Documento de Arquitetura

Risc

Build 1.0

Histórico de Revisões

Data	Descrição	Autor(s)
14/12/2015	Estruturação do documento	Patricia Gomes
19/12/2015	Adição dos datapath's por instrução	Fábio Barros
19/12/2015	Correção das tabelas	Matheus Borges
19/12/2015	Criação do capítulo Assembly	Matheus Borges
19/12/2015	Criação das tabelas de entrada e saída dos módulos	Patricia Gomes
19/12/2015	Preenchimento da tabela Acrônimos e Abreviações	Matheus Borges
19/12/2015	Preenchimento da tabela da tabela de sinais da unidade de controle	Fábio Barros
19/12/2015	Criação do capítulo Requisitos do Processador	Patricia Gomes
20/12/2015	Finalização do documento	Fábio Barros, Matheus Borges e Patricia Gomes
20/12/2015	Revisão do documento	Fábio Barros e Matheus Borges
15/01/2016	Alterações no documento	Patricia Gomes
29/01/2016	Correções do documento	Patricia Gomes

CONTEÚDO

1. Introdução

1.1. Objetivo

O processador pode ser definido como o cérebro do computador, é ele o responsável por realizar todas as instruções dentro do computador como operações de lógicas e cálculos. Além disso ele é responsável pela tomada de decisões do sistema.

O objetivo deste documento de arquitetura é definir as especificações do processador desenvolvido. O documento de arquitetura é importante por permitir que outras pessoas possam utilizá-lo para construir um sistema a partir dele com sucesso. O mesmo define os parâmetros de implementação que compõem os requisitos do processador implementado, tais requisitos incluem a arquitetura do conjunto de instruções, definições de entrada e saída e a arquitetura geral do processador.

1.2. Organização do Documento

Sessão 2: Apresenta uma definição de requisitos funcionais e não funcionais.
Sessão 3: Apresenta a visão geral da arquitetura.
Sessão 4: Especifica o conjunto de instruções do processador.
Sessão 5: Especifica os elementos que compõem o sistema.
Sessão 6: Descreve o montador.

1.3. Acrônimos e Abreviações

Sigla	Descrição
GPR	Registrador de Propósito Geral
ISA	Instruction Set Architecture - Conjunto de Instruções da Arquitetura
IF	Instruction Fetch - Busca da Instrução
ID	Instruction Decode - Decoficação da Instrução
RF	Register Fetch - Acesso aos Registradores
EX	Execute - Execução da Instrução
MEM	Memory - Acesso à Memória
WB	Write Back - Escrita de volta
OP	Operation Code - Código da Operação
RA	Read A - Ler A
RB	Read B - Ler B
continua na próxima página	

continuação da tabela anterior	
Sigla	**Descrição**
WC	Write C - Escreve C
COND	Condição
Const16	Constante de 16 bits
MI	Memory Instruction (Memória de Instruções)
PC	Program Counter (Contador de Programa)
RB	Banco de Registradores
UC	Unit Control (Unidade de Controle)
ALU	Arithmetic Logic Unit (Unidade Lógica Aritmética)
RF	Register Flag (Registrador de Flags)
TF	Tests Flag (Testador de Flags)
MX	Multiplexer (Multiplexador)
DM	Data Memory (Memória de Dados)
SE	Signal Extender (Extensor de Sinal)
RISC	Reduced Instruction Set Computer (Computador com um Conjunto Reduzido de Instruções)

Tabela 2: Tabela de acrônimos e abreviações

2. Requisitos do Processador

É requerido que o processador desenvolvido seja capaz de executar instruções que possibilitem gerenciar a máquina de forma extremamente rápida e eficiente.

A estrutura dos requisitos funcionais e não funcionais é descrita ao longo deste documento. Sendo requisito funcional e não funcional definidos na tabela abaixo:

Termos	Descrição
Requisito Funcional	Requisitos de hardware que compõem os módulos, descrevendo as ações que o mesmo deve estar apto a executar.
Requisito Não Funcional	Requisitos de hardware que compõem os módulos, representando as características que o mesmo deve ter, ou restrições que o mesmo deve operar.

Tabela 3: Tabela de Definição

Os requisitos funcionais e não-funcionais são listados nas tabelas abaixo respectivamente:

Requisitos Funcionais	Descrição
ISA composta por 42 instruções.	O processador deve ser capaz de executar 42 instruções.

Tabela 4: Tabela de Requisitos Funcionais

Requisitos Não-Funcionais	Descrição
Possuir 16 GPR de 32 bits de largura	O processador deve possuir 15 registradores de propósito geral, sendo que cada registrador deve ser capaz de armazenar uma palavra de 32 bits.
Arquitetura de 32 bits	O processador deve suportar operações com operandos que possuam 32 bits.

Tabela 5: Tabela de Requisitos Não-Funcionais

Os requisitos foram atendidos, de forma que as instruções, apresentadas nas sessões seguintes, foram sugeridas a partir dos documentos de testes que foi entregue pelos solicitantes. A operação que cada instrução realiza é descrita ao lado da instrução.

3. Visão Geral da Arquitetura

Este documento descreve um processador µRISC adaptado à 32 bits. Esse processador possui 16 registradores de propósito geral e é composto por um conjunto de 42 instruções.

