Documento de Arquitetura

µRisc

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
30/01/2016	Revisão das tabelas de instruções	Matheus Borges
31/01/2016	Revisão da formatação do documento	Matheus Borges
08/04/2016	Alterações dos datapath para pipeline	Matheus Borges
11/04/2016	Alterações das descrições dos módulos para pipeline	Fábio Barros
11/04/2016	Revisão das descrições de funcionamento	Matheus Borges

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 com sucesso a partir dele. 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
IM	Instruction Memory (Memória de Instruções)
PC	Program Counter (Contador de Programa)
RB	Register Bank (Banco de Registradores)
SG	Signal Generator (Gerador de Sinal)
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)
FU	Forward Unit (Unidade de encaminhamento)
RISC	Reduced Instruction Set Computer (Computador com um Conjunto Reduzido de Instruções)

Tabela 2: Tabela de acrônimos e abreviações

2. 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.

Cada instrução necessita de três ciclos de clock para serem finalizadas. Porém, devido a arquitetura ser baseada no princípio de pipeline, o processador consegue executar três instruções simultaneamente. Com isso, esta arquitetura possui o valor 1 como média de instruções por ciclo. A arquitetura foi dividida em 3 estágios, são eles: IF/ID/RF, EX/MEM e WB, apresentados na Figura 1. No primeiro estágio, a próxima instrução a ser executada, definida pelo contador de programa (PC), é lida da memória de instruções (IM), decodificada e os operandos são lidos do banco de registradores ou, em caso de operações com constantes, manipulados pelo extensor de sinal (SE). Ainda no primeiro estágio são tratados os desvios do fluxo de execução, quando as operações forem deste tipo. No segundo estágio, a instrução é executa pela unidade lógica aritmética (ALU) ou pela memória, em caso de operações de armazenamento ou leitura em memória. No terceiro e último estágio (WB) os resultados são escritos no banco de registradores (quando se aplicar).

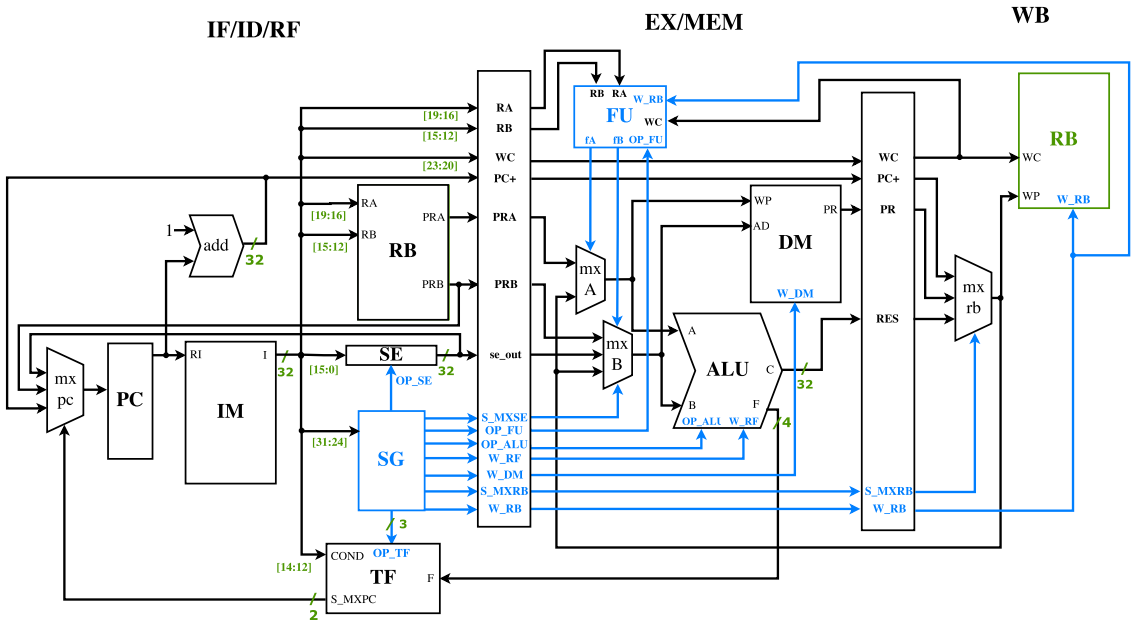

Figura 1: **Datapath Geral**

OBS: O banco de registradores (RB) presente no último estágio é uma representação do mesmo presente no primeiro, utilizado apenas para facilitar a visualização e compreensão do caminho dos dados.

