Documento de Arquitetura

Projeto ANTARES-R2

Universidade Estadual de Feira de Santana

Build 1.0

Histórico de Revisões

Date	Descrição	Autor(s)
13/08/2016	• Criação do Documento; • Requisitos mínimos; • Visão geral da arquitetura;	Arthur Hagnês de Jesus Ferreira
14/08/2016	• Montador Montador	Arthur Hagnês de Jesus Ferreira
14/08/2016	Simulador	Marcus José
xx/xx/xxxx	• Exemplo de; • Revisões em lista;	<Autor(es)>

SUMÁRIO

1 | Introdução

1. Propósito do Documento

Este documento descreve a arquitetura de instruções do Projeto ANTARES-R2, incluindo especificações do circuitos internos e cada componente. Ele também apresenta diagramas de classe. Além das caracteristicas basicas do projeto.O principal objetivo deste documento é definir as especificações do projeto Projeto ANTARES-R2e prover uma visão geral completa do mesmo.

2. Stakeholders

Nome	Papel/Responsabilidades
Arthur Hagnês de Jesus Ferreira	Desenvolvedor
Marcus José	Desenvolvedor

3. Visão Geral do Documento

O presente documento é apresentado como segue:

- **Capítulo 2** – Este capítulo aprensenta os requisitos mínimos para o projeto;
- **Capítulo 3** – Este capítulo apresenta uma visão geral da arquitetura, com o foco na descrição dos componentes;
- **Capítulo 4** – Este capítulo apresenta uma visão geral do Montador e Simulador para este processador;

4. Acrônimos e Abreviações

Sigla	Descrição
MIPS	Micropocessor without Interlocking Pipes Stages
RISC	Reduced Instruction Set Computer
Instruções R	Instruções do tipo Registrador
Instruções I	Instruções do tipo Imediato
Instruções J	Instruções do tipo Jump
ISA	Instruction Set Architecture
ULA	Unidade Lógico - Aritmética
UC	Unidade de COntrole
GPR	Registrador de Proposito Geral
OPCODE	Codigo de operação
RD	Registrador Destino
RA	Registrador Fonte A
RB	Registrador Fonte B
CONST	Constante

2 | Requisitos do Projeto

O processador desenvolvido deve ser capaz de executar instruções que possibilitam a gerenciar da máquina de forma rápida e eficiente. Além de possuir as seguintes características básicas de um processador.

- **1** – Arquitetura de 32 bits, ou seja, cada palavra de instrução deve conter 32 bits de comprimento;
- **2** – Arquitetura contém 32 GPR de 32 bits de comprimento;
- **3** – Arquitetura contém um ISA com 64 instruções;
- **4** – O Processador deve ser capaz de executar os algoritmos:
  - Geração da Sequência de Fibonacci $Recusivo$
  - Ordenação de um vetor usando o algoritmo Bubble Sort
  - Cálculo do fatorial de um número inteiro $Recursivo$
  - Geração de números primos
  - Algoritmo para cálculo de raiz quadrada de um número inteiro
  - Algoritmo para cálculo da potência x^y;

3 | Visão Geral da Arquitetura

1. Caracteristica da Arquitetura

- Arquitetura tipo RISC.
- 32 GPR de 32 bits.
- As palavras contém o tamanho de 32 bits, logo o endereçamento é consecutivo, diferenciando em uma 1 unidade.
- ISA composta por 64 instruções.
- Organização dos dados na memória do tipo Big Endian , ou seja, o dígito de mais significativo está sempre no início da palavra, mais a esquerda;
- Arquitetura possui 1 flag Zero;

2. Banco de Registradores

Na arquitetura MIPS as operações lógicas ou aritimeticas só podem ser feitas com os valores que esteja nos reg istradores, portanto é necessário que se tenha operações para carregar e armazenar os dados contidos na memória para os respectivos registradores, para só depois ser realziado a operação.
Logo é imprescindível saber como está mapeado o banco de registradores dessa arquitetura:

Valor	Registrador	Descrição
0	$zero	O valor constante 0
1	$at	Usado temporário pelo montador
2-3	$v0 –$v1	Valores para resultado de função e avaliação de expreções
4-7	$a0 –$a3	Parâmetros que serão passado para uma função
8-15	$t0 –$t7	Variaveis temporarias, não precisa ser preservada
16-23	$s0 –$s7	Variaveis de função, deve ser preservada
24-25	$t8 –$t9	Outras variaveis temporarias
26-27	$k0 –$k1	Reservado para uso do Kernel do Sistema Operacional
28	$gp	Global Pointer
29	$sp	Stack Pointer
30	$fp	Frame Pointer
31	$ra	Guarda o endereço da ultima sub-rotina chamada

- **Global Pointer** – Resgistrador responsável por apontar a posição dos dados estaticos;
- **Stack Pointer** – Registrador responsável apontar para a ultima posição ocupada na pilha;
- **Frame Pointer** – Registrador responsável apontar para a primeira palavra indicando o local dos registradores salvos e as variaveis locais de um determinado procedimento;

3. Datapath

A figura abaixo mostra o Datapath completo da arquitetura proposta neste documento. Além de mostrar como o dado é passado por cada parte do processador e os sinais de controle enviado pela UC para a realização das operações.