Para uma instrução ser executada são necessários quatro ciclos. Esses ciclos são denominados IF, ID, EX/MEN e WB, como apresentado na figura abaixo. No primeiro ciclo (IF), a próxima instrução a ser executada, definida pelo contador de programa (PC), é lida da memória de instruções (MI). No segundo ciclo (ID), a instrução é identificada e os operandos são lidos do banco de registradores ou, em caso de operações com constantes, manipulados pelo extensor de sinal (SE). No terceiro ciclo (EX/MEM), a instrução é executa pela unidade lógica aritmética (ALU) ou pela memória, em caso de operações de armazenamento ou leitura. No quarto ciclo (WB), os resultados são escritos no banco de registradores.

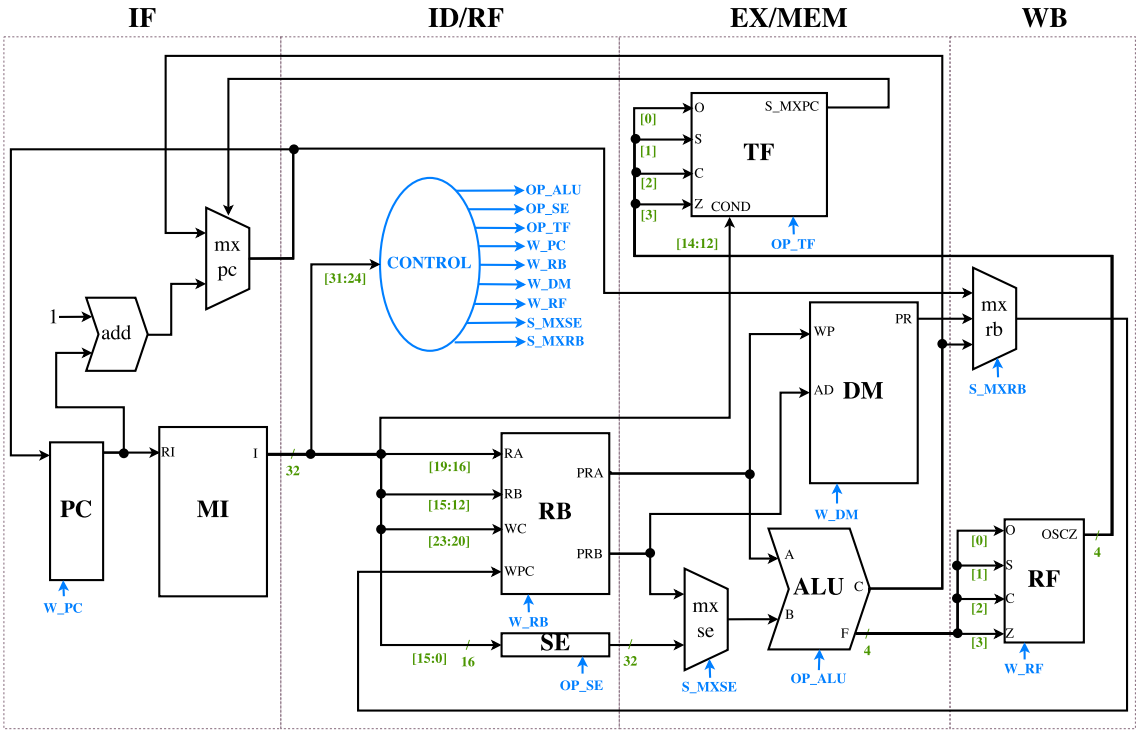

Figura 1: **Datapath Geral**

3.1. Principais características

- **Arquitetura de 32 bits;**
- **16 GPR de 32 bits de largura (r0 ... r15);**
- **ISA composta por 42 instruções;**
- **Instruções de até 3 operandos;**

- **Simulador e Montador desenvolvido na linguagem C;**
- **Armazenamento em memória de forma big-endian;**
- **Unidade de controle hardwired;**
- **Endereçamento por registrador e imediato.**

4. Arquitetura das Instruções

O conjunto de instruções do processador foi dividido nos seguintes grupos:

- **Instruções lógicas e aritméticas;**
- **Instruções com constante;**
- **Instruções de acesso à memória;**
- **Instruções de desvio;**
- **Instruções de desvio por registrador;**
- **NOP;**
- **HALT;**

