma constante de 8 bits nos 8 bits mais significativos de um registrador qualquer, foram realizados novamente testes extensivos com relação ao formato da constante a ser utilizada. Os testes de verificação da universalidade, bem como da utilização dos formatos inadequados também foram realizados. Os resultados estão presentes no Anexo I.

Foram observados alguns problemas nesta etapa de testes, com relação à base das constantes que quando se tenta utilizar um número fora do *range* de valores aceitos (-128 e 255). O montador acusa o erro, mas compila o programa normalmente preenchendo os códigos relativos às posições de ocorrência dos erros com o valor $\#FFFF_h$. O erro, como no caso das instruções testadas anteriormente, não deveria gerar nenhum resultado de compilação, que pode ser confundido com um resultado gerado pela compilação de uma instrução de sintaxe correta.

Foi observada a mesma peculiaridade encontrada nos testes da instrução Addi quando da utilização de caracteres alfa-numéricos nas constantes a serem carregadas. Os caracteres alfa-numéricos, como os presentes na base hexadecimal, geram o código de máquina #0 para qualquer um deles. Dado que nas especificações para a realização do *software* montador previu-se a base decimal para as constantes, a tradução de caracteres alfa-numéricos por zeros não é de todo incorreta. No entanto, para estes casos deveria ser mostrado algum tipo de dado ao usuário que permita informá-lo a respeito da utilização de constantes em bases diferentes da decimal.

7.1.8 Teste de Instrução Slt

Os testes realizados para a instrução Slt foram idênticos aos realizados para as demais instruções do tipo R, que utilizam quatro campos em sua composição: o de identificação da própria instrução e os três registradores, de destino, de origem e de origem 2 respectivamente. A Tabela I.12 utilizada para teste é encontrada no Anexo I.

Os resultados obtidos foram satisfatórios já que no caso desta instrução, a utilização dos formatos e sintaxe inadequada é indicada pelo montador como o erro #06 – Registrador inexistente no *set* do processador, e após a ocorrência do erro, a compilação não pode ser realizada até que o problema seja corrigido.

7.1.9 Teste das Instruções Beq e Blt

As duas microinstruções de desvio condicional, presentes no *set* de microinstruções desenvolvido para o SCW utilizam três campos que são o RegOrigem, o RegOrigem2 e o *offset*, também chamado de desvio que é a quantidade à qual o registrado $pc (contador de programa) será somado, de forma a possibilitar o desvio efetivo.

Assim foram efetuados testes de universalidade para os campos RegOrigem e RegOrigem2. Os testes de unicidade e reconhecimento de formato também foram realizados. Ainda, devido à utilização da constante de 4 bits representativa do desvio, foram realizados testes de valores para esta constante, nas três bases possíveis.

Os vetores bem como os resultados são mostrados nas Tabelas I.13 e I.14 presente no Anexo I.

Para a utilização de constantes maiores que o valor 7 em decimal, que é o maior valor de *offset* aceitável, o montador acusa o erro #12 – Valor de *offset* maior que 7 ou negativo. Sabe-se que a constante de desvio não pode assumir valores maiores que os representáveis por quatro bits. No entanto, como a especificação da linguagem de montagem prevê a utilização do sistema em complemento à dois, os desvios só podem ser feitos com valores entre –7 e 7. Porém o montador não aceita números de desvio negativos, em complemento à 2, gerando, para estas situações, o erro 12 descrito e parando assim a compilação.

Outra característica importante observada foi a utilização de espaços entre a declaração dos registradores. Este procedimento gera o erro #11 – Registrador não existente no *set* do processador, quando na verdade se deseja sim utilizar um registrador existente (tal como $s1 ou $s2), mas o montador interpreta o espaço como o emprego de um registrador com nome começando por espaço e não por $ como seria o padrão.

7.1.10 Teste das Instruções de Desvio Incondicional

Para as instruções de desvio condicional e incondicional, temos o formato do tipo J, ou seja, trabalha-se com o campo referente à instrução; outro referente a um registrador de origem (RegOrigem) e, outro campo de no máximo 8 bits destinado ao endereçamento. Neste caso, são necessários apenas testes com relação à universalidade dos registradores e à unicidade de reconhecimento do formato da instrução. Também, são necessários testes com relação aos bits de endereçamento. Os testes foram realizados para a instrução J e Jal e os resultados são mostrados nas Tabelas I.15 e I.16. Sabe-se que a instrução comporta constantes de até 8 bits, e que o desvio pode ser feito tanto para cima quanto para baixo em um programa. Assim o *range* que a constante de 8 bits pode assumir é –128 até 127, em decimal.

Algumas conclusões puderam ser obtidas com os testes para as instruções J e Jal. Para a instrução J foi gerado código de compilação normal para a utilização de letras, ao invés de números, nos campos destinados ao deslocamento. E este procedimento, tal como observado em outras microinstruções, gerou zeros no código de máquina ao invés de erros de compilação. A base binária também não é aceita para o deslocamento, pois os números em binário são interpretados pelo montador como números decimais, em geral maiores do que os aceitos pelo desvio.

Ainda nos testes de formatos realizados, verificou-se que a ocorrência de erros de sintaxe não parava a compilação, e ao final gerava códigos indevidos dentro do código de máquina ($\#FFFF_h$). Os erros gerados deveriam, como observado em testes com outras instruções, ocasionar a parada do processo de compilação, que só poderia ser reiniciado quando da resolução do problema.

No processo de tradução, o montador busca o final da instrução através do reconhecimento da constante. Caso seja inserido algum outro campo após a mesma, sua presença é ignorada pelo montador, que compila normalmente o programa.

7.2 Testes Realizados com as Pseudo-Instruções

Para efetuar os testes de tradução do montador para as pseudoinstruções, deveria ser utilizada a mesma metodologia de testes adotada para o teste das microinstruções. Tendo em vista que existem no *set* do SCW 48 pseudo-instruções que precisam ser convertidas para em média 4 microinstruções, o processo de testes extensivos, utilizado para as microinstruções se tornaria moroso e ineficiente. Assim sendo, foi estudada uma nova estratégia para realização dos testes das pseudoinstruções.

Tendo sido realizados anteriormente testes de formato e de universalidade no uso dos registradores para todas as 16 microinstruções, poderia ser reutilizado o resultado obtido para o teste do conjunto de pseudoinstruções.

Porém algumas pseudoinstruções se utilizam de registradores específicos dentro de sua implementação de forma a realizar a atividade requerida. Desta forma, na sintaxe destas pseudoinstruções deveria ser vedado o uso destes registradores, que são usados internamente pela pseudoinstrução. Desta maneira, uma primeira etapa de testes deveria, para as instruções que possuem o uso de registradores internos, testar como o montador responde à inclusão destes mesmos registradores.

Ainda, para as pseudoinstruções a tradução inclui a seqüência específica de microinstruções relativa. O código de máquina gerado deve então fazer a montagem da pseudoinstrução utilizando o código relativo às respectivas microinstruções. Foi necessário então o teste de todas as pseudoinstruções neste sentido, de forma a assegurar que a tradução seja feita da maneira correta. Os resultados obtidos nos testes encontram-se nas Tabelas de II.1 até II.43, mostradas no Anexo II.

Testando a instrução Mul, verificou-se que o *software* respondia da maneira correta, não permitindo a utilização dos registradores $t1 e $t2, usados internamente, na sintaxe da instrução. No entanto, já para o teste da instrução Div, verificou-se que a compilação acontecia sem erros utilizando na declaração da instrução o registrador $t1, que é usado internamente, constituindo assim uma falha do programa. O mesmo problema foi verificado nos testes das instruções Subi, Slti, Sgti, Seqi, Chg, Jalpcd, Jalrd e Bnei.

No caso da instrução Divi também foram observadas falhas. A compilação não pode ser realizada no caso da utilização da sintaxe Divi $s2,$s1,100, que deveria ser possível. O montador apresentou o erro #11 – Registrador inexistente no *set* do processador. Foram utilizados outros registradores, em sua sintaxe, e o erro retornado foi o mesmo. O mesmo ocorreu para as instruções Jr e Jpc.

Realizando o teste com a instrução Rem, que foi constituído apenas do teste do resultado obtido tendo em vista que esta instrução não utiliza internamente nenhum registrador além dos especificados pela sintaxe. Neste teste, verificou-se que o resultado obtido para a tradução da constante não representava o valor negativo da constante explicitada, como deveria ser.

Na etapa de testes das instruções Sftl e Sftr, por serem instruções de desvio deveriam ser utilizadas constantes de valor não superior a 16. Foram feitos testes então com constantes maiores que este valor, resultando em erro conforme esperado. No entanto, analisando o teste das duas instruções com a sintaxe correta verificou-se que o resultado gerado para ambas foi igual, sendo que na instrução Sftl, o valor da constante deveria ser negativo, indicando a rotação à esquerda.

Nos testes efetuados para a instrução Lb, verificou-se que o montador acusava erro interno do programa, mostrando a janela de erro do Windows “Este programa executou uma operação ilegal e será fechado”. Não foi possível realizar assim o teste da instrução Lb com a sintaxe correta. Um outro erro de programa (*Invalid pointer operation*) foi também observado na execução das instruções Sle, Sge, Sb, Jalr e Jald não podendo ser realizada a compilação nestes casos.

Quando da realização dos testes com a pseudoinstrução Chg verificou-se que o resultado gerado para a mesma era igual ao observado para a instrução Mov. Partindo do princípio da unicidade de microinstruções, isto não deveria acontecer, constituindo

assim mais uma falha a ser corrigida no *software*.

Ainda, nos testes realizados com a pseudoinstrução Jalrd, o código de compilação mostrado indica a utilização da instrução Jald, como se pode observar na Tabela II.42, Anexo II. No entanto, o código gerado para a compilação da instrução Jald com a mesma sintaxe, gera códigos de máquina diferentes, indicando um erro simples no momento de mostrar o resultado da compilação, sendo mostrada a palavra Jald ao invés da Jalrd.

As instruções Muli, Rem, Comp, Andi, Xori, Ori, Ld, Sd, Mflo, Mfhi, Move, Mov, Sne, Seq, Sgt, Bne, Bgti, Blti,Bgt, Beqi, Sgti, Jd, Jalpc e Jpc foram corretamente compiladas pelo *software* montador, sem falhas.

Todas as falhas encontradas nos testes com o *software* montador realizados neste trabalho foram reportadas ao programador [8], para correção. Foi então gerado o *software* final validado.

8 DISCUSSÃO

Apesar de não existir ainda meio físico para validação dos *softwares* que venham a ser desenvolvidos para o processador do SCW, a aplicação e a rotina de tratamento de exceções discutidas neste trabalho atendem de maneira completa aos objetivos primeiros da elaboração das mesmas: (1) confirmação das especificações de software, tais como a universalidade do *set* de instruções desenvolvido; (2) utilização das definições feitas para o hardware, como forma de associá-lo ao funcionamento do *software*, observando seu comportamento.

Desta maneira, foram obtidos os seguintes resultados:

- Uma metodologia de desenvolvimento em linguagem de máquina para o SCW foi criada e validada.

- Foram definidas todas as atividades relacionadas ao fluxo de dados entre o processador e as interfaces, atendendo às especificações básicas anteriormente criadas.

- Foi construída uma aplicação para o sistema, que captura dados provenientes das interfaces de entrada, processa-os e os envia à interface de saída, cobrindo assim algumas das tarefas básicas envolvidas em qualquer *software* que venha a ser elaborado para trabalhar com o SCW.

- Implementou-se uma rotina de tratamento de exceções, de maneira a controlar toda a sinalização feita entre as interfaces e o processador. Desta maneira, foi possível o estabelecimento completo dos parâmetros especificados para aquisição de dados por parte do processador e para a transmissão efetiva de dados por meio da interface de saída de RF, obedecendo às especificações de projeto anteriormente designadas.

- Trabalhou-se com a montagem dos programas desenvolvidos em memória, possibilitando a análise do comportamento em termos de tempo de execução do *software* criado, bem como o detalhamento do processo de tradução realizado pelo montador, que transforma um programa escrito em linguagem de máquina num programa em linguagem de montagem, interpretável pelo processador.

- Foram realizados testes extensivos com o *software* montador, criado especialmente para a linguagem desenvolvida para o SCW. Nesta etapa foram obtidos resultados com relação a um detalhamento maior do comportamento do montador, e da forma como a tradução a ele designada foi realizada. Foram ainda identificadas falhas, que foram posteriormente corrigidas e melhorias possíveis.

Tendo em vista as limitações em termos de projetos não finalizados de algumas das interfaces, e de não se poder neste primeiro momento testar em chip o funcionamento da aplicação aqui discutida, os resultados se mantiveram dentro das expectativas iniciais do projeto.

9 CONCLUSÃO

Os objetivos estabelecidos para o desenvolvimento de uma aplicação para o sistema SCW, que eram: (1) o teste da funcionalidade e da universalidade da linguagem de máquina desenvolvida para o sistema; (2) verificação da consistência das especificações criadas pela equipe de *hardware* para a comunicação com as interfaces; (3) validar e confirmar o projeto *hardware*/*software* foram alcançados.

Para tal, foi desenvolvida uma metodologia para programação na linguagem de máquina criada para o SCW. Foi necessário um estudo extensivo das especificações de projeto, de forma a tornar a aplicação desenvolvida o mais próximo possível de uma aplicação que venha a ser utilizada em escala, quando da fabricação do chip.