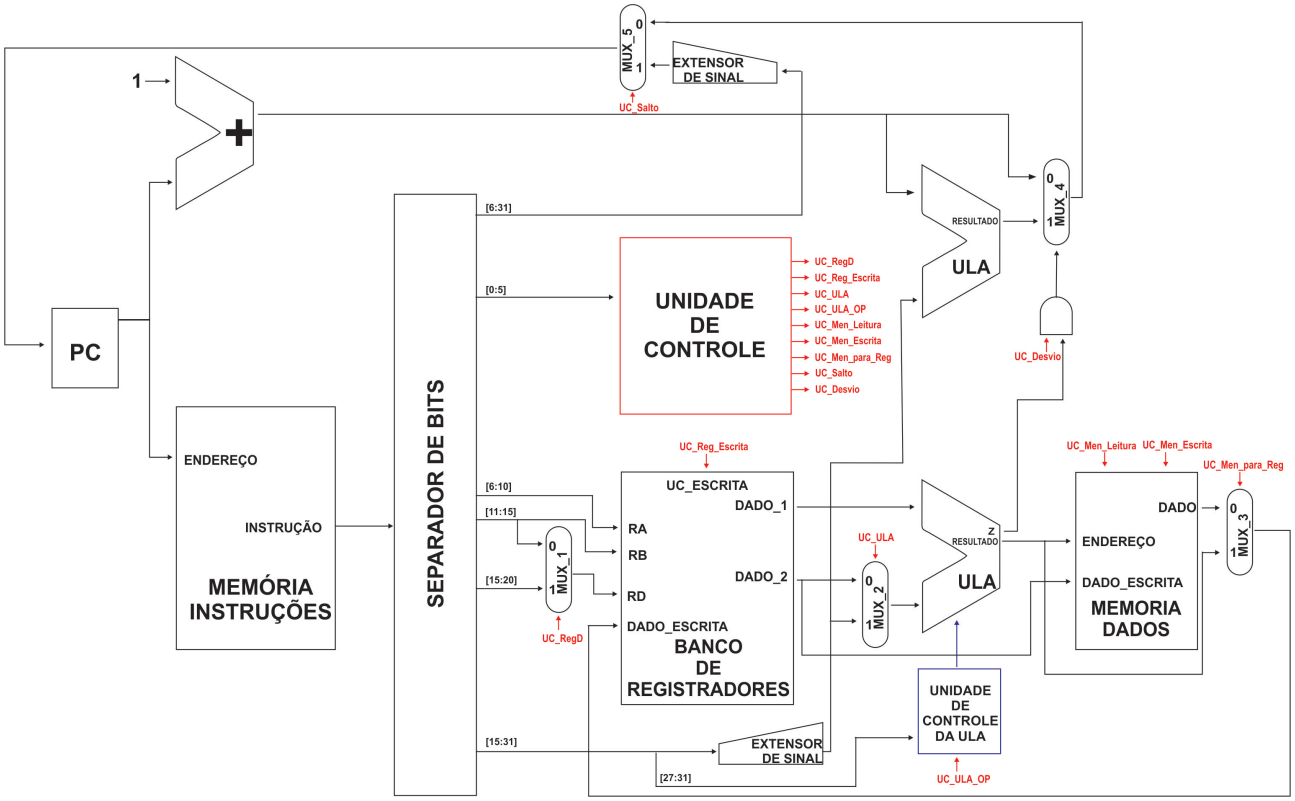

Figura 3.1: **Datapath da Arquitetura**

Sinal	Descrição
UC_RegD	Seleciona qual registror deve ser escrito
UC_Reg_Escrita	Indica ao banco de registrador para escrever no registrador
UC_ULA	Seleciona qual registrador será enviado para ULA
UC_ULA_OP	Ativa a unidade de controle da ULA
UC_Men_Leitura	Inidica a memória de dados que há uma leitura para ser feita
UC_Men_Escrita	Inidica a memória de dados que há uma escrita para ser feita
UC_Men_para_Reg	Seleciona qual dado será escrito no registrador(ula ou Memória)
UC_Salto	Inidica que a instrução é de Salto
UC_Desvio	Inidica que a instrução é de Desvio

4. Componentes do Datapath

Nesta secção detalharemos todos os componentes do datapath, tal como sua estrutura e seu funcionamento.

4.1. Memória Instruções e Memória Dados

A memória é todo dispositivo responsável por armazenar um dado, sendo temporário ou permanente, para nossa arquitetura a memória é responsável por armazenar as palavras de instrução e os dados do programa. Tanto os dados quanto as instruções estão contidas na mesma memória.
Para poder particionar a memória a primeira linha do arquivo binário contém a quantidade de instruções do arquivo, dessa forma é possível saber o tamanho da memória de instrução e o restante para os dados.

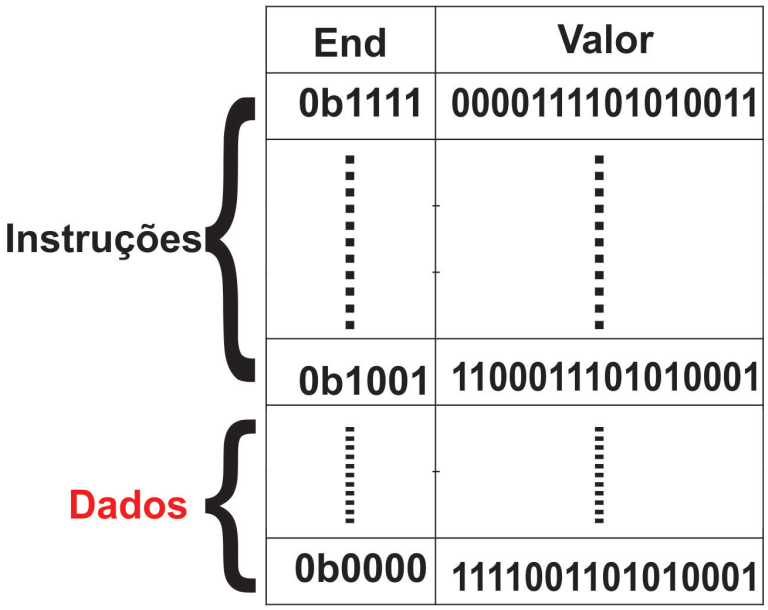

Figura 3.2: **Formato da memória**

4.2. Unidade de Controle

A unidade de controle é responsável por gerar todos os sinais que controlam das operações para os componentes, afim de dar todas as instruções para o correto funcionamento interno do CPU.
Para fazer o gerenciamento dos componentes a unidade de controle precisa decodificar as operações para que os componentes saibam quais funções devem ser ativadas. Como o processador só consegue entender sinais de tensão positivo ou negativo, trazendo para alto nível 1 ou 0. É necessário criar OPCODE para as operações, segue na tabela abaixo os OPCODE.
Porém, antes de aprensentar a tabela de OPCODE é importante aprensentar as estruturas das palavras de instrução para a esta arquitetura.