4.1. Instruções Lógicas e Aritméticas

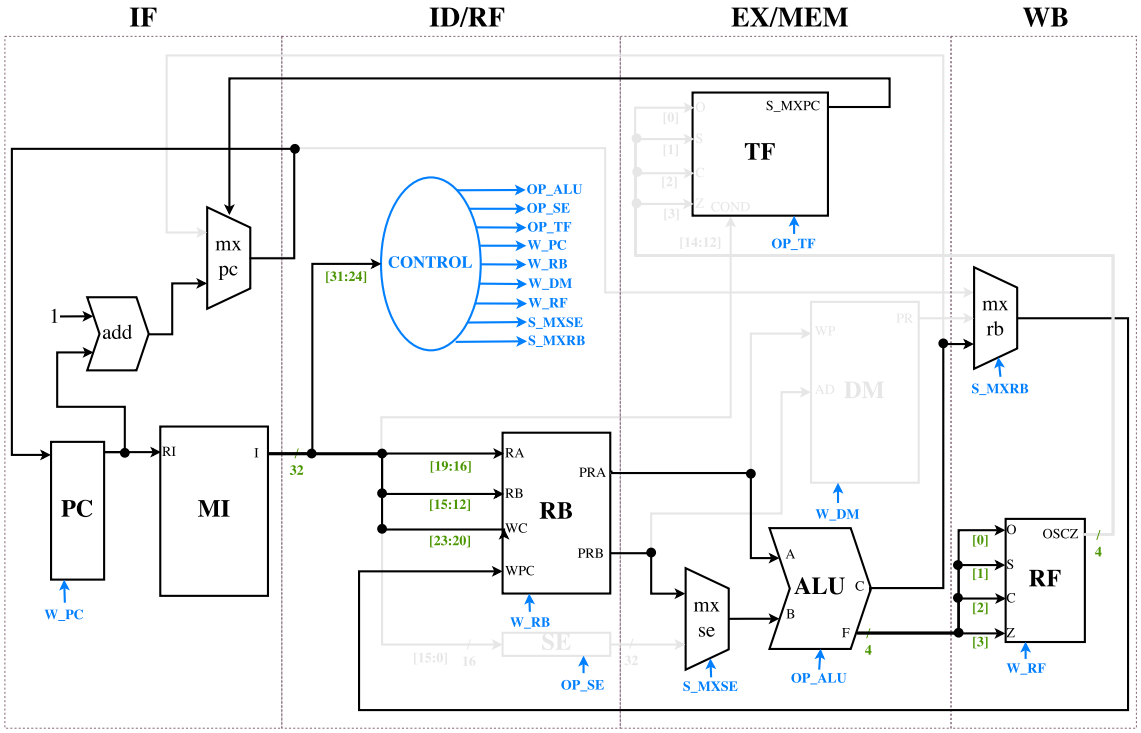

Figura 2: **Datapath de Lógicas Aritméticas**

A Figura 2 apresenta o caminho de dados específico de instruções lógicas e aritméticas, nela é possível observar quais elementos participam da realização destas operações.

Os elementos que caracterizam esse tipo de instrução são:

- Unidade logica aritmética (ALU): responsável por realiza a operação e gerar os sinais de flags;
- Registradores de flags (RF): responsável por armazena o último estado das flags atualizadas;

As instruções de lógica e aritmética possuem o seguinte formato:

31:29	28:24	23:20	19:16	15:12	11:0
0 0 1	OP	WC	RA	RB	X X X X X X X X X X X X

Tabela 6: **Tabela de Formato**

OPALU	Mnemônico	Operação	Flags Atualizadas
00000	add c,a,b	C = A + B	O S C Z
00001	addinc c,a,b	C = A + B + 1	O S C Z
00011	inca c,a	C = A + 1	O S C Z
00100	subdec c,a,b	C = A − B − 1	O S C Z
00101	sub c, a, b	C = A − B	O S C Z
00110	deca c, a	C = A − 1	O S C Z
01000	lsl c, a	C = Deslocamento Lógico Esq. (A)	S C Z
01001	asr c, a	C = Deslocamento Aritmético Dir. (A)	S C Z
10000	zeros c	C = 0	Z
10001	and c, a, b	C = A&B	S Z
10010	andnota c,a,b	C = !A&B	S Z
10011	passb c, b	C = B	S Z
10100	andnotb c, a, b	C = A&!B	S Z
10101	passa, c, a	C = A	S Z
10110	xor c, a, b	C = AˆB	S Z
10111	or c, a, b	C = A | B	S Z
continua na próxima página			

continuação da tabela anterior			
OPALU	**Mnemônico**	**Operação**	**Flags Atualizadas**
11000	nand c, a, b	C = !A&!B	S Z
11001	xnor c, a, b	C = !(A^B)	S Z
11010	passnota c, a	C = !A	S Z
11011	ornota c, a, b	C = !A|B	S Z
11100	passnotb c, b	C = !B	S Z
11101	ornotb c, a, b	C = A|!B	S Z
11110	nor c, a, b	C = !A|!B	S Z
11111	ones c	C = 1	