O desenvolvimento da aplicação em conjunto com a rotina para tratamento das exceções possibilitou:

- Trabalhar com todas as interfaces de comunicação especificadas bem como detalhar o controle do fluxo de dados entre as mesmas e o processador, realizado pelo *software*;
- Utilizar o maior número possível de instruções pertinentes ao *set*, sem aumentar em demasia a complexidade e o tamanho do programa que na versão final ocupou aproximadamente 2KB;
- Efetuar operações de processamento básicas.

Foram ainda realizados testes de validação com o *software* montador desenvolvido em outro projeto final. Os objetivos destes testes que eram a observação de seu comportamento face às mais diversas situações de operação, bem como identificação de falhas que pudessem ocorrer e estabelecimento de soluções adequadas à estas situações foram atingidos.

O ponto forte deste trabalho foi criar, a partir do *set* de instruções, uma aplicação, incluindo a rotina de tratamento de exceções, que realiza todas as tarefas básicas necessárias à operação do processador e das interfaces de comunicação. Ainda, foi possível estabelecer parâmetros que possam ser oportunamente utilizados por programadores que venham a desenvolver aplicações mais específicas para o sistema, onde o processamento realizado sobre os dados recebidos deve ser mais complexo. Desta maneira, criando um subsídio para realizações posteriores, onde algumas das operações básicas de controle de fluxo de dados e tratamento de exceções possam ser utilizadas. A flexibilidade ainda é permitida, tendo em vista que para desenvolver novas aplicações, a estrutura discutida neste trabalho não necessita de modificações drásticas.

A limitação mais séria deste trabalho foi a realização da aplicação sem que se possa testar seu funcionamento em meio físico, pois o chip encontra-se ainda em fase de projeto, não existindo assim uma primeira versão fabricada em que se pudesse rodar a aplicação e observar seu comportamento. No entanto, este projeto pode ser continuado futuramente, quando se passar à etapa de fabricação e estiver disponível o *hardware* físico para testes.

Ainda outras limitações foram encontradas no sentido de que algumas das especificações com relação às interfaces não se encontram ainda completas nesta etapa. Foi necessário então o estabelecimento de parâmetros não condizentes com o projeto para o atendimento dos objetivos da aplicação no que diz respeito ao uso das interfaces

de comunicação.

Assim, no andamento deste estágio supervisionado, trabalhou-se em conjunto com a equipe de *hardware* e *software* no desenvolvimento da aplicação e do *software* montador. Foram desenvolvidos mecanismos para tratamento das exceções suportadas pelas especificações de *hardware* para o SCW, bem como o detalhamento dos processos de transmissão e recepção de dados por parte do sistema. Com os testes realizados no *software* montador, foi possível a identificação de falhas que foram oportunamente corrigidas, e a validação de seu funcionamento.

Este trabalho, ao mesmo tempo em que encontrou respostas para muitas perguntas, solucionando as dificuldades iniciais do projeto, criou novas perguntas e oportunidades de estudo e pesquisa. O projeto precisará ser amplamente testado, quando o *hardware* estiver pronto. Estes testes certamente mostrarão oportunidades de correções, melhorias, e desenvolvimento.

REFERÊNCIAS

[1] BAGGA, S. “Interface sigma-delta”, Dissertação de Mestrado em Engenharia Elétrica, UnB, a ser defendida em Out. 2002.

[2] BECK, L. L. “System Software: An Introduction to Systems Programming”. 2° Edição. Addison-Wesley Ed. , Boston, USA, 1990

[3] BENEVENUTO, A. “Transceptor de Rádio Freqüência para Sistema em CHIP”, Dissertação de Mestrado em Engenharia Elétrica, UnB, a ser defendida em Dez. 2002.

[4] BENÍCIO JR, G. M. “Projeto de Microprocessador RISC 16-bit para Sistema de Comunicação sem Fio em CHIP”, Dissertação de Mestrado em Engenharia Elétrica, UnB, a ser defendida em Out. 2002.

[5] COSTA, J. D. “Desenvolvimento da ULA para o processador em Sistema de Comunicação sem Fio”. Projeto Final de Graduação em Engenharia Elétrica, UnB, Jun. 2002.

[6] LINDER, R.R. “Linguagem de Máquina para Processador num Sistema em CHIP (SOC)”. Dissertação de Mestrado em Engenharia Elétrica, a ser defendida em Out. 2002.

[7] PATTERSON, D. A. & HENNESSY, J. L. "Organização e Projeto de Computadores – A interface *Hardware* e *Software*". Editora Campus, Rio de Janeiro, Brasil, 2000.

[8] RANGEL, R. M. D. “Montador”, Projeto Final de Graduação em Engenharia Elétrica, UnB, a ser apresentado em Set. 2002.

[9] TAUB, H. “Circuitos digitais e microprocessadores”. McGraw-Hill Editora, São Paulo, Brasil, 1984.

[10] TOCCI, R. J. & LASKOWSKI, L. P. “Microprocessadores e microcomputadores, Hardware e Software”. 3º edição. Prentice-Hall do Brasil, 1987.

ANEXO I - TESTES DAS MICROINSTRUÇÕES

Tabela 0.1 – Teste das instrução Add

Teste da Instrução Add			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Add $s1,$s2,$s0	2897	2897	
Add $s1,$s2,$s1	2898	2898	
Add $s1,$s2,$s2	2899	2899	
Add $s1,$s2,$s3	289A	289A	
Add $s1,$s2,$t0	2891	2891	
Add $s1,$s2,$t1	2892	2892	
Add $s1,$s2,$t2	2893	2893	
Add $s1,$s2,$a0	2894	2894	
Add $s1,$s2,$a1	2895	2895	
Add $s1,$s2,$a2	2896	2896	
Add $s1,$s2,$gp	289C	289C	
Add $s1,$s2,$ra	289F	289F	
Add $s1,$s2,$pc	289E	289E	
Add $s1,$s2,$sp	289D	289D	
Bloco 2 – Variação no campo RegOrigem			
Add $s2,$s0,$s1	2978	2978	
Add $s2,$s1,$s1	2988	2988	
Add $s2,$s2,$s1	2998	2998	
Add $s2,$s3,$s1	29A8	29A8	
Add $s2,$t0,$s1	2918	2918	
Add $s2,$t1,$s1	2928	2928	
Add $s2,$t2,$s1	2938	2938	
Add $s2,$a0,$s1	2948	2948	
Add $s2,$a1,$s1	2958	2958	
Add $s2,$a2,$s1	2968	2968	
Add $s2,$gp,$s1	29C8	29C8	
Add $s2,$ra,$s1	29F8	29F8	
Add $s2,$pc,$s1	29E8	29E8	
Add $s2,$sp,$s1	29D8	29D8	
Bloco 3 – Variação no campo RegDestino			
Add $s0,$s1,$s2	2789	2789	
Add $s1,$s1,$s2	2889	2889	
Add $s2,$s1,$s2	2989	2989	
Add $s3,$s1,$s2	2A89	2A89	
Add $t0,$s1,$s2	2189	2189	
Add $t1,$s1,$s2	2289	2289	
Add $t2,$s1,$s2	2389	2389	
Add $a0,$s1,$s2	2489	2489	
Add $a1,$s1,$s2	2589	2589	
Add $a2,$s1,$s2	2689	2689	
Add $gp,$s1,$s2	2C89	2C89	
Add $ra,$s1,$s2	2F89	2F89	
Add $pc,$s1,$s2	2E89	2E89	
Add $sp,$s1,$s2	2D89	2D89	
Teste dos formatos			
Add $s1,1000	Erro	Erro	#06 – Registrador inexistente no set do processador
Add $s1,$s2,1000	Erro	Erro	#06 – Registrador inexistente no set do processador
Add 1000,$s1,$s2	Erro	Erro	#06 – Registrador inexistente no set do processador
Add $s1,$s2,1000	Erro	Erro	#06 – Registrador inexistente no set do processador

Tabela I.2 – Teste das instrução Sub

Teste da Instrução Sub			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Sub $s1,$s2,$s0	3897	3897	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Sub $s1,$s2,$s1	3898	3898	
Sub $s1,$s2,$s2	3899	3899	
Sub $s1,$s2,$s3	389A	389A	
Sub $s1,$s2,$t0	3891	3891	
Sub $s1,$s2,$t1	3892	3892	
Sub $s1,$s2,$t2	3893	3893	
Sub $s1,$s2,$a0	3894	3894	
Sub $s1,$s2,$a1	3895	3895	
Sub $s1,$s2,$a2	3896	3896	
Sub $s1,$s2,$gp	389C	389C	
Sub $s1,$s2,$ra	389F	389F	
Sub $s1,$s2,$pc	389E	389E	
Sub $s1,$s2,$sp	389D	389D	
Bloco 2 – Variação no campo RegOrigem			
Sub $s2,$s0,$s1	3978	3978	
Sub $s2,$s1,$s1	3988	3988	
Sub $s2,$s2,$s1	3998	3998	
Sub $s2,$s3,$s1	39A8	39A8	
Sub $s2,$t0,$s1	3918	3918	
Sub $s2,$t1,$s1	3928	3928	
Sub $s2,$t2,$s1	3938	3938	
Sub $s2,$a0,$s1	3948	3948	
Sub $s2,$a1,$s1	3958	3958	
Sub $s2,$a2,$s1	3968	3968	
Sub $s2,$gp,$s1	39C8	39C8	
Sub $s2,$ra,$s1	39F8	39F8	
Sub $s2,$pc,$s1	39E8	39E8	
Sub $s2,$sp,$s1	39D8	39D8	
Bloco 3 – Variação no campo RegDestino			
Sub $s0,$s1,$s2	3789	3789	
Sub $s1,$s1,$s2	3889	3889	
Sub $s2,$s1,$s2	3989	3989	
Sub $s3,$s1,$s2	3A89	3A89	
Sub $t0,$s1,$s2	3189	3189	
Sub $t1,$s1,$s2	3289	3289	
Sub $t2,$s1,$s2	3389	3389	
Sub $a0,$s1,$s2	3489	3489	
Sub $a1,$s1,$s2	3589	3589	
Sub $a2,$s1,$s2	3689	3689	
Sub $gp,$s1,$s2	3C89	3C89	
Sub $ra,$s1,$s2	3F89	3F89	
Sub $pc,$s1,$s2	3E89	3E89	
Sub $sp,$s1,$s2	3D89	3D89	
Teste dos formatos			
Sub $s1,1000	Erro	Erro	#06 – Registrador inexistente no set do processador
Sub $s1,$s2,1000	Erro	Erro	#06 – Registrador inexistente no set do processador
Sub $s1, $s2,1000	Erro	Erro	#06 – Registrador inexistente no set do processador
Add $s1,$s2,1000	Erro	Erro	#06 – Registrador inexistente no set do processador

Tabela I.3 – Teste das instrução Addi

Teste da Instrução Addi			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem			
Addi $s0,20	8714	8714	
Addi $s1,30	881E	881E	
Addi $s2,00110010	Erro	Erro	#09 – Valor de offset utilizado maior do que 255 ou menor que –128.
Addi $s3,11001100	Erro	Erro	#09 – Valor de offset utilizado maior do que 255 ou menor que –128.
Addi $t0,AA	8100	81AA	
Addi $t1,BB	8200	82BB	
Addi $t2,CC	8300	83CC	
Addi $a0,A1	8400	84A1	
Addi $a1,A2	8500	85A2	
Addi $a2,1A	8601	861A	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Addi $gp,1B	8C01	8C1B	
Addi $ra,2A	8F02	8F2A	
Addi $pc,2B	8E02	8E2B	
Addi $sp,20	8D14	8D14	
Teste dos formatos			
Addi $s0,$s1,$s2	Erro	Erro	#14 – Valor de offset inválido. O mesmo deve ser numérico.
Addi $s1,$s2,00000001	Erro	Erro	#14 – Valor de offset inválido. O mesmo deve ser numérico.
Addi $s1,00000001,$s2	Erro	Erro	Compilação normal
Addi 000000001,$s1	Erro	Erro	#11 – Registrador inexistente no set do processador.
Addi $s1, 400000	Erro	Erro	#09 – Valor de offset utilizado maior do que 255 ou menor que –128.

Tabela I.4 – Teste das instrução Shift

Teste da Instrução Shift			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem			
Sft $s0,16	9710	9710	
Sft $s1,15	980F	980F	
Sft $s2,14	990E	990E	
Sft $s3,13	9A0D	9A0D	
Sft $t0,12	910C	910C	
Sft $t1,11	920B	920B	
Sft $t2,10	930A	930A	
Sft $a0,9	9409	9409	
Sft $a1,8	9508	9508	
Sft $a2,7	9607	9607	
Sft $gp,6	9C06	9C06	
Sft $ra,5	9F05	9F05	
Sft $pc,4	9E04	9E04	
Sft $sp,3	9D03	9D03	
Teste dos formatos			
Sft $s0,$s1,$s2			#07 – Valor de offset inválido. O mesmo deve ser numérico.
Sft $s1,$s2,1			#07 – Valor de offset inválido. O mesmo deve ser numérico.
Sft $s1,1,$s2			Compilação normal
Sft 1,$s1			#11 – Valor de offset utilizado inválido ou com espaço entre a vírgula e o número.