2.1. 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;**
- **Unidade de controle (Gerador de Sinais) hardwired;**
- **Funcionamento em pipeline com três estágios**
- **Detecção e tratamento de conflito de dados;**
- **Execução de três instruções por ciclo de clock;**
- **Sem atrasos devido a conflitos;**
- **Armazenamento em memória de forma big-endian;**
- **Endereçamento por registrador e imediato;**
- **Montador desenvolvido na linguagem C;**
- **Processador descrito utilizando a linguagem Verilog.**

3. 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;**

3.1. Instruções Lógicas e Aritméticas

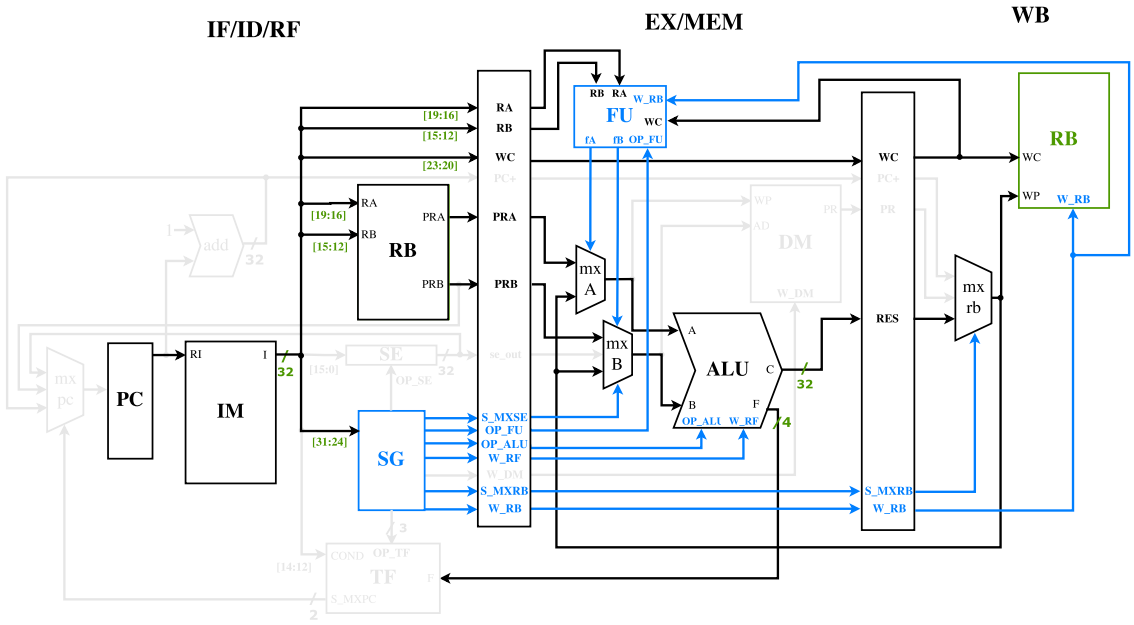

Figura 2: **Datapath de Lógicas Aritméticas**

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 gerar e armazenar os sinais de flags;

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_ALU	WC	RA	RB	X X X X X X X X X X X X

Tabela 3: **Tabela de Formato**

OP_ALU	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	
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
11000	nand c, a, b	C = !A&!B	S Z
11001	xnor c, a, b	C = !(AˆB)	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	