4.2. Instruções com Constante

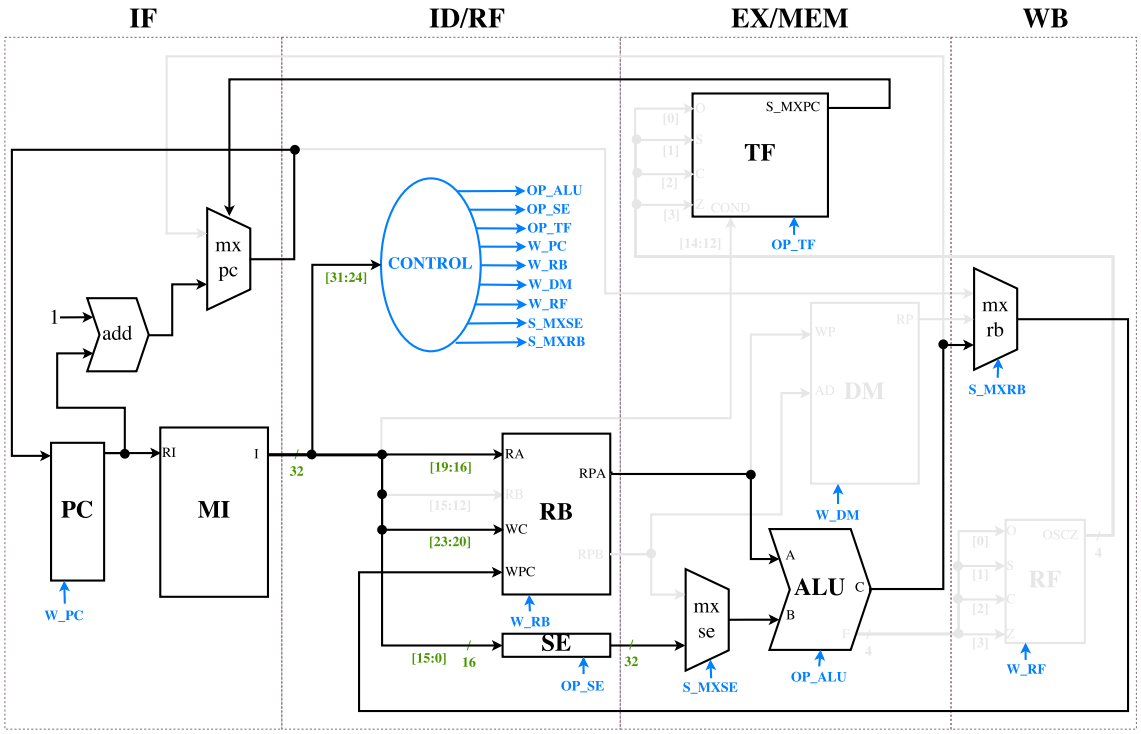

Figura 3: **Datapath de Constantes**

A Figura 3 apresenta o caminho de dados específico de instruções com constantes, nela é possível observar quais elementos participam da realização destas operações.

O elemento que caracteriza esse tipo de instrução é o extensor de sinal responsável por transformar uma constante de 16 ou 12 bits em 32 bits. O nível lógico do sinal OP_SE define se a constante a ser extendida é de tamanho 12 bits no caso de operações de jump ou 16 bits no caso de operações com constante. Assim é garantido que o restante dos bits não tenha informações erradas que irão ser operadas na unidade lógica e aritimetica (ALU).

31:29	28:24	23:20	19:16	15:0
0 1 0	OP	WC	X X X X	CONSTANTE

Tabela 8: **Tabela de Formato**

OPALU	Mnemônico	Operação
01100	loadlit c, Const16	C = CONSTANTE
01101	lcl c, Const16	C = Const16|(C&0xffff0000)
01110	lch c, Const16	C = (Const16 « 16)|(C&0x0000ffff)

Tabela 9: **Tabela de Instruções com Constante**

4.3. Instruções de Acesso à Memória

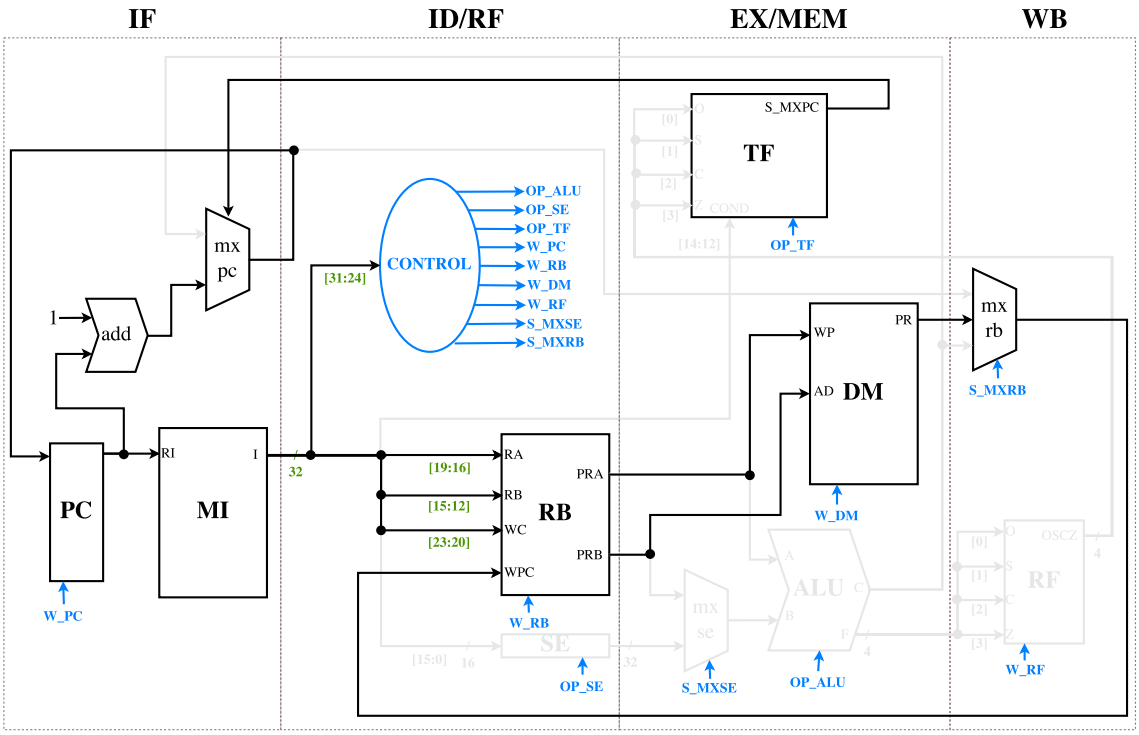

Figura 4: **Datapath Memória**

As instruções de acesso à memória são Load e Store. Na instrução Load o endereço presente no registrador B é lido da memória, e a saída é escrita no banco de registradores no endereço especificado na pelo registrador A. Na instrução Store os dados são lidos do banco de registradores e escritos na memória, sendo registrador A o dado a ser escrito e o registrador B o endereço onde será armazenado.