Tabela I.5 – Teste das instrução And

Teste da Instrução And			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
And $s1,$s2,$s0	4897	4897	
And $s1,$s2,$s1	4898	4898	
And $s1,$s2,$s2	4899	4899	
And $s1,$s2,$s3	489A	489A	
And $s1,$s2,$t0	4891	4891	
And $s1,$s2,$t1	4892	4892	
And $s1,$s2,$t2	4893	4893	
And $s1,$s2,$a0	4894	4894	
And $s1,$s2,$a1	4895	4895	
And $s1,$s2,$a2	4896	4896	
And $s1,$s2,$gp	489C	489C	
And $s1,$s2,$ra	489F	489F	
And $s1,$s2,$pc	489E	489E	
And $s1,$s2,$sp	489D	489D	
Bloco 2 – Variação no campo RegOrigem			
And $s2,$s0,$s1	4978	4978	
And $s2,$s1,$s1	4988	4988	
And $s2,$s2,$s1	4998	4998	
And $s2,$s3,$s1	49A8	49A8	
And $s2,$t0,$s1	4918	4918	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
And $s2,$t1,$s1	4928	4928	
And $s2,$t2,$s1	4938	4938	
And $s2,$a0,$s1	4948	4948	
And $s2,$a1,$s1	4958	4958	
And $s2,$a2,$s1	4968	4968	
And $s2,$gp,$s1	49C8	49C8	
And $s2,$ra,$s1	49F8	49F8	
And $s2,$pc,$s1	49E8	49E8	
And $s2,$sp,$s1	49D8	49D8	
Bloco 3 – Variação no campo RegDestino			
And $s0,$s1,$s2	4789	4789	
And $s1,$s1,$s2	4889	4889	
And $s2,$s1,$s2	4989	4989	
And $s3,$s1,$s2	4A89	4A89	
And $t0,$s1,$s2	4189	4189	
And $t1,$s1,$s2	4289	4289	
And $t2,$s1,$s2	4389	4389	
And $a0,$s1,$s2	4489	4489	
And $a1,$s1,$s2	4589	4589	
And $a2,$s1,$s2	4689	4689	
And $gp,$s1,$s2	4C89	4C89	
And $ra,$s1,$s2	4F89	4F89	
And $pc,$s1,$s2	4E89	4E89	
And $sp,$s1,$s2	4D89	4D89	
Teste dos formatos			
And $s1, 10			#06 – Registrador inexistente no set do processador
And $s1, $s2, 1000			#06 – Registrador inexistente no set do processador
And $s1, 1000, $s2			#06 – Registrador inexistente no set do processador

Tabela I.6 – Teste das instrução Or

Teste da Instrução Or			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Or $s1,$s2,$s0	5897	5897	
Or $s1,$s2,$s1	5898	5898	
Or $s1,$s2,$s2	5899	5899	
Or $s1,$s2,$s3	589A	589A	
Or $s1,$s2,$t0	5891	5891	
Or $s1,$s2,$t1	5892	5892	
Or $s1,$s2,$t2	5893	5893	
Or $s1,$s2,$a0	5894	5894	
Or $s1,$s2,$a1	5895	5895	
Or $s1,$s2,$a2	5896	5896	
Or $s1,$s2,$gp	589C	589C	
Or $s1,$s2,$ra	589F	589F	
Or $s1,$s2,$pc	589E	589E	
Or $s1,$s2,$sp	589D	589D	
Bloco 2 – Variação no campo RegOrigem			
Or $s2,$s0,$s1	5978	4978	
Or $s2,$s1,$s1	5988	4988	
Or $s2,$s2,$s1	5998	4998	
Or $s2,$s3,$s1	59A8	49A8	
Or $s2,$t0,$s1	5918	4918	
Or $s2,$t1,$s1	5928	4928	
Or $s2,$t2,$s1	5938	4938	
Or $s2,$a0,$s1	5948	4948	
Or $s2,$a1,$s1	5958	4958	
Or $s2,$a2,$s1	5968	4968	
Or $s2,$gp,$s1	59C8	49C8	
Or $s2,$ra,$s1	59F8	49F8	
Or $s2,$pc,$s1	59E8	49E8	
Or $s2,$sp,$s1	59D8	49D8	
Bloco 3 – Variação no campo RegDestino			
Or $s0,$s1,$s2	5789	5789	
Or $s1,$s1,$s2	5889	5889	
Or $s2,$s1,$s2	5989	5989	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Or $s3,$s1,$s2	5A89	5A89	
Or $t0,$s1,$s2	5189	5189	
Or $t1,$s1,$s2	5289	5289	
Or $t2,$s1,$s2	5389	5389	
Or $a0,$s1,$s2	5489	5489	
Or $a1,$s1,$s2	5589	5589	
Or $a2,$s1,$s2	5689	5689	
Or $gp,$s1,$s2	5C89	5C89	
Or $ra,$s1,$s2	5F89	5F89	
Or $pc,$s1,$s2	5E89	5E89	
Or $sp,$s1,$s2	5D89	5D89	
Teste dos formatos			
Or $s1,10			#06 – Registrador inexistente no set do processador
Or $s1,$s2,1000			#06 – Registrador inexistente no set do processador
Or $s1,1000,$s2			#06 – Registrador inexistente no set do processador
Or 1000,$s1,$s2			#06 – Registrador inexistente no set do processador

Tabela I.7 – Teste das instrução Xor

Teste da Instrução Xor			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Xor $s1,$s2,$s0	6897	6897	
Xor $s1,$s2,$s1	6898	6898	
Xor $s1,$s2,$s2	6899	6899	
Xor $s1,$s2,$s3	689A	689A	
Xor $s1,$s2,$t0	6891	6891	
Xor $s1,$s2,$t1	6892	6892	
Xor $s1,$s2,$t2	6893	6893	
Xor $s1,$s2,$a0	6894	6894	
Xor $s1,$s2,$a1	6895	6895	
Xor $s1,$s2,$a2	6896	6896	
Xor $s1,$s2,$gp	689C	689C	
Xor $s1,$s2,$ra	689F	689F	
Xor $s1,$s2,$pc	689E	689E	
Xor $s1,$s2,$sp	689D	689D	
Bloco 2 – Variação no campo RegOrigem			
Xor $s2,$s0,$s1	6978	6978	
Xor $s2,$s1,$s1	6988	6988	
Xor $s2,$s2,$s1	6998	6998	
Xor $s2,$s3,$s1	69A8	69A8	
Xor $s2,$t0,$s1	6918	6918	
Xor $s2,$t1,$s1	6928	6928	
Xor $s2,$t2,$s1	6938	6938	
Xor $s2,$a0,$s1	6948	6948	
Xor $s2,$a1,$s1	6958	6958	
Xor $s2,$a2,$s1	6968	6968	
Xor $s2,$gp,$s1	69C8	69C8	
Xor $s2,$ra,$s1	69F8	69F8	
Xor $s2,$pc,$s1	69E8	69E8	
Xor $s2,$sp,$s1	69D8	69D8	
Bloco 3 – Variação no campo RegDestino			
Xor $s0,$s1,$s2	6789	6789	
Xor $s1,$s1,$s2	6889	6889	
Xor $s2,$s1,$s2	6989	6989	
Xor $s3,$s1,$s2	6A89	6A89	
Xor $t0,$s1,$s2	6189	6189	
Xor $t1,$s1,$s2	6289	6289	
Xor $t2,$s1,$s2	6389	6389	
Xor $a0,$s1,$s2	6489	6489	
Xor $a1,$s1,$s2	6589	6589	
Xor $a2,$s1,$s2	6689	6689	
Xor $gp,$s1,$s2	6C89	6C89	
Xor $ra,$s1,$s2	6F89	6F89	
Xor $pc,$s1,$s2	6E89	6E89	
Xor $sp,$s1,$s2	6D89	6D89	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Teste dos formatos			
Xor $s1,10			#06 – Registrador inexistente no set do processador
Xor $s1, $s2, 1000			#06 – Registrador inexistente no set do processador
Xor $s1, 1000, $s2			#06 – Registrador inexistente no set do processador
Xor 1000, $s1, $s2			#06 – Registrador inexistente no set do processador

Tabela I.8 – Teste das instrução Not

Teste da Instrução Not			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Not $s0	A700	A700	
Not $s1	A800	A800	
Not $s2	A900	A900	
Not $s3	AA00	AA00	
Not $t0	A100	A100	
Not $t1	A200	A200	
Not $t2	A300	A300	
Not $a0	A400	A400	
Not $a1	A500	A500	
Not $a2	A600	A600	
Not $gp	AC00	AC00	
Not $ra	AF00	AF00	
Not $pc	AE00	AE00	
Not $sp	AD00	AD00	
Teste dos formatos			
Not $s1,10	A828		
Not $s1, $s2, 1000			#07 – Valor de Offset inválido ou com espaço entre a vírgula e o número
Not 1000	A000		

Tabela I.9 – Teste das instrução Lw

Teste da Instrução Lw			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Lw $s1,$s2,$s0	0897	0897	
Lw $s1,$s2,$s1	0898	0898	
Lw $s1,$s2,$s2	0899	0899	
Lw $s1,$s2,$s3	089A	089A	
Lw $s1,$s2,$t0	0891	0891	
Lw $s1,$s2,$t1	0892	0892	
Lw $s1,$s2,$t2	0893	0893	
Lw $s1,$s2,$a0	0894	0894	
Lw $s1,$s2,$a1	0895	0895	
Lw $s1,$s2,$a2	0896	0896	
Lw $s1,$s2,$gp	089C	089C	
Lw $s1,$s2,$ra	089F	089F	
Lw $s1,$s2,$pc	089E	089E	
Lw $s1,$s2,$sp	089D	089D	
Bloco 2 – Variação no campo RegOrigem			
Lw $s2,$s0,$s1	0978	0978	
Lw $s2,$s1,$s1	0988	0988	
Lw $s2,$s2,$s1	0998	0998	
Lw $s2,$s3,$s1	09A8	09A8	
Lw $s2,$t0,$s1	0918	0918	
Lw $s2,$t1,$s1	0928	0928	
Lw $s2,$t2,$s1	0938	0938	
Lw $s2,$a0,$s1	0948	0948	
Lw $s2,$a1,$s1	0958	0958	
Lw $s2,$a2,$s1	0968	0968	
Lw $s2,$gp,$s1	09C8	09C8	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Lw $s2,$ra,$s1	09F8	09F8	
Lw $s2,$pc,$s1	09E8	09E8	
Lw $s2,$sp,$s1	09D8	09D8	
Bloco 3 – Variação no campo RegDestino			
Lw $s0,$s1,$s2	0789	0789	
Lw $s1,$s1,$s2	0889	0889	
Lw $s2,$s1,$s2	0989	0989	
Lw $s3,$s1,$s2	0A89	0A89	
Lw $t0,$s1,$s2	0189	0189	
Lw $t1,$s1,$s2	0289	0289	
Lw $t2,$s1,$s2	0389	0389	
Lw $a0,$s1,$s2	0489	0489	
Lw $a1,$s1,$s2	0589	0589	
Lw $a2,$s1,$s2	0689	0689	
Lw $gp,$s1,$s2	0C89	0C89	
Lw $ra,$s1,$s2	0F89	0F89	
Lw $pc,$s1,$s2	0E89	0E89	
Lw $sp,$s1,$s2	0D89	0D89	
Teste dos formatos			
Lw $s1,10			#06 – Registrador inexistente no set do processador
Lw $s1, $s2, 1000			#06 – Registrador inexistente no set do processador
Lw $s1, 1000, $s2			#06 – Registrador inexistente no set do processador
Lw 1000, $s1, $s2			#06 – Registrador inexistente no set do processador

Tabela I.10 – Teste das instrução Sw

Teste da Instrução Sw			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Sw $s1,$s2,$s0	1897	1897	
Sw $s1,$s2,$s1	1898	1898	
Sw $s1,$s2,$s2	1899	1899	
Sw $s1,$s2,$s3	189A	189A	
Sw $s1,$s2,$t0	1891	1891	
Sw $s1,$s2,$t1	1892	1892	
Sw $s1,$s2,$t2	1893	1893	
Sw $s1,$s2,$a0	1894	1894	
Sw $s1,$s2,$a1	1895	1895	
Sw $s1,$s2,$a2	1896	1896	
Sw $s1,$s2,$gp	189C	189C	
Sw $s1,$s2,$ra	189F	189F	
Sw $s1,$s2,$pc	189E	189E	
Sw $s1,$s2,$sp	189D	189D	
Bloco 2 – Variação no campo RegOrigem			
Sw $s2,$s0,$s1	1978	1978	
Sw $s2,$s1,$s1	1988	1988	
Sw $s2,$s2,$s1	1998	1998	
Sw $s2,$s3,$s1	19A8	19A8	
Sw $s2,$t0,$s1	1918	1918	
Sw $s2,$t1,$s1	1928	1928	
Sw $s2,$t2,$s1	1938	1938	
Sw $s2,$a0,$s1	1948	1948	
Sw $s2,$a1,$s1	1958	1958	
Sw $s2,$a2,$s1	1968	1968	
Sw $s2,$gp,$s1	19C8	19C8	
Sw $s2,$ra,$s1	19F8	19F8	
Sw $s2,$pc,$s1	19E8	19E8	
Sw $s2,$sp,$s1	19D8	19D8	
Bloco 3 – Variação no campo RegDestino			
Sw $s0,$s1,$s2	1789	1789	
Sw $s1,$s1,$s2	1889	1889	
Sw $s2,$s1,$s2	1989	1989	
Sw $s3,$s1,$s2	1A89	1A89	
Sw $t0,$s1,$s2	1189	1189	
Sw $t1,$s1,$s2	1289	1289	
Sw $t2,$s1,$s2	1389	1389	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Sw $a0,$s1,$s2	1489	1489	
Sw $a1,$s1,$s2	1589	1589	
Sw $a2,$s1,$s2	1689	1689	
Sw $gp,$s1,$s2	1C89	1C89	
Sw $ra,$s1,$s2	1F89	1F89	
Sw $pc,$s1,$s2	1E89	1E89	
Sw $sp,$s1,$s2	1D89	1D89	
Teste dos formatos			
Sw $s1,10			#06 – Registrador inexistente no set do processador
Sw $s1, $s2, 1000			#06 – Registrador inexistente no set do processador
Sw $s1, 1000, $s2			#06 – Registrador inexistente no set do processador
Sw 1000, $s1, $s2			#06 – Registrador inexistente no set do processador