3.2. Instruções com Constante

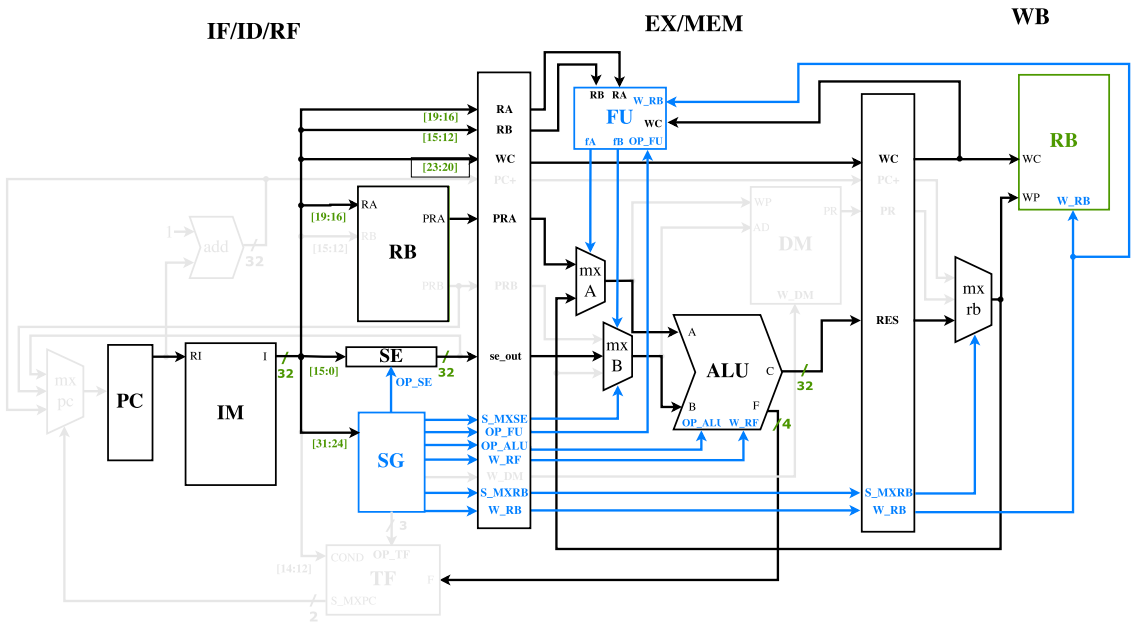

Figura 3: **Datapath de Constantes**

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_ALU	WC	X X X X	CONSTANTE

Tabela 5: **Tabela de Formato**

OP_ALU	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 6: **Tabela de Instruções com Constante**