O sinal que habilita a escrita na memória de dados é o mesmo que habilita a leitura, quando em nível lógico 1 apenas a escrita é permitida, e em 0, apenas a leitura.

As instruções de acesso à memória possuem o seguinte formato:

31:29	28:25	24	23:20	19:16	15:12	11:0
1 0 0	X X X X	OP	WC	RA	RB	X X X X X X X X X X X X

Tabela 10: **Tabela de Formato**

OPM	Mnemônico	Operação
0	load c, a	C = Mem[A]
1	store a, b	Mem[A] = B

Tabela 11: **Tabela de Instruções de Acesso à Memória**

4.4. Instruções de Desvio

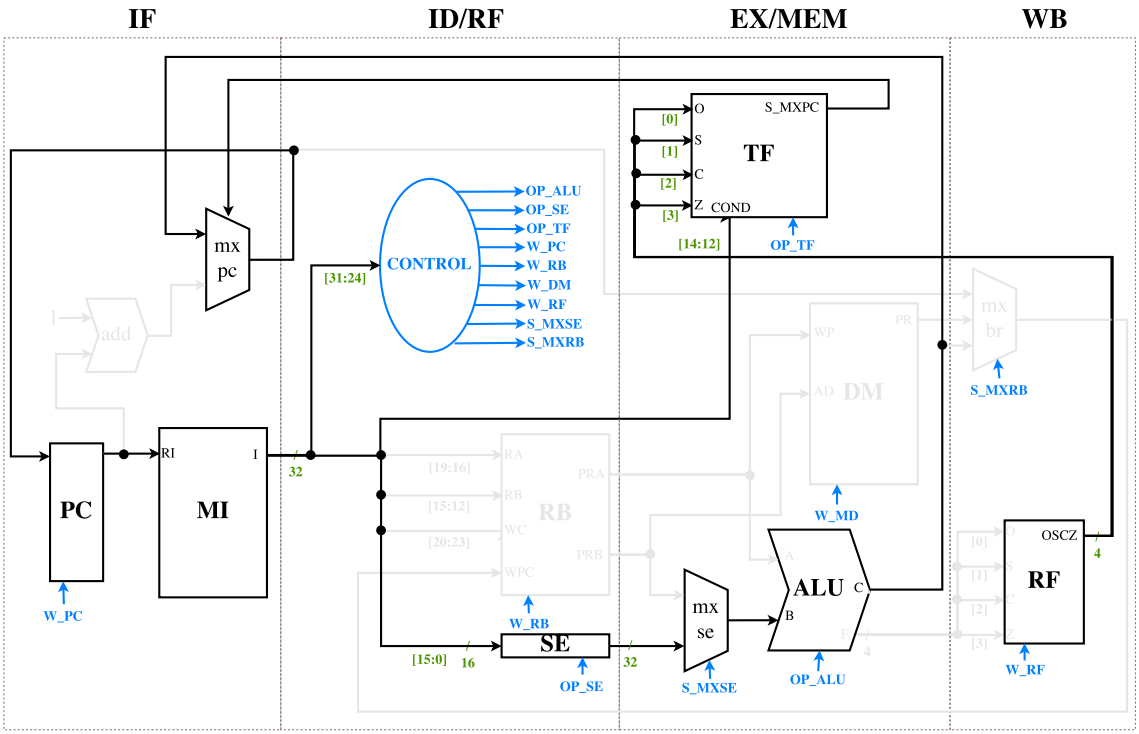

Figura 5: **Datapath Desvio Condicional**

O módulo que caracteriza essa instrução é o testador de flags (TF) que, a partir de uma condição, determina se um salto será ou não realizado.

As instruções de desvio condicional e incondicional realizam um salto para um endereço absoluto da memória de instruções, informado pelo campo DESTINO (demonstrado na Tabela de Formato). Este endereçamento pode ser de até 2^12 bits. Em caso de endereços maior que 12 bits, deve-se utilizar as instruções lcl e lch para carregar o endereço do Label de destino em um registrador. Em seguida, deve-se realizar um desvio por registrador (especificado na sessão 4.5), utilizando como parâmetro este registrador. Esta decisão de utilziar endereçamento absoluto retira da ALU a responsabilidade de calcular o novo destino.

Tais instruções possuem o formato descrito abaixo na tabela:

31:29	28:27	26:24	23:15	14:12	11:0
000	X X	OPTF	X X X X X X X X X	COND	DESTINO

Tabela 12: Tabela de Formato

OPTF	Mnemônico	Operação
000	jf.cond DESTINO	Jump False
001	jt.cond DESTINO	Jump True
010	j DESTINO	Jump Incondicional

Tabela 13: Tabela de Instruções de Desvio

As instruções de desvio condicional devem testar as condições apresentadas no quadro abaixo:

COND	Mnemônico	Condição
000	true	TRUE
001	neg	Resultado da ALU negativo
010	zero	Resultado da ALU zero
100	carry	Carry da ALU
101	negzero	Resultado da ALU negativo ou zero
111	overflow	Resultado da ALU overflow