Tabela I.11 – Teste das instrução Lui

Teste da Instrução Lui			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem			
Lui $s0,16	B710	B710	
Lui $s1,16	B810	B710	
Lui $s2,10000	FFFF	Erro	#09 – Valor de offset utilizado maior que 255 ou menor que –128.
Lui $s3,10000	FFFF	Erro	#09 – Valor de offset utilizado maior que 255 ou menor que –128.
Lui $t0,AA	B100	B1AA	
Lui $t1,BB	B200	B2BB	
Lui $t2,CC	B300	B3CC	
Lui $a0,A1	B400	B4A1	
Lui $a1,A2	B500	B5A2	
Lui $a2,1A	B601	B61A	
Lui $gp,1B	BC01	BC1B	
Lui $ra,2A	BF02	BF2A	
Lui $pc,2B	BE02	BE2B	
Lui $sp,20	BD14	BD14	
Teste dos formatos			
Lui $s0,$s1,$s2	FFFF	Erro	#14 – Valor de offset inválido. O mesmo deve ser numérico.
Lui $s1,$s2,1	FFFF	Erro	#14 – Valor de offset inválido. O mesmo deve ser numérico.
Lui $s1,1,$s2	B801		Compilação normal
Lui 1,$s1	FFFF		#07 – Valor de offset utilizado inválido ou com espaço entre a vírgula e o número
Lui $s1,18	B812		Compilação normal

Tabela I.12 – Teste das instrução Slt

Teste da Instrução Slt			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Slt $s1,$s2,$s0	7897	7897	
Slt $s1,$s2,$s1	7898	7898	
Slt $s1,$s2,$s2	7899	7899	
Slt $s1,$s2,$s3	789A	789A	
Slt $s1,$s2,$t0	7891	7891	
Slt $s1,$s2,$t1	7892	7892	
Slt $s1,$s2,$t2	7893	7893	
Slt $s1,$s2,$a0	7894	7894	
Slt $s1,$s2,$a1	7895	7895	
Slt $s1,$s2,$a2	7896	7896	
Slt $s1,$s2,$gp	789C	789C	
Slt $s1,$s2,$ra	789F	789F	
Slt $s1,$s2,$pc	789E	789E	
Slt $s1,$s2,$sp	789D	789D	
Bloco 2 – Variação no campo RegOrigem			
Slt $s2,$s0,$s1	7978	7978	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Slt $s2,$s1,$s1	7988	7988	
Slt $s2,$s2,$s1	7998	7998	
Slt $s2,$s3,$s1	79A8	79A8	
Slt $s2,$t0,$s1	7918	7918	
Slt $s2,$t1,$s1	7928	7928	
Slt $s2,$t2,$s1	7938	7938	
Slt $s2,$a0,$s1	7948	7948	
Slt $s2,$a1,$s1	7958	7958	
Slt $s2,$a2,$s1	7968	7968	
Slt $s2,$gp,$s1	79C8	79C8	
Slt $s2,$ra,$s1	79F8	79F8	
Slt $s2,$pc,$s1	79E8	79E8	
Slt $s2,$sp,$s1	79D8	79D8	
Bloco 3 – Variação do campo RegDestino			
Slt $s0,$s1,$s2	7789	7789	
Slt $s1,$s1,$s2	7889	7889	
Slt $s2,$s1,$s2	7989	7989	
Slt $s3,$s1,$s2	7A89	7A89	
Slt $t0,$s1,$s2	7189	7189	
Slt $t1,$s1,$s2	7289	7289	
Slt $t2,$s1,$s2	7389	7389	
Slt $a0,$s1,$s2	7489	7489	
Slt $a1,$s1,$s2	7589	7589	
Slt $a2,$s1,$s2	7689	7689	
Slt $gp,$s1,$s2	7C89	7C89	
Slt $ra,$s1,$s2	7F89	7F89	
Slt $pc,$s1,$s2	7E89	7E89	
Slt $sp,$s1,$s2	7D89	7D89	
Teste dos formatos			
Slt $s1,10			#06 – Registrador inexistente no set do processador
Slt $s1, $s2, 1000			#06 – Registrador inexistente no set do processador
Slt $s1, 1000, $s2			#06 – Registrador inexistente no set do processador
Slt 1000, $s1, $s2			#06 – Registrador inexistente no set do processador

Tabela I.13 – Teste das instrução Beq

Teste da Instrução Beq			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Beq $s1,$s0,15			#12 - Valor de offset maior do que 3 ou negativo.
Beq $s1,$s1,14			#12 - Valor de offset maior do que 3 ou negativo.
Beq $s1,$s2,13			#12 - Valor de offset maior do que 3 ou negativo.
Beq $s1,$s3,1000			#12 - Valor de offset maior do que 3 ou negativo.
Beq $s1,$t0,0010			#12 - Valor de offset maior do que 3 ou negativo.
Beq $s1,$t1,A	C820	Erro	
Beq $s1,$t2,B	C830	Erro	
Beq $s1,$a0,C	C840	Erro	
Beq $s1,$a1,D	C850	Erro	
Beq $s1,$a2,E	C860	Erro	
Beq $s1,$gp,F	C8C0	Erro	
Beq $s1,$ra,1	C8F4	C8F1	
Beq $s1,$pc,2	C8E8	C8E2	
Beq $s1,$sp,3	C0DC	C0D3	
Bloco 2 – Variação no campo RegOrigem			
Beq $s0,$s1,0	C780	C780	
Beq $s1,$s1,1	C884	C884	
Beq $s2,$s1,2	C988	C988	
Beq $s3,$s1,3	CA8C	CA8C	
Beq $t0,$s1,1000			#12 - Valor de offset maior do que 3 ou negativo.
Beq $t1,$s1,0010			#12 - Valor de offset maior do que 3 ou negativo.
Beq $t2,$s1,A	C380	C380	
Beq $a0,$s1,B	C480	C480	
Beq $a1,$s1,C	C580	C580	
Beq $a2,$s1,D	C680	C680	
Beq $gp,$s1,E	CC80	CC80	
Beq $ra,$s1,F	CF80	CF80	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Beq $pc,$s1,1	CE84	CE84	
Beq $sp,$s1,2	CD88	CD88	
Teste dos formatos			
Beq $s1,10			#11 – Registrador inexistente no set do processador
Beq $s1, $s2, 16			#12 - Valor de offset maior do que 3 ou negativo.
Beq $s1, 1000, $s2			#11 – Registrador inexistente no set do processador
Beq 1000, $s1, $s2			#11 – Registrador inexistente no set do processador

Tabela I.14 – Teste das instrução Blt

Teste da Instrução Blt			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem2			
Blt $s1,$s0,15			#12 - Valor de offset maior do que 3 ou negativo.
Blt $s1,$s1,14			#12 - Valor de offset maior do que 3 ou negativo.
Blt $s1,$s2,13			#12 - Valor de offset maior do que 3 ou negativo.
Blt $s1,$s3,1000			#12 - Valor de offset maior do que 3 ou negativo.
Blt $s1,$t0,0010			#12 - Valor de offset maior do que 3 ou negativo.
Blt $s1,$t1,A	D820	Erro	
Blt $s1,$t2,B	D830	Erro	
Blt $s1,$a0,C	D840	Erro	
Blt $s1,$a1,D	D850	Erro	
Blt $s1,$a2,E	D860	Erro	
Blt $s1,$gp,F	D8C0	Erro	
Blt $s1,$ra,1	D8F4	D8F1	
Blt $s1,$pc,2	D8E8	D8F2	
Blt $s1,$sp,3	D0DC	D8F3	
Bloco 2 – Variação no campo RegOrigem			
Blt $s0,$s1,0	D780	D780	
Blt $s1,$s1,1	D884	D884	
Blt $s2,$s1,2	D988	D988	
Blt $s3,$s1,3	DA8C	DA8C	
Blt $t0,$s1,1000		Erro	#12 - Valor de offset maior do que 3 ou negativo.
Blt $t1,$s1,0010		Erro	#12 - Valor de offset maior do que 3 ou negativo.
Blt $t2,$s1,A	D380	D380	
Blt $a0,$s1,B	D480	D480	
Blt $a1,$s1,C	D580	D580	
Blt $a2,$s1,D	D680	D680	
Blt $gp,$s1,E	DC80	DC80	
Blt $ra,$s1,F	DF80	DF80	
Blt $pc,$s1,1	DE84	DE84	
Blt $sp,$s1,2	DD88	DD88	
Teste dos formatos			
Blt $s1,10			#11 – Registrador inexistente no set do processador
Blt $s1, $s2, 16			#12 - Valor de offset maior do que 3 ou negativo.
Blt $s1, 1000, $s2			#11 – Registrador inexistente no set do processador
Blt 1000, $s1, $s2			#11 – Registrador inexistente no set do processador

Tabela I.15 – Teste das instrução J

Teste da Instrução J			
Instrução com registradores	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem			
J $s0,16	E740	E710	
J $s1,16	E840	E710	
J $s2,10000	E940	Erro	
J $s3,10000	EA40	Erro	
J $t0,AA	E100	Erro	
J $t1,BB	E200	Erro	
J $t2,CC	E300	Erro	
J $a0,A1	E400	Erro	

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
J $a1,A2	E500	Erro	
J $a2,1A	E604	Erro	
J $gp,1B	EC04	Erro	
J $ra,2A	EF08	Erro	
J $pc,2B	EE08	Erro	
J $sp,20	ED50	Erro	
Teste dos formatos			
J $s0,$s1,$s2	FFFF	Erro	#14 – Valor de offset inválido. O mesmo deve ser numérico.
J $s1,$s2,1	FFFF	Erro	#14 – Valor de offset inválido. O mesmo deve ser numérico.
J $s1,1,$s2	E804	Erro	Compilação normal
J 1,$s1	FFFF	Erro	#07 – Valor de offset utilizado inválido ou com espaço entre a vírgula e o número
J $s1,18	E848	Erro	Compilação normal

Tabela I.16 – Teste das instrução Jal

Teste da Instrução Jal			
Instrução com registradores	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bloco 1 – Variação no campo RegOrigem			
Jal $s0,16	F740	F740	
Jal $s1,16	F840	F840	
Jal $s2,10000	F940	F940	#09 – Valor de offset utilizado maior que 255 ou menor que –128.
Jal $s3,10000	FA40	FA40	#09 – Valor de offset utilizado maior que 255 ou menor que –128.
Jal $t0,AA	F100	Erro	
Jal $t1,BB	F200	Erro	
Jal $t2,CC	F300	Erro	
Jal $a0,A1	F400	Erro	
Jal $a1,A2	F500	Erro	
Jal $a2,1A	F604	Erro	
Jal $gp,1B	FC04	Erro	
Jal $ra,2A	FF08	Erro	
Jal $pc,2B	FE08	Erro	
Jal $sp,20	FD50	FD50	
Teste dos formatos			
Jal $s0,$s1,$s2	FFFF	Erro	#14 – Valor de offset inválido. O mesmo deve ser numérico.
Jal $s1,$s2,1	FFFF	Erro	#14 – Valor de offset inválido. O mesmo deve ser numérico.
Jal $s1,1,$s2	F804	Erro	Compilação normal
Jal 1,$s1	FFFF	Erro	#07 – Valor de offset utilizado inválido ou com espaço entre a vírgula e o número
Jal $s1,18	F848	Erro	Compilação normal

ANEXO II - TESTES DAS PSEUDOINSTRUÇÕES

Tabela II.1 - Teste das instrução Mul

Teste da Instrução Mul			

Conjunto de microinstruções representativas da instrução Mul $s3,$s1,$s2
Add $s3,$zero,$zero
Addi $t2,1
And $t1,$s1,$t2
Beq $t1,$zero,1
Add $s3,$s3,$s2
Sft $s1,1
Sft $s2,-1
Beq $s1,$zero,1
J $zero,-6			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Mul $t1, $s0, $s1	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução
Mul $t2, $s0, $s1	Erro	Erro	#04 – Registrador '$T2' sendo utilizado pelo montador na pseudo-instrução

Mul $s2, $s0, $s1	Pseudo-Instrução: Mul
	{
	2900
	2300
	8301
	4273
	C204
	2998
	9701
	98FF
	C704
	EEE8
	}
		2900
		2300
		8301
		4273
		C204
		2998
		9701
		98FF
		C704
		EEE8	

Tabela II.2 - Teste das instrução Div

Teste da Instrução Div			
Conjunto de microinstruções representativas da instrução Div $s3,$s1,$s2 Add $s3,$zero,$zero Slt $t1 ,$s1,$s2 Sft $s3,-1 Beq $t1,$zero,1 Add $s3,$s3 ,$t1 Sft $s2,1 Beq $s2,$zero,1 J -6			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Div $t1, $s0, $s1	Pseudo-Instrução: Div
	{
	2200
	7378
	92FF
	C304
	2223
	9801
	C804
	EEE8
	}
		Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Div $s2, $s0, $s1	Pseudo-Instrução: Div
	{
	2900
	7378
	99FF
	C304
	2993
	9801
	C804
	EEE8
	}
		2900
		7378
		99FF
		C304
		2993
		9801
		C804
		EEE8	