3.3. Instruções de Acesso à Memória

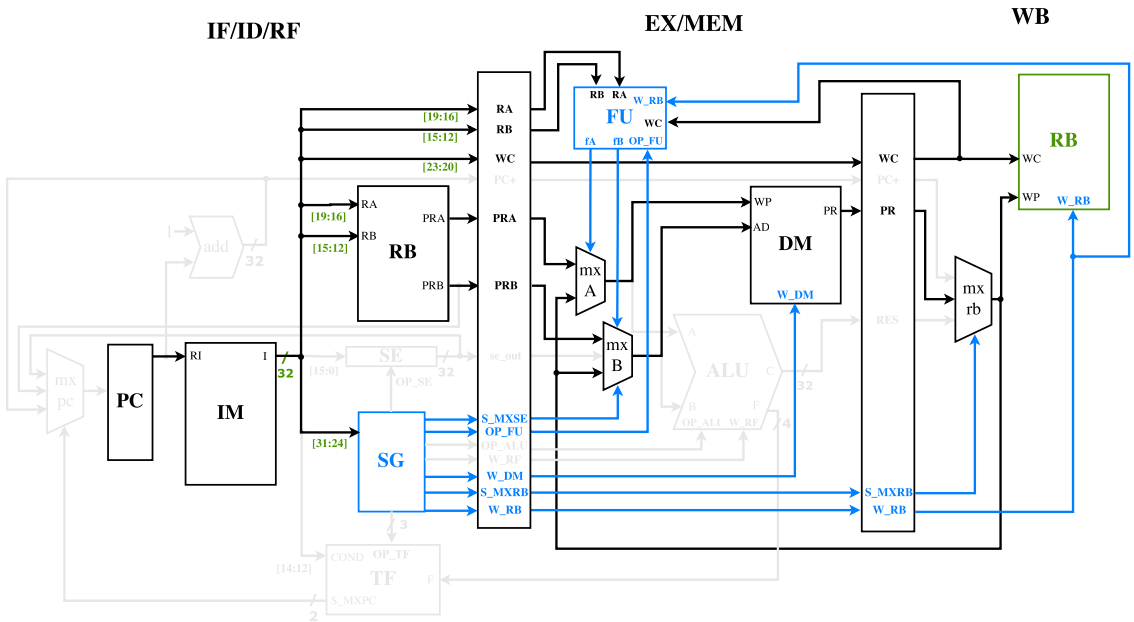

Figura 4: **Datapath Memória**

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 WC. 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. A memória de dados realiza a leitura constantemente. Já a escrita apenas é realizada quando o sinal W_DM é mandado em nível lógico alto. 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	W_DM	WC	RA	RB	X X X X X X X X X X X X

Tabela 7: **Tabela de Formato**

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

Tabela 8: **Tabela de Instruções de Acesso à Memória**

3.4. Instruções de Desvio

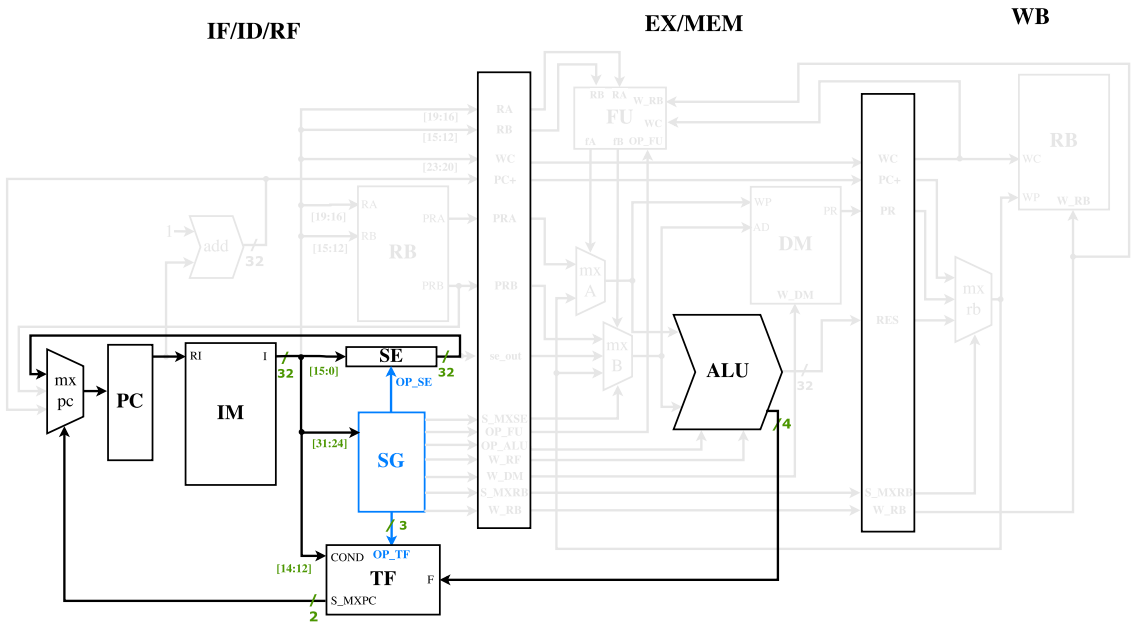

Figura 5: **Datapath Desvio Condicional**

O módulo que caracteriza essa instrução é o testador de flags (TF) que, a partir de uma condição, determina qual será o próximo valor do PC. Este valor pode ser o PC + 1 (fluxo normal), a saída do SE (instruções jf, jt e j) ou a saída PRB do banco de registradores (instruções jal e jr, especificados na sessão 4.5)

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	OP_TF	X X X X X X X X X	COND	DESTINO