Tabela 14: Tabela de Condições

4.5. Instruções de Desvio por Registrador

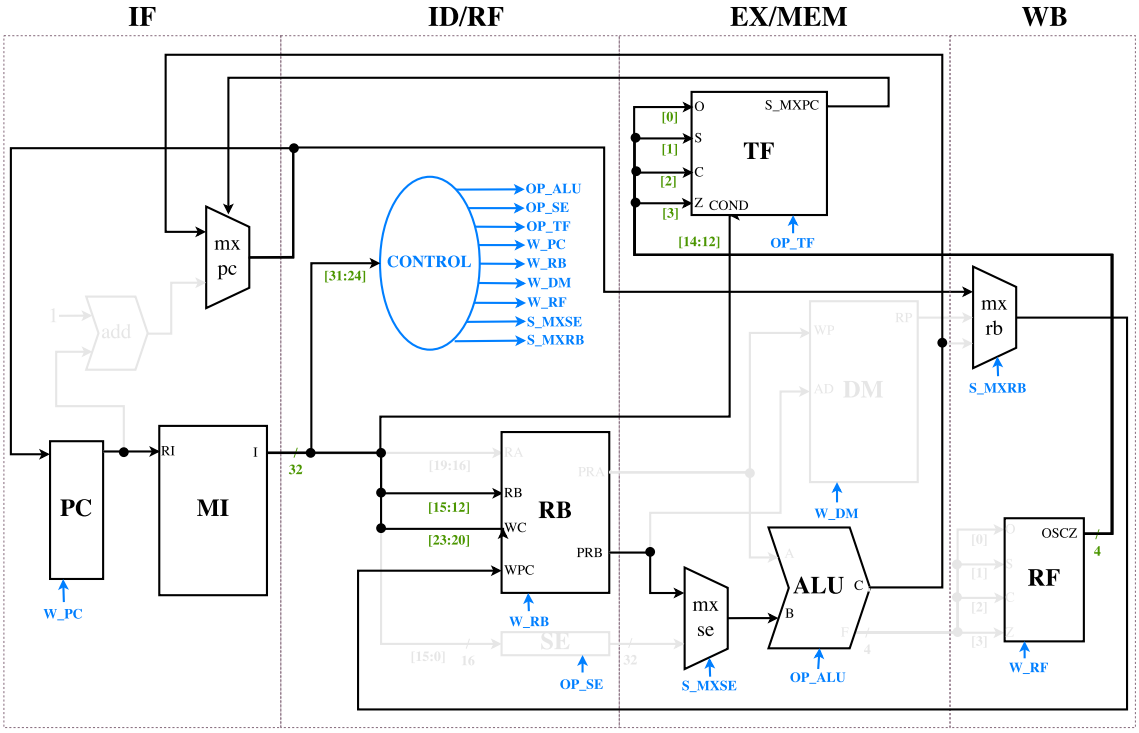

Figura 6: **Datapath Desvio por Registrador**

São duas as instruções de desvio por registrador, jump and link e jump register. Na instrução de jump and link, o valor de PC+1 deve ser armazenado no registrador r15 (definido pela unidade de controle) e o conteúdo do registrador RB armazenado em PC. Já no caso da instrução jump register, a única operação a ser feita é armazenar o conteúdo do registrador RB em PC.

As instruções de desvio por registrador possuem o seguinte formato:

31:29	28:27	26:24	23:16	15:12	11:0
1 1 0	X X	OPDR	X X X X X X X X	RB	X X X X X X X X X X X X

Tabela 15: **Tabela de Formato**

OPDR	Mnemônico	Operação
011	jal b	Jump and Link
100	jr b	Jump Register

Tabela 16: **Tabela de Instruções de Desvio por Registrador**

4.6. HALT

A instrução HALT representa uma parada no sistema. Nesta instrução é realizado um salto para o endereço atual.

O HALT possui o formato seguinte:

31:29	28:27	26:24	23:12	11:0
1 0 1	X X	00	X X X X X X X X X X X X	L

Tabela 17: Tabela de Formato

Mnemônico	Operação
L: j L	PC = PC

Tabela 18: Tabela de Instrução HALT

4.7. NOP

Nessa instrução todos os sinais de controle são zerados, desta forma nada é registrado na memória ou no banco de registradores. A NOP é realizada executando um jump False com a condição TRUE para o endereço 0, resultando no seguinte formato:

31:29	28:0
0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Tabela 19: Tabela de Formato

5. Descrição dos Componentes

Durante algumas discussões foi decidido quais componentes constituiriam o processador. A partir de análises das instruções foram listados os componentes apresentados nas subsessões a seguir:

5.1. Contador de Programa (Program Counter)

O Contador de Programa (PC) é o registrador que armazena o endereço da próxima instrução a ser executada. Ele é atualizado assim que a instrução atual é finalizada.

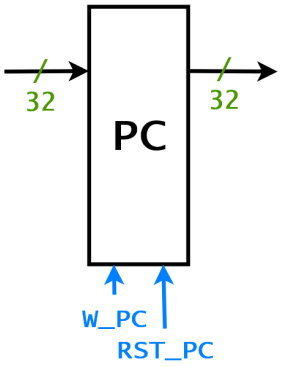

Figura 7: **Contador de Programa**

Nome	Tamanho	Direção	Descrição
in_PC	32	Entrada	Endereço de destino do salto
out_PC	32	Saída	Saída do PC
W_PC	1	Entrada	Sinal de controle do PC
RST_PC	1	Entrada	Sinal que limpa o pc

Tabela 20: **Tabela de Sinais do PC**

5.2. Memória de Instruções

Tem como funcionalidade armazenar todas as instruções do programa a ser executado.