Tabela II.3 - Teste das instrução Subi

Teste da Instrução Subi			

Conjunto de microinstruções representativas da instrução Subi $s1,100
Addi $t1,100
Sub $s1,$s1,$t1			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Subi $t1,100	Pseudo-Instrução: Subi
	{
	8364
	3223
	}
		Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Subi $s1,100	Pseudo-Instrução: Subi
	{
	8364
	3883
	}
		8364
		3883	

Tabela II.4 - Teste das instrução Muli

Teste da Instrução Muli			

Conjunto de microinstruções representativas da instrução Muli $s3,$s2,100
Addi $s1,100
Add $s3,$zero,$zero
Addi $t2,1
And $t1,$s1,$t2
Beq $t1,$Zero,1
Add $s3,$s3,$s2
Sft $s1,1
Sft $s2,-1
Beq $s1,$zero,1
J -6			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Muli $t1, $s0, $s1	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução
Muli $t2, $s0, $s1	Erro	Erro	#04 – Registrador '$T2' sendo utilizado pelo montador na pseudo-instrução

Muli $s2,$s0,100	Pseudo-Instrução: Muli
	{
	2800
	8864
	2900
	2300
	8301
	4283
	C204
	2997
	9801
	97FF
	C804
	EEE8
	}
		2800
		8864
		2900
		2300
		8301
		4283
		C204
		2997
		9801
		97FF
		C804
		EEE8	

Tabela II.5 - Teste das instrução Divi

Teste da Instrução Divi			

Conjunto de microinstruções representativas da instrução Divi $s3,$s2,100
Addi $s1,100
Add $s3,$zero,$zero
Slt $t1,$s1,$s2
Sft $s3,-1
Beq $t1,$zero,1
Add $s3,$s3,$t1
Sft $s2,1
Beq $s2,$zero,1
J –6			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Divi $t1, $s0,100	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução
Divi $s1,$s0,100	Erro	Erro	#04 – Registrador '$S1' sendo utilizado pelo montador na pseudo-instrução

Instrução	Resultado obtido	Resultado esperado	Código numérico do erro
Divi $s2,$s0,100	Erro	8864 2A00 7289 9AFF C204 2AA2 9901 C904 E000	#11 – Registrador inexistente no set do processador.

Tabela II.6 - Teste das instrução Rem

Teste da Instrução Rem			

Conjunto de microinstruções representativas da instrução Rem $s1,$s2
Add $s3,$zero,$zero
Slt $t1,$s1,$s2
Sft $s3,-1
Beq $t1,$zero,1
Add $s3,$s3,$t1
Sft $s2,1
Beq $s2,$zero,1
J -6			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Rem $s3, $s0,100	Erro	Erro	#04 – Registrador '$S3' sendo utilizado pelo montador na pseudo-instrução
Rem $t1,$s0,100	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Rem $s2,$s0,100	Pseudo-Instrução: Rem
	{
	2A00
	7297
	9AFF
	C204
	2AA2
	9701
	C704
	EEE8
	}
		2A00
		7297
		9AFF
		C204
		2AA2
		9701
		C704
		EEE8	

Tabela II.7 - Teste das instrução Sftl

Teste da Instrução Sftl			
Conjunto de microinstruções representativas da instrução Sftl $s1,100 Shift $s1,-100			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Sftl $s1,100	Erro	Erro	#08 – Valor de deslocamento utilizado maior que 16

Sftl $s1,10	Pseudo-Instrução: Sftr
	{
	980A
	}
		980A	

Tabela II.8 - Teste das instrução Sftr

Teste da Instrução Sftr			
Conjunto de microinstruções representativas da instrução Sftr $s1,100 Sft $s1,$s2,100			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Sftr $s1,100	Erro	Erro	#08 – Valor de deslocamento utilizado maior que 16

Sftr $s1,10	Pseudo-Instrução: Sftr
	{
	980A
	}		

Tabela II.9 - Teste das instrução Comp

Teste da Instrução Comp			
Conjunto de microinstruções representativas da instrução Comp $s1 Sub $s1,$ Zero,$s1			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Comp $s1	Pseudo-Instrução: Comp
	{
	3808
	}
		3808	

Tabela II.10 - Teste das instrução Andi

Teste da Instrução Andi			

Conjunto de microinstruções representativas da instrução Andi $s1,100
Add $t1,$Zero,$Zero
Addi $t1,100
And $s1,$s1,$t1			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Andi $t1,100	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Andi $s1,100	Pseudo-Instrução: Andi
	{
	2200
	8264
	4882
	}
		2200
		8264
		4882	

Tabela II.11 - Teste das instrução Ori

Teste da Instrução Ori			

Conjunto de microinstruções representativas da instrução Ori $s1,100
Add $t1,$Zero,$Zero
Addi $t1,100
Or $s1,$s1,$t1			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Ori $t1,100	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Ori $s1,100	Pseudo-Instrução: Ori
	{
	2200
	8264
	5882
	}
		2200
		8264
		5882	

Tabela II.12 - Teste das instrução Xori

Teste da Instrução Xori			

Conjunto de microinstruções representativas da instrução Xori $s1,100
Add $t1,$Zero,$Zero
Addi $t1,100
Xori $s1,$s1,$t1			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Xori $t1,100	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Xori $s1,100	Pseudo-Instrução: Xori
	{
	2200
	8264
	6882
	}
		2200
		8264
		6882	

Tabela II.13 - Teste das instrução Lb

Teste da Instrução Lb			

Conjunto de microinstruções representativas da instrução Lb $s1,$s2,$s3
Lw $t1,$s2,$s3
Shift $t1,-8
Shift $t1,8
Shift $s1,8
Shift $s1, -8
Add $s1,$s1,$t1			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Lb $t1,100	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução
Lb $s1,100	Erro do programa		Este programa executou uma operação ilegal e será fechado

Tabela II.14 - Teste das instrução Sb

Teste da Instrução Sb			

Conjunto de microinstruções representativas da instrução Sb $s1,$s2,$s3
Add $t1,$Zero,$s1
Sft $t1,$Zero,-8
Sft $t1,$Zero,+8
Lw $t2,$s2,$s3
Sft $t2,$Zero,+8
Sft $t2,$Zero,-8
Add $s1,$t1,$t2
Sw $s1,$s2,$s3			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Sb $t1,$s2,$s3	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução
Sb $t2, $s2,$s3	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução
Sb $t2, $s2,$s3	Erro do Windows	2208 9208 92F8 039A 9308 9308 2823 189A	Invalid pointer operation

Tabela II.15 - Teste das instrução Ld

Teste da Instrução Ld			

Conjunto de microinstruções representativas da instrução Ld $s1,$s2,(100)$s3
Add $t1,$Zero,$Zero
Addi $t1,100
Lw $s1,$t1,$s3
Addi $s3,1
Lw $s2,$t1,$s3			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Ld $t1,$s2,(100)$s3	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Ld $t2, $s2,(100)$s3	Pseudo-Instrução: Ld
	{
	2200
	8264
	032A
	8A01
	092A
	}
		2200
		8264
		032A
		8A01
		092A	

Tabela II.16 - Teste das instrução Sd

Teste da Instrução Sd			

Conjunto de microinstruções representativas da instrução Sd $s1,$s2,(100)$s3
Add $t1,$Zero,$Zero
Addi $t1,100
Sw $s1,$t1,$s3
Addi $s3,1
Sw $s2,$t1,$s3			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Sd $t1,$s2,(100)$s3	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Sd $t2,
$s2,(100)$s3
	Pseudo-Instrução: Sd
	{
	2200
	8264
	132A
	8A01
	192A
	}
		2200
		8264
		132A
		8A01
		192A	

Tabela II.17 - Teste das instrução Mov

Teste da Instrução Mov			

Conjunto de microinstruções representativas da instrução Mov $s1,$s2
Add $t1,$s1,$Zero
Add $s1,$s2,$Zero
Add $s2,$t1,$Zero			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Mov $t1,$s2	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Mov $s2,$s3	Pseudo-Instrução: Mov
	{
	2290
	29A0
	2A20
	}
		2290
		29A0
		2A20	

Tabela II.18 - Teste das instrução Mflo

Teste da Instrução Mflo			
Conjunto de microinstruções representativas da instrução Mflo $s1,$s2 Shift $s1,$Zero,-8			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Mflo $t1,$s2	Pseudo-Instrução: Mflo
	{
	92F8
	}
		92F8	

Tabela II.19 - Teste das instrução Mfhi

Teste da Instrução Mfhi			
Conjunto de microinstruções representativas da instrução Mfhi $s1,$s2 Shift $s1,$Zero,+8			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Mfhi $t1,$s2	Pseudo-Instrução: Mfhi
	{
	9208
	}
		9208	

Tabela II.20 - Teste das instrução Move

Teste da Instrução Move			

Conjunto de microinstruções representativas da instrução Move $s1,$s2
Add $s1,$s2,$Zero
Add $s1,$Zero,$Zero			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Move $t1,$s2	Pseudo-Instrução: Move
	{
	2290
	2900
	}
		2290
		2900	

Tabela II.21 - Teste das instrução Chg

Teste da Instrução Chg			

Conjunto de microinstruções representativas da instrução Chg $s1,$s2
Add $t1,$s1,$Zero
Add $s1,$s2,$Zero
Add $s2,$t1,$Zero			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Chg $t1,$s2	Erro	Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Chg $s2,$s3	Pseudo-Instrução: Chg
	{
	2290
	29A0
	2A20
	}
		2290
		29A0
		2A20	

Tabela II.22 - Teste das instrução Sle

Teste da Instrução Sle			
Conjunto de microinstruções representativas da instrução Sle $s1,$s2,$s3 Slt $s1,$s3,$s2 Not $s1			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Sle $s1,$s2,$s3	Erro do programa	78A9 A800	Invalid Pointer Operation

Tabela II.23 - Teste das instrução Seq

Teste da Instrução Seq			

Conjunto de microinstruções representativas da instrução Seq $s1,$s2,$s3
Add $s1,$Zero,$Zero
Addi $s1,1
Beq $s1,$s2,2
Add $s1,$Zero,$Zero			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Seq $s1,$s2,$s3	Pseudo-Instrução: Seq
	{
	2800
	8801
	C9A8
	2800
	}
		2800
		8801
		C9A8
		2800	

Tabela II.24 - Teste das instrução Sne

Teste da Instrução Sne			

Conjunto de microinstruções representativas da instrução Sne $s1,$s2,$s3
Add $s1,$Zero,$Zero
Beq $s1,$s2,2
Addi $s1,1			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Sne $s1,$s2,$s3	Pseudo-Instrução: Sne
	{
	2800
	C9A8
	8801
	}
		2800
		C9A8
		8801	

Tabela II.25 - Teste das instrução Sgt

Teste da Instrução Sgt			
Conjunto de microinstruções representativas da instrução Sgt $s1,$s2,$s3 Slt $s1,$s3,$s2			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Sgt $s1,$s2,$s3	Pseudo-Instrução: Sgt
	{
	78A9
	}
		78A9	

Tabela II.26 - Teste das instrução Sge

Teste da Instrução Sge			
Conjunto de microinstruções representativas da instrução Sge $s1,$s2,$s3 Slt $s1,$s2,$s3 Not $s1			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Sge $s1,$s2,$s3	Erro no programa	789A A800	Invalid pointer operation

Tabela II.27 - Teste das instrução Bne

Teste da Instrução Bne			
Conjunto de microinstruções representativas da instrução Bne $s1,$s2,5 Beq $s1,$s2,2 J $pc,5			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Bne $s1,$s2,16	Pseudo-Instrução: Bne
	{
	C892
	EE10
	}
		C892
		EE10	

Bne $s1,$s2,7	Pseudo-Instrução: Bne
	{
	C892
	EE07
	}
		C892
		EE07	

Tabela II.28 - Teste das instrução Bgt

Teste da Instrução Bgt			
Conjunto de microinstruções representativas da instrução Bne $s1,$s2,5 Blt $s2,$s1,5			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Blt $s1,$s2,16	Erro	Erro	#12 – Valor do deslocamento maior que 7 ou menor que -7
Blt $s1,$s2,7	D897	D897	

Tabela II.29 - Teste das instrução Slti

Teste da Instrução Slti			

Conjunto de microinstruções representativas da instrução Slti $s1,$s2,100
Add $s3,$Zero,$Zero
Addi $s3,100
Slt $s1,$s2,$s3			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Slti $s3,$s2,100	Pseudo-Instrução: Slti
	{
	2300
	8364
	7A93
	}
		Erro	#04 – Registrador '$S3' sendo utilizado pelo montador na pseudo-instrução

Slti $s1,$s2,100	Pseudo-Instrução: Slti { 2300 8364 7893	2300 8364 7893	

Tabela II.30 - Teste das instrução Seqi

Teste da Instrução Seqi			

Conjunto de microinstruções representativas da instrução Seqi $s1,$s2,100
Add $s3,$Zero,$Zero
Addi $s3,100
Add $s1,$Zero,$Zero
Addi $s1,1
Beq $s2,$s3,2
Add $s1,$Zero,$Zero			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Seqi $s3,$s2,16	Pseudo-Instrução: Seqi
	{
	2300
	8310
	2A00
	8A01
	C938
	2A00
	}
		Erro	#04 – Registrador '$S3' sendo utilizado pelo montador na pseudo-instrução