Tabela 9: **Tabela de Formato**

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

Tabela 10: 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 11: Tabela de Condições

3.5. Instruções de Desvio por Registrador

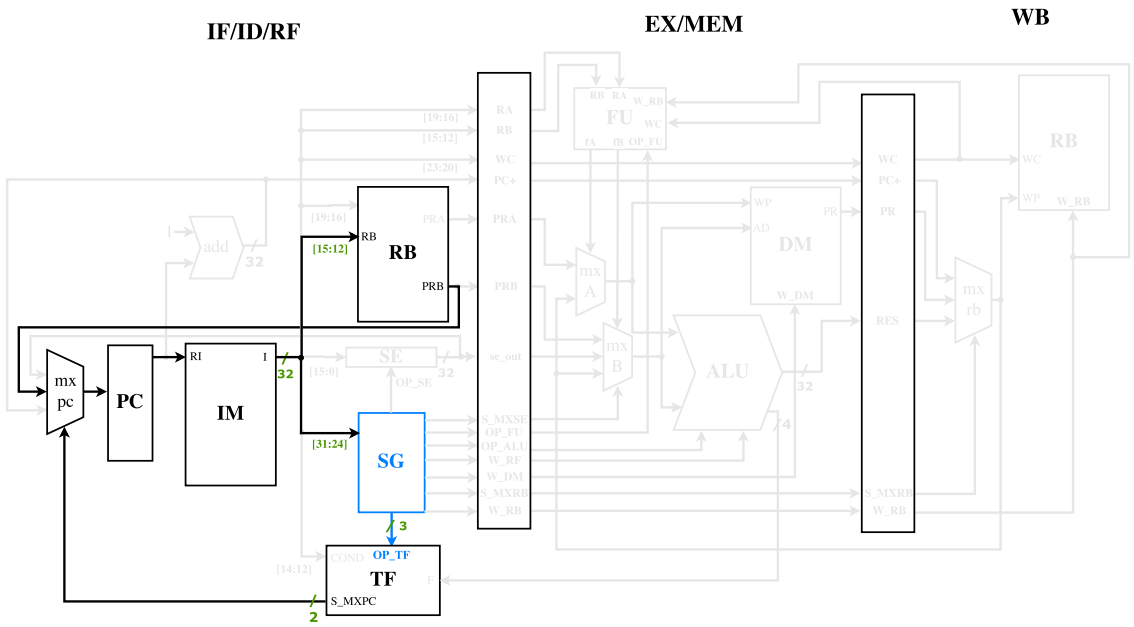

Figura 6: **Datapath Desvio por Registrador**

São duas as instruções de desvio por registrador, jump and link (jal) e jump register (jr). Na instrução de jump and link, o valor de PC+1 deve ser armazenado no registrador r15 (definido pelo compilador) e o conteúdo do registrador RB armazenado em PC. Já no caso da instrução jump register, a única operação a ser feita é passar o conteúdo do registrador RB para o 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	OP_TF	X X X X X X X X	RB	X X X X X X X X X X X X

Tabela 12: **Tabela de Formato**

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

Tabela 13: **Tabela de Instruções de Desvio por Registrador**

3.6. 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	010	X X X X X X X X X X X X	DESTINO ATUAL

Tabela 14: Tabela de Formato

Mnemônico	Operação
L: j L	PC = PC

Tabela 15: Tabela de Instrução HALT

3.7. 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 16: Tabela de Formato

4. 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:

4.1. 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 a cada borda ascendente de clock.

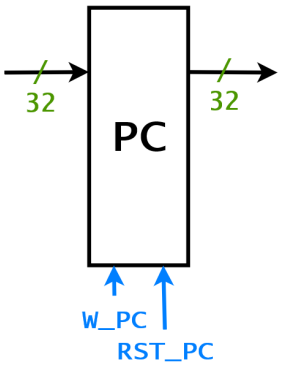

Figura 7: **Contador de Programa**

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

Tabela 17: **Tabela de Sinais do PC**

4.2. Memória de Instruções

Tem como funcionalidade armazenar todas as instruções do programa a ser executado. A leitura da instrução acontece sempre.