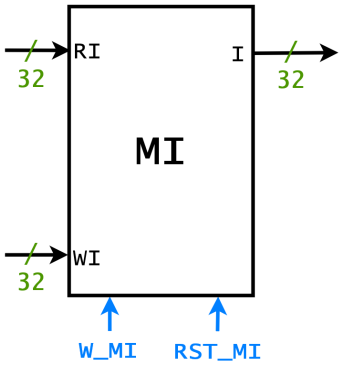

Figura 8: **Memória de Instruções**

Nome	Tamanho	Direção	Descrição
RI	32	Entrada	Endereço da instrução a ser lida
WI	32	Entrada	Endereço de entrada na memória de instrução
I	32	Saída	Instrução atual
W_MI	1	Entrada	Sinal que habilita a leitura
RST_MI	1	Entrada	Sinal que limpa a memória de instruções

Tabela 21: **Tabela de Sinais da Memória de Instruções**

5.3. Banco de Registradores

O banco de registradores consiste em um bloco formado por 16 registradores de propósito geral de 32 bits.

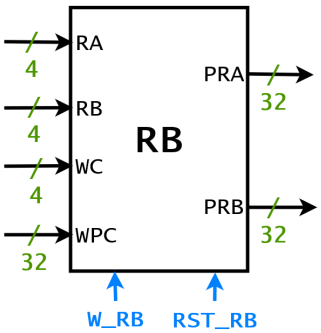

Figura 9: **Banco de Registradores**

Nome	Tamanho	Direção	Descrição
RA	4	Entrada	Endereço do registrador A
RB	4	Entrada	Endereço do registrador B
WC	4	Entrada	Endereço de escrita para o registrador destino
WPC	32	Entrada	Entrada do dado a ser armazenado no registrador destino
PRA	32	Saída	Saída do registrador A
PRB	32	Saída	Saída do registrador B
W_RB	1	Entrada	Sinal que habilita a escrita no banco de resgistradores
RST_RB	1	Entrada	Sinal que limpa a memória de instruções

Tabela 22: Tabela de Sinais do Banco de Registradores

5.4. Extensor de Sinal

O extensor de sinal é utilizado para extender o sinal dos bits de entrada nas operações com constantes e operações de desvio.

Figura 10: Extensor de Sinais

Nome	Tamanho	Direção	Descrição
InSE	16	Entrada	Constante a ser extendida
InSE	12	Entrada	Constante a ser extendida
OutSE	32	Saída	Constante extendida
OP_SE	1	Entrada	Sinal de controle do extensor

Tabela 23: Tabela de Sinais do Extensor de Sinais

5.5. Unidade Lógica e Aritmética (Arithmetic Logic Unit)

A ALU é um circuito combinacional responsável por realizar operações aritméticas e lógicas dentro do processador. As operações a serem executadas são determinadas por

meio dos sinais de controle e das suas entradas de operação, logo após os dados de entrada são computados e o resultado é obtido na saída do circuito.

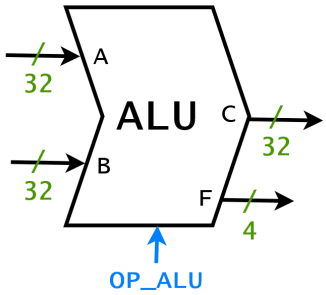

Figura 11: **Unidade Lógica Aritmética**

Nome	Tamanho	Direção	Descrição
A	32	Entrada	Operando A
B	32	Entrada	Operando B
C	32	Saída	Resultado da operação
F	4	Saída	Flags
OP_ALU	5	Entrada	Código de operação da instrução atual

Tabela 24: **Tabela de sinais da ALU**

5.6. Memória de Dados (Data Memory)

A Memória de dados tem como propósito salvar/ler dados proveniente das instruções de acesso à memória.

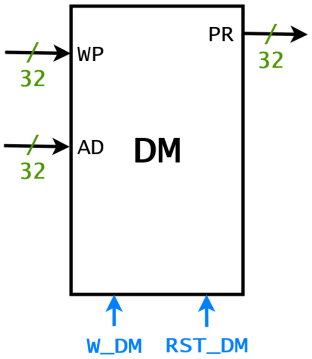

Figura 12: **Memória de Dados**

Nome	Tamanho	Direção	Descrição
WP	32	Entrada	Dado as ser armazenado na memória
AD	32	Entrada	Endereço onde o dado será armazenado
PR	32	Saída	Dado que sai da memória e é escrito no registrador
W_DM	1	Entrada	Sinal que habilita a escrita na memória de dados
RST_DM	1	Entrada	Sinal que limpa a memória de dados

Tabela 25: **Tabela de Sinais da Memória de Dados**

5.7. Registrador de Flags

Responsável por armazenar os estados das flags Overflow, Carry, Sinal e Zero. Estes estados são atualizados de acordo com o tipos e resultados das operações realizadas pela unidade lógica e aritmética.