Seqi $s1,$s2,7	Pseudo-Instrução: Seqi
	{
	2300
	8307
	2800
	8801
	C938
	2800
	}
		2300
		8307
		2800
		8801
		C938
		2800	

Tabela II.31 - Teste das instrução Sgti

Teste da Instrução Sgti			

Conjunto de microinstruções representativas da instrução Sgti $s1,$s2,100
Add $s3,$Zero,$Zero
Addi $s3,100
Slt $s2,$s1,$s3			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Sgti $s3,$s2,16	Pseudo-Instrução: Sgti
	{
	2300
	8310
	7A39
	}
		Erro	#04 – Registrador '$S3' sendo utilizado pelo montador na pseudo-instrução

Sgti $s1,$s2,7	Pseudo-Instrução: Sgti
	{
	2300
	8307
	7839
	}
		2300
		8307
		7839	

Tabela II.32 - Teste das instrução Beqi

Teste da Instrução Beqi			

Conjunto de microinstruções representativas da instrução Beqi $s1,$s2,100
Add $s2,$Zero,$Zero
Addi $s2,100
Beq $s1,$s2,5			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Beqi $s2,10,16		Erro	#12 – Valor de desvio maior que 7 ou menor que -7

Beqi $s2,10,7	Pseudo-Instrução: Beqi
	{
	2300
	830A
	C937
	}
		2300
		830A
		C937	

Tabela II.33 - Teste das instrução Bnei

Teste da Instrução Bnei			

Conjunto de microinstruções representativas da instrução Bnei $s1,100,5
Add $s2,$Zero,$Zero
Addi $s2,100
Beq $s1,$s2,2
J $pc,5			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Bnei $s2,10,16	Pseudo-Instrução: Bnei
	{
	2300
	830A
	C932
	EE10
	}
		2300
		830A
		C932
		EE10
			#04 – Registrador '$S2' sendo utilizado pelo montador na pseudo-instrução

Bnei $s2,10,7	Pseudo-Instrução: Bnei
	{
	2300
	830A
	C932
	EE07
	}
		2300
		830A
		C932
		EE07	

Tabela II.34 - Teste das instrução Blti

Teste da Instrução Blti			

Conjunto de microinstruções representativas da instrução Blti $s1,100,5
Add $s2,$Zero,$Zero
Addi $s2,100
Blt $s1,$s2,5			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Blti $s2,10,16		Erro	#12 – Valor de desvio maior que 7 ou menor que -7

Blti $s2,10,7	Pseudo-Instrução: Blti
	{
	2300
	830A
	DA37
	}		

Tabela II.35 - Teste das instrução Bgti

Teste da Instrução Bgti			

Conjunto de microinstruções representativas da instrução Bgti $s1,100,5
Add $s2,$Zero,$Zero
Addi $s2,100+1
Blt $s1,$s2,5			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Bgti $s2,10,16	Erro	Erro	#12 – Valor de desvio maior que 7 ou menor que -7

Bgti $s3,10,7	Pseudo-Instrução: Bgti
	{
	2300
	830B
	DA37
	}		

Tabela II.36 - Teste das instrução Jr

Teste da Instrução Jr			
Conjunto de microinstruções representativas da instrução Jr $s1 J $s1,0			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Jr $s1	Erro	E800	#11 – Registrador inexistente no set do processador
Jr $s2	Erro	E900	#11 – Registrador inexistente no set do processador

Tabela II.37 - Teste das instrução Jpc

Teste da Instrução Jpc			
Conjunto de microinstruções representativas da instrução Jpc 100 J $pc,100			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Jpc 100	Erro	E800	#10 – Valor de offset utilizado maior que 63 ou menor que –32.
Jpc 256	Erro	E8FF	#11 – Registrador inexistente no set do processador.

Jpc 63	Pseudo-Instrução: Jpc
	{
	EEFC
	}
		EEFC	

Tabela II.38 - Teste das instrução Jalr

Teste da Instrução Jalr			
Conjunto de microinstruções representativas da instrução Jalr $s1,$s2,100 Jal $s2,100 Add $s1,$ra,$Zero			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Jalr $ra,$s1,63	Erro do programa	E800	Invalid Pointer Operation
Jalr $s1,$s2,256	Erro	E900	#10 – Valor de offset utilizado maior que 63 ou menor que –32.
Jalr $s1,$s2,63	Erro do programa	EEFC	Invalid Pointer Operation

Tabela II.39 - Teste das instrução Jalpc

Teste da Instrução Jalpc			
Conjunto de microinstruções representativas da instrução Jalpc 100 Jal $pc,100			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Jalpc 100	Erro	Erro	#10 – Valor de offset utilizado maior que 63 ou menor que –32.
Jalpc 256	Erro	Erro	#10 – Valor de offset utilizado maior que 63 ou menor que –32.

Jalpc 63	Pseudo-Instrução: Jalpc
	{
	FEFC
	}
		FEFC	

Tabela II.40 - Teste das instrução Jd

Teste da Instrução Jd			
Conjunto de microinstruções representativas da instrução Jd 100 Add $s1,$Zero,$Zero Lui $s1,mais significativa J $s1,menos significativa			
Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Jd 100	Erro	Erro	#10 – Valor de offset utilizado maior que 63 ou menor que –32.

Jd 63	Pseudo-Instrução: Jd
	{
	2300
	B300
	E3FC
	}
		2300
		B300
		E3FC	

Tabela II.41 - Teste das instrução Jald

Teste da Instrução Jald			

Conjunto de microinstruções representativas da instrução Jald $s1,100
Add $t1,$Zero,$Zero
Lui $t1,mais significativa
Add $s1,$s1,$t1
Jal $s1,menos significativa			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**
Jald $t1,63	Erro do programa	Erro pelo uso do registrador $t1	Invalid pointer operation
Jald $s1,63	Erro do programa	2200 B203 2882 F83C	Invalid pointer operation

Tabela II.42 - Teste das instrução Jalrd

Teste da Instrução Jalrd			

Conjunto de microinstruções representativas da instrução Jalrd $s1,$s2,100
Add $t1,$Zero,$Zero
Lui $t1,mais significativa
Add $s2,$s2,$t1
Jal $s2,menos significativa
Add $s1,$ra,$Zero			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Jalrd $t1,$s1,63	Pseudo-Instrução: Jald
	{
	2300
	B300
	2883
	F8FC
	22F0
	}
		Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Jalrd $s1,$s2,63	Pseudo-Instrução: Jald
	{
	2300
	B300
	2993
	F9FC
	28F0
	}		

Tabela II.43 - Teste das instrução Jalpcd

Teste da Instrução Jalpcd			

Conjunto de microinstruções representativas da instrução Jalpcd $s1,100
Add $t1,$Zero,$Zero
Lui $t1,mais significativa
Add $t1,$pc,$t1
Jal $t1,menos significativa			

Instrução	**Resultado obtido**	**Resultado esperado**	**Código numérico do erro**

Jalpcd $t1,63	Pseudo-Instrução:
	Jalpcd
	{
	2300
	B300
	23E3
	F33F
	22F0
	}
		Erro	#04 – Registrador '$T1' sendo utilizado pelo montador na pseudo-instrução

Jalpcd $s1,63	Pseudo-Instrução:
	Jalpcd
	{
	2300
	B300
	23E3
	F33F
	28F0
	}
		2300
		B300
		23E3
		F33F
		28F0	

ANEXO III - O CONJUNTO COMPLETO DA APLICAÇÃO, INCLUINDO A ROTINA DE TRATAMENTO DE EXCEÇÕES EM LINGUAGEM DE MÁQUINA

Tabela III.1 – Conjunto programacional completo da aplicação

Label	Instrução	Comentários
	J Interrupção	Realiza salto incondicional para posição em que se encontra a rotina de tratamento de interrupções.
Posição início aplicação	Lwi $t0, pl_sd	Carrega dado proveniente da pilha ΣΔ
	Lw $t1, $t0, $zero	
	Lw $t2, $t1, $zero	
	Lwi $s0, proc	Armazena o dado na posição de processamento
	Sw $t2, $s0, $zero	
	Lwi $t2, fim_de_pilha_SD	Testa se a posição do ponteiro é válida. Se chegou à última posição, recoloca o ponteiro no topo da pilha. Se não, ajusta o ponteiro, incrementando-o e armazenando-o novamente
	Beq $t1,$t2, 5	
	Addi $t1,0001h	
	Sw $t1,$t0,$zero	
	Lwi $a1, Dados_Controle	
	J $a1, 0	
	Lwi $t1, $início_de_pilha	
	Sw $t1,$t0,$zero	
Dados_Controle	Lwi $s1 $TX_controle	Vai para rotina de envio de dados de controle
	Jal $s1,0	
	Lwi $t0, const_mul	Carrega o dado e realiza a multiplicação por uma constante
	Lw $a0, $t0, $zero	
	Lwi $t0, proc	
	Lw $a1, $t0, $zero	
	Mul $s1, $a0, $a1	
	Lwi $s2, result	Armazena o resultado
	Sw $s1, $s2, $zero	
	Lwi $s1 $TX_controle	Vai para rotina de envio de dados de controle
	Jal $s1,0	
	Lwi $s0, result	Coloca o resultado na posição que vai ser utilizada pela rotina de envio para RF que fica armazenada em memória como constante dado_RF
	Lw $s1, $s0, $zero	
	Lwi $s0, dado_RF	
	Sw $s1, $s0, $zero	
	Lwi $s1, $TX_RF	Chama rotina de envio de dados para RF
	Jal $s1, 0	
	Lwi $s0, Posição_início_aplicação	Retorna ao início para novo processamento
	Jal $s0, 0	
TX_controle	Lwi $a0, Guarda_$ra	Armazena $ra para retorno da subrotina
	Sw $ra, $a0, $zero	
	Lwi $t0, dado_Serial	Armazena $pc no endereço para envio de dados para Serial
	Sw $pc,$t0, $zero	
	Lwi $t0, $TX_Serial	Chama subrotina de envio de dados para Serial
	Jal $t0, 0	
	Lwi $t0, dado_serial	Armazena $int no endereço para envio de dados para Serial
	Sw $int,$t0, $zero	
	Lwi $t0, $TX_Serial	Chama subrotina de envio de dados para Serial
	Jal $t0, 0	
	Lwi $t0, Guarda_$ra	Reestabelece o valor de $ra antes da chamada ao procedimento de controle
	Lw $ra, $t0, $zero	
	J $ra,0	Retorna à execução normal do programa
TX_Serial	Lwi $t1, FFF9h	Carrega *Status* da interface Serial
	Lw $t0,$t1, $zero	
	Andi $t0,0000 0000 0000 00001	Isola o bit menos significativo (ready), para testar o estado da interface.
	Beq $t1,$zero, Ready_Serial	Se a interface está pronta, vai para Ready_Serial
	Lwi $t0, Cód_Erro_TX_Serial	Se não está pronta, envia código de erro para a interface serial
	Lw $t0, $t0, $zero	
	Lwi $t1, Dado_Serial	
	Sw $t0,$t1,$zero	
	J TX_Serial	
Ready_Serial	Lwi $s0,dado_serial	Carrega dado a ser enviado para a interface Serial
	Lw $s1,$s0,$zero	
	Lwi $s2, FFF8h	Armazena o dado no registrador apropriado mapeado em memória
	Sw $s1, $s2, $zero	
	Lwi $t0, setup_TX_serial	Carrega palavra de *setup* de transmissão Serial
	Lw $t1, $t0, $zero	