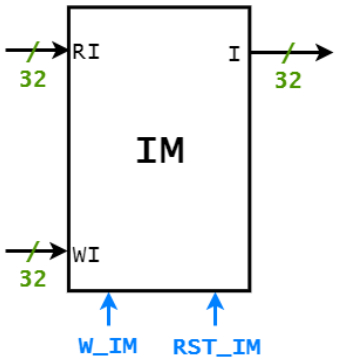

Figura 8: **Memória de Instruções**

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_IM	1	Entrada	Sinal que habilita a escrita
RST_IM	1	Entrada	Sinal que limpa a memória de instruções

Tabela 18: **Tabela de Sinais da Memória de Instruções**

4.3. Banco de Registradores

O banco de registradores consiste em um bloco formado por 16 registradores de propósito geral de 32 bits. Sua função é registrar os dados utilizado por um programa. A entrada que controla escrita é ligada ao clock do processador, e a cada borda descendente de clock, a escrita é realizada, caso o sinal W_RB esteja em nível lógico alto.

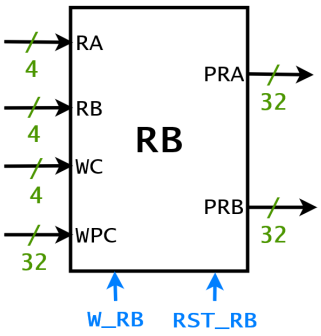

Figura 9: **Banco de Registradores**

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 19: Tabela de Sinais do Banco de Registradores

4.4. Extensor de Sinal

O extensor de sinal é utilizado para estender o sinal dos bits de entrada nas operações com constantes e operações de desvio, afim de padronizar os dados que são operados em 32 bits.

Figura 10: Extensor de Sinais

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

Tabela 20: Tabela de Sinais do Extensor de Sinais

4.5. 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. Além disso é computado as flags utilizada nas comparações das condições de salto junto ao módulo testador de flags.

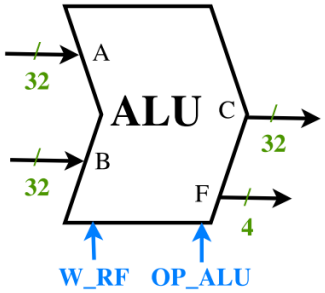

Figura 11: **Unidade Lógica Aritmética**

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
W_RF	3	Entrada	Define quais flags serão armazenadas

Tabela 21: **Tabela de sinais da ALU**

4.6. Memória de Dados (Data Memory)

A Memória de dados tem como propósito salvar/ler dados proveniente das instruções de acesso à memória. A leitura dessa memoria funciona da mesma maneira que a Memória de Instrução.

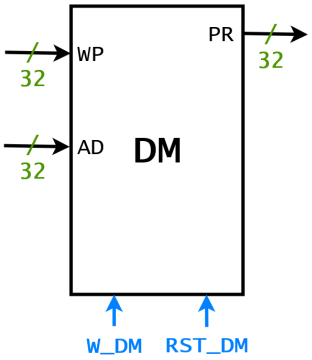

Figura 12: **Memória de Dados**

Nome	Tamanho	Direção	Descrição
WP	32	Entrada	Dado as ser armazenado na memória
AD	32	Entrada	Endereço para armazenamento
PR	32	Saída	Dado lido da memória de dados
W_DM	1	Entrada	Sinal de escrita na memória de dados
RST_DM	1	Entrada	Sinal que limpa a memória de dados

Tabela 22: **Tabela de Sinais da Memória de Dados**

4.7. Gerador de Sinais

Responsável por gerar todos sinais de controle necessários para executar a instrução atual. Esses sinais são responsável por definir o fluxo dos dados por todo o processador, garantindo que a instrução atual não acesse módulos não desejados.