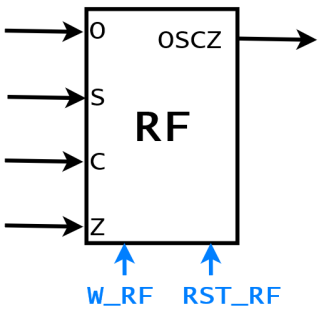

Figura 13: **Registrador de Flags**

Nome	Tamanho	Direção	Descrição
O	1	Entrada	Flag de overflow
S	1	Entrada	Flag de sinal
C	1	Entrada	Flag de carry
Z	1	Entrada	Flag de zero
OSCZ	4	Saída	Flags atualizadas
W_RF	3	Entrada	Sinal que define quais flags serão atualizada
RST_RF	1	Entrada	Limpa os dados do registrador

Tabela 26: **Tabela de Sinais do Registrador de Flags**

5.8. Testador de Flags

Esse módulo combinacional foi criado com o objetivo de decidir se um jump será ou não realizado conforme a condição (COND).O testador verifica se uma determinada flag esta no estado desejado.

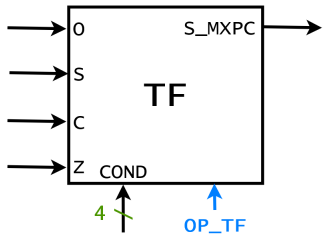

Figura 14: **Testador de Flags**

Nome	Tamanho	Direção	Descrição
O	1	Entrada	Flag de overflow a ser testada
S	1	Entrada	Flag de sinal a ser testada
C	1	Entrada	Flag de carry a ser testada
Z	1	Entrada	Flag de zero a ser testada
COND	3	Entrada	Condição que será testada
S_MXPC	1	Saída	Sinal que indica se o salto será ou não realizado
OP_TF	1	Entrada	Sinal de controle do testador de flags

Tabela 27: **Tabela de Sinais do Testador de Flags**

5.9. Unidade de Controle

Esta unidade é responsável por gerar todos os sinais que controlam o fluxo das tarefas dentro do processador, sinais estes que permitem a leitura ou escrita em registradores, controle de barramento por meio de multiplexadores e códigos de operação para módulos combinacionais.

A unidade de controle é implementada no modelo hardwired, que tem como sua característica principal uma máquina de estado finita (FSM), que seus estados são definidos pelos estágios do processador (IF,ID,EX/MEM,WB). A cada estado todos os sinais de controle são redefinidos afim de controla todos os elementos para realizar a tarefa desejada.

Esse modelo de implementação tem melhor desempenho em relação ao modelo microprogramada, pois a microprogramada é composta por uma memoria de microprograma, o que lhe da a capacidade de reprograma-la sem precisar reestruturar o circuito. Por outro lado projetar uma unidade de controle hardwired é mais dificil que microprogramada. Porém em geral no modelo RISC, como formato de instrução é mais simples, utiliza-se o modelo hardwired.[

Nome	Tamanho	Direção	Descrição
TYPE_OP	8	Entrada	Instrução atual
OP_ALU	5	Saída	Código de operação da instrução atual
OP_SE	1	Saída	Sinal de controle do extensor de sinais. Sendo 0 extende de 12 para 32, sendo 1 extende de 16 para 32
OP_TF	3	Saída	Sinal que indica se o jump é true ou false
W_PC	1	Saída	Sinal de controle do PC
W_RB	1	Saída	Sinal que habilita a escrita no banco de registradores
W_DM	1	Saída	Sinal que habilita a escrita na memória de dados
W_RF	1	Saída	Sinal que habilita o registrador de flags
continua na próxima página			

continuação da tabela anterior			
Nome	**Tamanho**	**Direção**	**Descrição**
S_MXSE	1	Saída	Sinal que controla o multiplexador MXSE. Sendo 1, habilita para a saída do mux o valor de saída do SE, sendo 0, habilita para a saída do mux o valor do registrador B do banco de registradores.
S_MXRB	2	Saída	Sinal que controla o multiplexador MXRB. Sendo 11, habilita para a saída do mux o valor da saída da ULA, sendo 10, habilita para a saída do mux a saída da memória de dados e sendo 01, habilita para a saída do mux o valor de PC.

Tabela 28: Tabela de Sinais da Unidade de Controle

6. Assembly

Para a codificação dos programas deve ser utilizada a linguagem Assembly. Através dos mnemônicos já descritos anteriormente, serão escritas as instruções a serem executadas. Em seguida, através de um programa montador, o código fonte do programa será traduzido em linguagem de máquina para um arquivo binário que poderá ser entendido pelo processador e executado.

Além das instruções, o código fonte deve/pode conter algumas diretivas importantes, são elas:

- .module NOME - Indica o início do programa com o nome informado;
- .pseg - Indica o segmento de programa, ou seja, a partir desse ponto devem conter as instruções a serem executadas. Esta diretiva é encerrada após a ocorrência de uma das diretivas seguintes;
- .dseg - Indica o segmento de dados que deve ser usado para declaração de variáveis globais. Esta diretiva não é obrigatória.
- .end - Indica ao montador o fim do programa. Assim, qualquer instrução subsequente será desconsiderada.

A qualquer momento no código podem ser inseridos comentários. Para isto deve-se inserir o caractere ";"(ponto e vírgula), indicando que a partir dalí todo o restante da linha é comentário e será desconsiderado pelo montador.

Outro ponto importante a ser ressaltado é os labels, usados para indicar um destino de desvios ou uma nova variável. Estes labels devem conter apenas caracteres alfanuméricos. E, para os labels de desvios, devem estar na mesma linha da próxima instrução, para que esta, possa ser interpretada pelo montador.