Label	Instrução	Comentários
	Lwi $t2, FFFAh	Armazena setup no registrador apropriado mapeado em memória
	Sw $t1, $t2, $zero	
	J $ra,0	Retorna à execução normal do programa
TX_RF	Lwi $t1, FFFD	Carrega palavra de *status* de RF
	Lw $t0,$t1, $zero	
	Andi $t0,0000 0000 0000 00001	Isola o bit menos significativo (ready), para testar o estado da interface.
	Beq $t0,$zero, Ready_RF	Se a interface está pronta, vai para Ready_RF
	Lwi $t0, Código_de_Erro_RF	Se não está pronta, envia código de erro para a interface serial
	Lw $t0, $t0, $zero	
	Lwi $t1,Dado_Serial	
	Sw $t0,$t1,$zero	
	J TX_Serial	
Ready_RF	Lwi $s0, dado_RF	Carrega o dado a ser transmitido pela interface de RF
	Lw $s1, $s0, $zero	
	Lwi $s2, FFFC	Armazena o dado no registrador apropriado mapeado em memória
	Sw $s1, $s2, $zero	
	Lwi $t0, setup_TX_RF1	Carrega palavra de setup de transmissão de RF
	Lw $t1, $t0, $zero	
	Lwi $t2, FFFE	Armazena setup no registrador apropriado mapeado em memória
	Sw $t1, $t2, $zero	
	Lwi $t0, setup_TX_RF2	Carrega palavra de setup de transmissão de RF
	Lw $t1, $t0, $zero	
	Lwi $t2, FFFF	Armazena setup no registrador apropriado mapeado em memória
	Sw $t1, $t2, $zero	
	J $ra	Retorna a execução do programa
PI_SD	Ponteiro leitura pilha ΣΔ	Posições de memória que contém outras posição de memória dentro das pilhas respectivas
Pe_SD	Ponteiro escrita pilha ΣΔ	
Cód_Erro_TX_RF		Código para o erro resultante da tentativa de transmissão para interface de RF
Cód_Erro_TX_Serial		Código para o erro resultante da tentativa de transmissão para interface Serial
Cód_Erro_RX_RF		Código para o erro resultante da tentativa de aquisição de dados da interface de RF
Cód_Erro_RX_Serial		Código para o erro resultante da tentativa de aquisição de dados da interface Serial
Cód_Erro_Overflow		Código de erro de *Overflow*
Cód_Erro_Endereçamento		Código de erro de endereçamento
Proc	Dado a ser processado	Dado proveniente da interface ΣΔ
Const_mul	Constante para multiplicação	Constante para multiplicação
Result	Resultado	Resultado do processamento
Dado_RF	Dado de saída RF	Dado a ser usado pela rotina de envio RF
Dado_serial	Dado de saída serial	Dado a ser usado para rotina de envio Serial
Setup_TX_Serial	Setup de transmissão Serial	Palavra de setup de transmissão Serial
Setup_TX_RF1	Setup de Transmissão RF	1ª Palavra de setup de transmissão RF
Setup_TX_RF2	Setup 2 de transmissão RF	2ª Palavra de setup de transmissão RF
Salv1	Pilha de salvamento do contexto	Posições de memória reservadas para o salvamento do contexto na ocorrência de uma interrupção
Salv2	Pilha de salvamento do contexto	
Salv3	Pilha de salvamento do contexto	
Salv4	Pilha de salvamento do contexto	
Salv5	Pilha de salvamento do contexto	
Salv6	Pilha de salvamento do contexto	
Salv7	Pilha de salvamento do contexto	
Salv8	Pilha de salvamento do contexto	
Salv9	Pilha de salvamento do contexto	
Salv10	Pilha de salvamento do contexto	
Salv11	Pilha de salvamento do contexto	
Salv12	Pilha de salvamento do contexto	
Salv13	Pilha de salvamento do contexto	
Guarda_$ra	Salva $ra	Salva $ra quando da chamada à rotina de envio de dados de controle
Interrupção	Lui $s3, FF	Desabilita interrupção setando o bit 0 do endereço $FFEB
	Addi $s3, EB	
	Sw $s3, $s3, $zero	
	Lui $s3, MSBSalv1	Carrega primeiro endereço de salvamento do contexto
	Addi $s3, LSBSalv1	
	Sw $ra, $s3, $zero	Guarda os 13 registradores de contexto usando $s3 como base
	Addi $s3, 0001	
	Sw $sp, $s3, $zero	
	Addi $s3, 0001	
	Sw $t0, $s3,$zero	

Label	Instrução	Comentários
	Addi $s3, 0001	
	Sw $t1, $s3, $zero	
	Addi $s3, 0001	
	Sw $t2, $s3, $zero	
	Addi $s3, 0001	
	Sw $a0, $s3, $zero	
	Addi $s3, 0001	
	Sw $a1, $s3,$zero	
	Addi $s3, 0001	
	Sw $a2, $s3, $zero	
	Addi $s3, 0001	
	Sw $s0, $s3, $zero	
	Addi $s3, 0001	
	Sw $s1, $s3, $zero	
	Addi $s3, 0001	
	Sw $s2, $s3, $zero	
	Addi $s3, 0001	
	Sw $int, $s3, $zero	
	Addi $s3,0001	
	Sw $gp,$s3,$zero	
	Lwi $t1,dado_serial	Transfere conteúdo de $int para saída serial
	Sw $int,$t1,$zero	
	Lwi $s1,TX_Serial	
	Jal $s1,0	
	Andi $t0,$int,$000000000000 1000_b	Testes sucessivos com o conteúdo de $int para identificar que tipo de interrupção aconteceu e transferir controle para a rotina de tratamento adequada
	Bgt $t0,$zero, 9(Endereçamento)	
	Andi $t0,$int, 0000000000000100_b	
	Bgt $t0,$zero, 8(*Overflow*)	
	Andi $t0,$int,$0000 0000 0000 0 0 01_b	
	Bgt $t0, $zero, 7(RF)	
	And $t0, $int, $0000 0000 0000 0 0 10_b	
	Bgt $t0, $zero, 6(Serial)	
	And $t0, $int, $0000000000001 0 11_b	
	Bgt $t0, $zero, 5 ($\Sigma\Delta$)	
	Jd IntEndereçamento	Desvios para rotinas de tratamento individuais
	Jd zero,IntOverflow	
	Jd zero,IntRF	
	Jd zero,IntSerial	
IntSD	Lwi $t0,pe_SD	Carrega os dois ponteiros (leitura e escrita) para testes e verificação da possibilidade de escrita em pilha
	Lw $t0,$t0,$zero	
	Lwi $t1,pl_SD	
	Lw $t1,$t1,$zero	
	Lwi $s1, ultima posição da pilha	Testa se a próxima posição a ser escrita é a última da pilha
	Beq $t0, $s1, 2	
	Jd not_EOS_SD	Pula para próximo teste
	Lwi $s0,FFF3	Caso tenha sido alcançado o EOS, habilita BCP, usado para sinalização
	Lwi $s1,0000 0000 0000 0010	
	Sw $s1,$s0,$zero	
Not_EOS_SD	Bgt $t1, $t0, 3	Se pl_SD > pe_SD, testa $FFF3
	Lwi $s0, FFF3	Se pl_SD <pe_SD e BCP=0 escreve dado. Se BCP=1, posição inválida.
	Lw $s0,$s0,$zero	
	Beq $s0,$zero, 7	
	Jd etapa_final	Finaliza tratamento
	Beqi $s0, 0000, 5	Se pl_SD > pe_SD e BCP=1 escreve dado
	Lwi $a0, 0001	Testa se pl_SD - 1 < pe_SD
	Sub $a1, $t1, $a0	
	Blt $t0, $a1, 2	
	J etapa_final	Finaliza tratamento
Wr_SD	Lwi $s1, FFF4h	Carrega dado enviado pela SD
	Lw $s1, $s1, $zero	
	Lwi $t0, pe_SD	Carrega ponteiro de escrita SD
	Lw $t0, $t0, $zero	
	Sw $s1, $t0, $zero	Armazena dado enviado na pilha usando o ponteiro
	Bnei $t0, ultima_posição_da_pilha, 4	Se a última posição utilizada para escrita foi a última da pilha, retorna o ponteiro ao topo da pilha
	Lwi $t2, primeira_posição_da_pilha	
	Sw $t2, $t1, $zero	
	Jd etapa_final	
	Addi $t0, 0001	Se não é a última posição, ajusta o ponteiro de escrita, somando 1 ao seu conteúdo
	Lwi $t1, pe_SD	
	Sw $t0, $t1, $zero	
	Jd etapa_final	Finaliza tratamento

Label	Instrução	Comentários
IntRF	Lwi $t1,FFFD	Teste para identificar interrupção gerada de transmissão ou recepção
	Lw $t1,$t1,$zero	
	Andi $t1,0000 0000 0001 0000	
	Beq $t1,$zero, RX_RF	Se interrupção gerada por recepção, envia código para Serial
	Lwi $t1, FFFD	Carrega palavra de status
	Lw $t1,$t1,$zero	
	And $t1, 0000 0000 0000 0010	Testa se houve erro, usando os 15 bits mais significativos do status. Se um deles é diferente de 0, houve erro.
	Beq $t1,$zero, ErroTX_RF	Se houve erro, desvia para rotina adequada
Not_erro_RF	Lwi $t0,0000 0000 0000 0001	Carrega ready = 1 no status da interface, significando que ela deve estar pronta para nova transmissão
	Lwi $t1, FFFD	
	Sw $t0,$t1,$zero	
ErroTX_RF	Lwi $t0, Cód _Erro_TX_RF	Carrega código de erro de TX RF e armazena no dado a ser usado pela interface de transmissão para serial e chama a subrotina de envio de dados para serial
	Lw $t0, $t0, $zero	
	Lwi $t1, Dado_Serial	
	Sw $t0,$t1,$zero	
	Jal TX_Serial	
	J etapa_final	Finaliza tratamento
RX_RF	Lwi $t0, Cód _Erro_RX_RF	Carrega código de erro de RX RF
	Lw $t0,$t0,$zero	
	Lwi $t1, Dado_Serial	Armazena o código de erro na posição especificada e chama a subrotina de envio de dados para a Serial
	Sw $t0,$t1,$zero	
	Jal TX_Serial	
	Jd etapa_final	Finaliza tratamento
IntSerial	Lwi $t1,FFFAh	Teste para identificar interrupção gerada de transmissão ou recepção
	Lw $t1,$t1,$zero	
	Andi $t1,0000 0000 0001 0000	
	Beq $t1,$zero, RX_Serial	Se interrupção gerada por recepção, envia código para Serial
	Lwi $t1, FFFAh	Carrega palavra de status
	Lw $t1,$t1,$zero	
	And $t1, 0000 0000 0000 0010	Testa se houve erro, usando os 15 bits mais significativos do status. Se um deles é diferente de 0, houve erro.
	Beq $t1,$zero, ErroTX_Serial	Se houve erro, desvia para rotina adequada
Não_erro_Serial	Lwi $t0,0000 0000 0000 0001	Carrega ready = 1 no status da interface, significando que ela deve estar pronta para nova transmissão
	Lwi $t1, FFFAh	
	Sw $t0,$t1,$zero	
ErroTX_Serial	Lwi $t0, Cód _Erro_TX_Serial	Carrega código de erro de TX Serial e armazena no dado a ser usado pela interface de transmissão e chama a subrotina de envio de dados para serial
	Lw $t0, $t0, $zero	
	Lwi $t1, Dado_Serial	
	Sw $t0,$t1,$zero	
	Jal TX_Serial	
	J etapa_final	Finaliza tratamento
RX	Lwi $t0, Cód_Erro_RX_Serial	Carrega código de erro de RX Serial
	Lw $t0,$t0,$zero	
	Lwi $t1, Dado_Serial	Armazena o código de erro na posição especificada e chama a subrotina de envio de dados para a Serial
	Sw $t0,$t1,$zero	
	Jal TX_Serial	
	J etapa_final	Finaliza tratamento
Overflow	Lwi $t0, salv14	Carrega $int salvo em memória
	Lw $t1, $t0, $zero	
	Andi $t1, $0000 0000 0001 1111	Seta $pc dentro de $int = 0001
	Sw $t1, $t0, $zero	Armazena $int na mesma posição
	Lwi $a0, código_overflow	Carrega código para ocorrência de overflow e envia para interface de RF
	Lwi $a1, dado_RF	
	Sw $a0, $a1, $zero	
	J TX_RF	
	J etapa_final	Finaliza tratamento
Endereçamento	Lwi $t0, salv14	Carrega $int salvo em memória
	Lw $t1, $t0, $zero	
	Andi $t1, $0000 0000 0000 1111	Seta $pc dentro de $int = 0001
	Sw $t1, $t0, $zero	Armazena $int na mesma posição
	Lwi $a0, código_endereçamento	Carrega código para ocorrência de endereçamento e envia para interface de RF
	Lwi $a1, dado_RF	
	Sw $a0, $a1, $zero	
	J TX_RF	
Etapa_Final	Lwi $s3, $salv1	Devolve o conteúdo dos 14 registradores para retorno à execução normal do programa
	Sw $s3, $ra, $zero	

Label	Instrução	Comentários
	Addi $s3, $0001	
	Sw $s3, $sp, $zero	
	Addi $s3, $0001	
	Sw $s3, $t0, $zero	
	Addi $s3, $0001	
	Sw $s3, $t1, $zero	
	Addi $s3, $0001	
	Sw $ s3, $t2, $zero	
	Addi $s3, $0001	
	Sw $s3, $a0, $zero	
	Addi $s3, $0001	
	Sw $ s3, $a1, $zero	
	Addi $s3, $0001	
	Sw $ s3, $a2, $zero	
	Addi $s3, $0001	
	Sw $s3, $s0, $zero	
	Addi $s3, $0001	
	Sw $s3, $s1, $zero	
	Addi $s3, $0001	
	Sw $s3, $s2, $zero	
	Addi $s3, $0001	
	Sw $s3, $s3, $zero	
	Addi $s3, $0001	
	Sw $ s3, $int, $zero	
	Addi $s3, $0001	
	Sw $ s3, $pc, $zero	
	Addi $s3, $0001	
	Sw $s3, $int, $zero	
	Lui $s3, FF	Habilitar ocorrência de nova interrupção zerando o bit zero do endereço FFEB usando $int que terá sempre o último bit 0 depois do tratamento de dados
	Addi $s3, EB	
	Sw $int, $s3, $zero	
	Lwi $s3, $int	Recupera o valor de $pc armazenado no $int
	Lw $s3, $s3, $zero	
	Andi $s3, $1111 1111 1111 0000	
	Sft $s3, 4	Desloca o resultado do andi pois $pc só tem 12 bits
	Addi $s3, 1	Coloca $pc na posição anterior +1
	J $s3	Volta para execução normal do programa