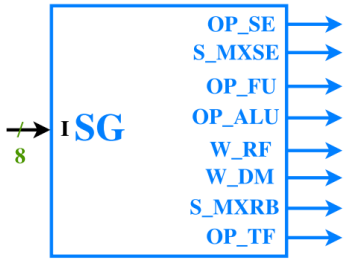

Figura 13: **Gerador de Sinais**

Nome	Tamanho	Direção	Descrição
I	8	Entrada	Define qual tipo de instrução
OP_SE	1	Saída	Código de operação que indica qual extensão ira ser realizada
OP_FU	2	Saída	Sinal que define se é necessário utilizar o adiantamento de dados para o operador A ou B das operações da ALU e memória
S_MXSE	1	Saída	Seletor do multiplexador (mx_b) que seleciona a origem do operador B das operações da ALU e memória
OP_ALU	5	Saída	Código da operação a ser realizada na ALU
continua na próxima página			

continuação da tabela anterior			
Nome	**Tamanho**	**Direção**	**Descrição**
W_RF	1	Saída	Sinal que define quais flags serão armazenadas
W_DM	1	Saída	Sinal que indica uma escrita na Memória de Dados
S_MXRB	2	Saída	Seletor do multiplexador do RB que indica qual a origem do dado a ser armazenado
OP_TF	3	Saída	Sinal que indica qual é o tipo de comparação será feita para condição de pulo

Tabela 23: **Tabela de Sinais do Gerador de Sinais**

4.8. Unidade de Encaminhamento

Responsável por adiantar dados necessários para a operação atual, porém ainda não salvos no banco de registradores, evitando assim o conflito de dados. Este módulo verifica se:

- é necessário checar o adiantamento de dados (através do sinal OP_FU. Sendo que 00 ele não realiza a checagem, 01 verifica apenas RB, 10 apenas RA e 11 verifica os dois endereços;
- a operação anterior escreverá no banco de registradores (através do sinal W_RB);
- o registrador indicado por WC (da instrução anterior) é igual a RA, ou RB.

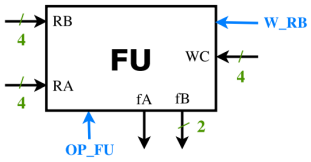

Figura 14: **Unidade de Encaminhamento**

Nome	Tamanho	Direção	Descrição
RA	4	Entrada	Registrador A da instrução atual
continua na próxima página			

continuação da tabela anterior			
Nome	**Tamanho**	**Direção**	**Descrição**
RB	4	Entrada	Registrador B da instrução atual
WC	4	Entrada	Endereço a ser escrito da instrução anterior
OP_FU	2	Entrada	Sinal que define se é necessário verificar o operador A, ou o operador B
W_RB	1	Entrada	Sinal da instrução anterior que indica se ela escreverá no banco de registradores
fa	1	Saída	Sinal que indica o adiantamento do operador A
fb	1	Saída	Sinal que indica o adiantamento do operador B

Tabela 24: Tabela de Sinais da Unidade de Encaminhamento

4.9. Testador de Flags

Esse módulo combinacional foi criado com o objetivo de decidir se um jump será ou não realizado conforme a condição (COND) é o sinal vindo da unidade de controle. A partir do sinal COND, este módulo decide se o teste será feito utilizando a flag O(111), S(001), C(100), Z(010), S | Z(101), ou ainda, se o teste será feito com a condição true(000). A saída deste módulo é ligada diretamente ao multiplexador do pc, que seleciona se o próximo valor do PC virá do extensor de sinal (saída 00 do testador, indicando que um desvio por endereço de memória), do banco de registradores (saída 01 do testador, indicando um desvio por registrador) ou ainda, do somador do PC (saída 10 do testador, indicando que deve seguir o fluxo normal de execução).

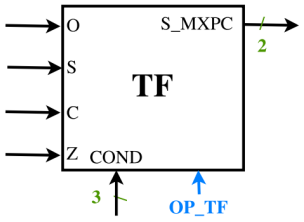

Figura 15: Testador de Flags

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	3	Entrada	Sinal de controle do testador de flags

Tabela 25: Tabela de Sinais do Testador de Flags

5. 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.