ANEXO IV - MONTAGEM DO BLOCO DA APLICAÇÃO EM LINGUAGEM DE MONTAGEM

Tabela IV.1 – Bloco de montado da aplicação em linguagem de montagem

Endereço	Código em linguagem de máquina	Microinstrução	Pseudoinstrução	Ciclos gastos
$000	E0FF	J $zero,205	J $zero,Interrupção	3
$001	2100	Add $t0,$zero,$zero	Lwi $t0, pl_SD	4
$002	B100	Lui $t0, 0		4
$003	81AF	Addi $t0,175		4
$004	0210	Lw $t1,$t0,$zero	Lw $t1, $t0, $zero	5
$005	0320	Lw $t2,$t1,$zero	Lw $t2, $t1, $zero	5
$006	2700	Add $s0,$zero,$zero	Lwi $s0, proc	4
$007	B700	Lui $s0,0		4
$008	87B7	Addi $s0,183		4
$009	1370	Sw $t2,$s0,$zero	Sw $t2, $s0, $zero	4
$00A	2300	Add $t2,$zero,$zero	Lwi $t2, fim_de_pilha_SD	4
$00B	B3FF	Lui $t2, 255		4
$00C	83FF	Addi $t2, 255		4
$00D	C235	Beq $t1,$t2,5	Beq $t1,$t2,5	3
$00E	8201	Addi $t1,1	Addi $t1, 0001h	4
$00F	1210	Sw $t1,$t0,$zero	Sw $t1, $t0, $zero	4
$010	2500	Add $a1,$zero,$zero	Lwi $a1, Dados_Controle	4
$011	B500	Lui $a1,00		4
$012	8546	Addi $a1, 70		4
$013	E518	J $a1,24	J $a1, 0	3
$014	2200	Add $t1,$zero,$zero	Lwi $t1, $início_de_pilha	4
$015	B204	Lui $t1,4		4
$016	8200	Addi $t1, 0		4
$017	1210	Sw $t1,$t0,$zero	Sw $t1, $t0, $zero	4
$018	2800	Add $s1,$zero,$zero	Lwi $s1 $TX_controle	4
$019	B800	Lui $s1,0		4
$01A	8846	Addi $s1,70		4
$01B	F800	Jal $s1,0	Jal $s1,0	4
$01C	2100	Add $t0,$zero,$zero	Lwi $t0, const_mul	4
$01D	B100	Lui $t0,0		4
$01E	81B8	Addi $t0,184		4
$01F	0410	Lw $a0,$t0,$zero	Lw $a0, $t0, $zero	5
$020	2100	Add $t0,$zero,$zero	Lwi $t0, proc	4
$021	B100	Lui $t0,0		4
$022	81B7	Addi $t0,183		4
$023	0510	Lw $a1,$t0,$zero	Lw $a1, $t0, $zero	5
$024	2800	Add $s1,$zero,$zero	Mul $s1, $a0, $a1	4
$025	2300	Add $t2,$zero,$zero		4
$026	8301	Addi $t2,1		4

Endereço	Código em linguagem de máquina	Microinstrução	Pseudoinstrução	Ciclos gastos
$027	4243	And $t1,$a0,$t2		4
$028	C201	Beq $1,$zero,1		3
$029	2885	Add $s1,$s1,$a1		4
$02A	9401	Sft $a0,1		4
$02B	95FF	Sft $a1,-1		4
$02C	C401	Beq $a0,$zero,1		3
$02D	EEFA	J $pc -6		3
$02E	2900	Add $s2,$zero,$zero	Lwi $s2, result	4
$02F	B900	Lui $s2,0		4
$030	81BB	Addi $t0,185		4
$031	1890	Sw $s1,$s2,$zero	Sw $s1, $s2, $zero	4
$032	2800	Add $s1,$zero,$zero	Lwi $s1 $TX_controle	4
$033	B800	Lui $s1,0		4
$034	88AA	Addi $s1,170		4
$035	F800	Jal $s1,0	Jal $s1,0	4
$036	2700	Add $s0,$zero,$zero	Lwi $s0, result	4
$037	B700	Lui $s0,0		4
$038	81BB	Addi $s0, 185		4
$039	0870	Lw $s1,$s0,$zero	Lw $s1, $s0, $zero	5
$03A	2700	Add $s0,$zero,$zero	Lwi $s0, dado_RF	4
$03B	B700	Lui $s0,0		4
$03C	82BA	Addi $s0,186		4
$03D	1870	Sw $s1,$s0,$zero	Sw $s1, $s0, $zero	4
$03E	2800	Add $s1,$zero,$zero	Lwi $s1 $TX_RF	4
$03F	B800	Lui $s1,0		4
$040	8883	Addi $s1,131		4
$041	F800	Jal $s1,0	Jal $s1,0	4
$042	2700	Add $s0,$zero,$zero	Lwi $s0,0001h	4
$043	B700	Lui $s0,0		4
$044	8701	Addi $s0,1		4
$045	F700	Jal $s0,0	Jal $s0,0	4
$046	2400	Add $a0,$zero,$zero	Lwi $a0, Guarda_$ra	4
$047	B400	Lui $a0,0		4
$048	81CC	Addi $a0, 204		4
$049	1F40	Sw $ra,$a0,$zero	Sw $ra, $a0, $zero	4
$04A	2100	Add $t0,$zero,$zero	Lwi $t0, dado_Serial	4
$04B	B100	Lui $t0,0		4
$04C	81BB	Addi $t0,187		4
$04D	1E10	Sw $pc,$t0,$zero	Sw $pc,$t0, $zero	4
$04E	2100	Add $t0,$zero,$zero	Lwi $t0, $TX_Serial	4
$04F	B100	Lui $t0,0		4
$050	815F	Addi $t0,95		4
$051	F100	Jal $t0,0	Jal $t0,0	4
$052	2100	Add $t0,$zero,$zero	Lwi $t0, dado_serial	4
$053	B100	Lui $t0,0		4

Endereço	Código em linguagem de máquina	Microinstrução	Pseudoinstrução	Ciclos gastos
$054	81BB	Addi $t0,187		4
$055	1B10	Sw $int,$t0,$zero	Sw $int,$t0, $zero	4
$056	2100	Add $t0,$zero,$zero	Lwi $t0, $TX_Serial	4
$057	B100	Lui $t0,0		4
$058	815F	Addi $t0,95		4
$059	F100	Jal $t0,0	Jal $t0,0	4
$05A	2100	Add $t0,$zero,$zero	Lwi $t0, Guarda_$ra	4
$05B	B100	Lui $t0, 0		4
$05C	81CC	Addi $t0, 204		4
$05D	0F10	Lw $ra,$t0,$zero	Lw $ra, $t0, $zero	5
$05E	EF00	J $ra,0	J $ra,0	3
$05F	2100	Add $t0,$zero,$zero	Lwi $t, FFF9	4
$060	B2FF	Lui $t1,255		4
$061	82F9	Addi $t1,249		4
$062	0120	Lw $t0,$t1,$zero	Lw $t0,$t1,$zero	5
$063	2200	Add $t1,$zero,$zero	Andi $t0,1	4
$064	8201	Addi $t1,1		4
$065	4112	And $t0,$t0,$t1		4
$066	C207	Beq $t1,$zero,7	Beq $t1,$zero,Ready_Serial	3
$067	2700	Add $s0,$zero,$zero	Lwi $t0,Cód_Erro_TX_Serial	4
$068	B700	Lui $s0,0		4
$069	B8B1	Addi $s0,178		4
$06A	0110	Lw $t0,$t0,$zero	Lw $t0,$t0,$zero	5
$06B	2200	Add $t1,$zero,$zero	Lwi $t1, Dado_serial	4
$06C	B200	Lui $t1,0		4
$06D	82BB	Addi $t1,187		4
$06E	1120	Sw $t0,$t1,$zero	Sw $t0,$t1,$zero	4
$06F	2800	Add $s1,$zero,$zero	Jd TX_Serial	4
$070	B800	Lui $s1,0		4
$071	E85F	J $s1,95		3
$072	2700	Add $s0,$zero,$zero	Lwi $s0, dado_serial	4
$073	2800	Lui $s0,0		4
$074	B801	Addi $s0,187		4
$075	0870	Lw $s1,$s0,$zero	Lw $s1, $s0, $zero	5
$076	2900	Add $s2,$zero,$zero	Lwi $s2, FFF8h	4
$077	B9FF	Lui $s2,255		4
$078	89F8	Addi $s2,248		4
$079	1890	Sw $s1,$s2,$zero	Sw $s1, $s2, $zero	4
$07A	2100	Add $t0,$zero,$zero	Lwi $t0, setup_TX_serial	4
$07B	B100	Lui $t0, 0		4
$07C	81BC	Addi $t0,188		4
$07D	0210	Lw $t1,$t0,$zero	Lw $t1, $t0, $zero	5
$07E	2300	Add $t2,$zero,$zero	Lwi $t2, FFFAh	4
$07F	89FF	Lui $t2,255		4
$080	B9FA	Addi $t2,250		4
$081	1230	Sw $t1,$t2,$zero	Sw $t1, $t2, $zero	4

Endereço	Código em linguagem de máquina	Microinstrução	Pseudoinstrução	Ciclos gastos
$082	EF00	J $ra,0	J $ra,0	3
$083	2200	Add $t1,$zero,$zero	Lwi $t1,FFFD	4
$084	B2FF	Lui $t1,255		4
$085	82FD	Addi $t1,253		4
$086	0120	Lw $t0,$t1,$zero	Lw $t0,$t1,$zero	5
$087	2200	Add $t1,$zero,$zero	Andi $t0,1	4
$088	8201	Addi $t1,1		4
$089	4112	And $t0,$t0,$t1		4
$08A	C102	Beq $t0,$zero,2	Beq $t0,$zero,Ready_RF	3
$08B	E07E	J $zero, 151		3
$08C	2100	Add $t0,$zero,$zero	Lwi $t0,Cód_Erro_TX_RF	4
$08D	B100	Lui $t0,0		4
$08E	81B1	Addi $t0, 177		4
$08F	0110	Lw $t0,$t0,$zero	Lw $t0,$t0,$zero	5
$090	2200	Add $t1,$zero,$zero	Lwi $t1,dado_serial	4
$091	B200	Lui $t1,0		4
$092	82B9	Addi $t1, 187		4
$093	1120	Sw $t0,$t1,$zero	Sw $t0,$t1,$zero	4
$094	8200	Addi $s1,$zero,$zero	Jd $TX_Serial	4
$095	B800	Lui $s1,0		4
$096	E85F	J $s1,95		3
$097	2700	Add $s0,$zero,$zero	Lwi $s0, dado_RF	4
$098	B700	Lui $s0,0		4
$099	87BA	Addi $s0,186		4
$09A	2900	Add $s2,$zero,$zero	Lwi $s2,FFFC	4
$09B	B9FF	Lui $s2,255		4
$09C	89FB	Addi $s2,252		4
$09D	1890	Sw $s1,$s2,$zero	Sw $s1,$s2,$zero	4
$09E	2100	Add $t0,$zero,$zero	Lwi $t0, setup_RF_1	4
$09F	B100	Lui $t0,0		4
$0A0	81BD	Addi $t0,189		4
$0A1	0210	Lw $t1,$t0,$zero	Lw $t1,$t0,$zero	5
$0A2	2100	Add $t2,$zero,$zero	Lwi $t2,FFFE	4
$0A3	B1FF	Lui $t2,255		4
$0A4	81FE	Addi $t2,254		4
$0A5	1230	Sw $t1,$t2,$zero	Sw $t1, $t2, $zero	4
$0A6	2100	Add $t0,$zero,$zero	Lwi $t0, Setup_RF_2	4
$0A7	B100	Lui $t0,0		4
$0A8	81BE	Addi $t0, 190		4
$0A9	0210	Lw $t1,$t0,$zero	Lw $t1, $t0, $zero	5
$0AA	2100	Add $t0,$zero,$zero	Lwi $t1,FFFF	4
$0AB	B1FF	Lui $t0,255		4
$0AC	81FF	Addi $t0,255		4
$0AD	1230	Sw $t1,$t2,$zero	Sw $t1,$t2,$zero	4
$0AE	EF00	J $ra,0	J $ra,0	3
$0AF	FFFF	PI_AD	Posição do final do programa	X
$0B0	0040	Pe_AD	Posição do final do programa +1	X
$0B1	0001	Cód_Erro_TX_RF	1	X

Endereço	Código em linguagem de máquina	Microinstrução	Pseudoinstrução	Ciclos gastos
$0B2	0002	Cód_Erro_TX_Serial	2	X
$0B3	0003	Cód_Erro_RX_RF	3	X
$0B4	0000	Cód_Erro_RX_Serial	0	X
$0B5	7FA7	Cód_Erro_Overflow	32769	X
$0B6	7FA8	Cód_Erro_Endereçamento	32768	X
$0B7	0000	Proc	0	X
$0B8	00FF	Const_Mul	0	X
$0B9	0000	Result	0	X
$0BA	0000	Dado_RF	0	X
$0BB	0000	Dado_Serial	0	X
$0BC	5555	Setup_TX_Serial	0101010101010101	X
$0BD	9999	Setup_TX_RF	1001100110011001	X
$0BE	8101	Setup_TX_RF2	1000000100000001	X
$0BF	0000	Salv1 ($ra)	0	X
$0C0	0000	Salv2 ($sp)	0	X
$0C1	0000	Salv3 ($t0)	0	X
$0C2	0000	Salv4 ($t1)	0	X
$0C3	0000	Salv5 ($t2)	0	X
$0C4	0000	Salv6 ($a0)	0	X
$0C5	0000	Salv7 ($a1)	0	X
$0C6	0000	Salv8 ($a2)	0	X
$0C7	0000	Salv9 ($s0)	0	X
$0C8	0000	Salv10 ($s1)	0	X
$0C9	0000	Salv11 ($s2)	0	X
$0CA	0000	Salv12 ($int)	0	X
$0CB	0000	Salv13 ($gp)	0	X
$0CC	0000	Guarda_$ra	0	X