Instrução R

A instrução do tipo R é composta por 6 campos: Opcode, Function, 3 GPR e SHIFT AMOUNT (deslocamento).

0:5	6:10	11:15	16:20	21:25	26:31
OPCODE	RA	RB	RD	SHIFT AMOUNT	FUNCTION

Tabela 3.1: Instrução R

Instrução I

A instrução do tipo I é composta por 4 campos: Opcode, 2 GPR e Campo para o valor Imediato.

0:5	6:10	11:15	15:31
OPCODE	RA	RB	IMEDIATO

Tabela 3.2: Instrução I

Instrução J

A instrução do tipo J é composta por 2 campos: Opcode, Endereço.

0:5	6:31
OPCODE	Endereçamento

Tabela 3.3: Instrução J

Opcode	Func	Tipo	Mnemônico	Operação
000000	100000	R	ADD $RD, $RA, $RB	$RD = $RA + $RB
000000	100001	R	ADDU $RD, $RA, $RB	$RD = $RA + $RB
011100	100000	R	CLZ $RD, $RA	$RD = COUNTLEADINGZEROS($RA)
011100	100001	R	CLO $RD, $RA	$RD = COUNTLEADINGONES($RA)
000000	100010	R	SUB $RD, $RA, $RB	$RD = $RA - $RB
000000	100011	R	SUBU $RD, $RA, $RB	$RD = $RA - $RB
000000	100100	R	AND $RD, $RA, $RB	$RD = $RA & $RB
000000	100111	R	NOR $RD, $RA, $RB	$RD = ¬($RA | $RB)
000000	100101	R	OR $RD, $RA, $RB	$RD = $RA | $RB
000000	100110	R	XOR $RD, $RA, $RB	$RD = $RA $\hat{}$RB
011111	000000	R	EXT $RB, $RA, pos, size	$rt = EXT($RA, size, pos)
011111	000100	R	INS rt, rs, pos, size	$rt= InsertField($rt, $rs, msb, lsb)
000000	011010	R	DIV $RD, $RA, $RB	$RD = $RA / $RB
000000	011011	R	DIVU $RD, $RA, $RB	$RD = $RA / $RB
011100	000000	R	MADD $RA, $RB	(HI- LO) += $RA * $RB
011100	000001	R	MADDU $RA, $RB	(HI- LO) += $RA * $RB
011100	000100	R	MSUB $RA, $RB	(HI- LO) -= $RA * $RB
011100	000101	R	MSUBU $RA, $RB	(HI- LO) -= $RA * $RB
011100	000010	R	MUL $RD, $RA, $RB	$RD = $RA * $RB
000000	011000	R	MULT $RA, $RB	acc = $RA * $RB
000000	011001	R	MULTU $RA, $RB	acc = $RA * $RB
000000	000000	R	SLL $RD, $RA, c	$RD = $RA « CONST
000000	000100	R	SLLV $RD, $RA, $RB	$RD = $RA«$RB
000000	000010	R	SRL $RD, $RA, c	$RD = $RA » CONST
000000	000011	R	SRA $RD, $RA, c	$RD = $RA » CONST (arithmetic)
000000	000111	R	SRAV $RD, $RA, $RB	$RD = $RB » $RA] (arithmetic)

Tabela 3.4: Instruções da Arquitetura - parte 1

Opcode	Func	Tipo	Mnemônico	Operação
000000	000110	R	SRLV $RD, $RA, $RB	$RD = $RB » $RA
000000	000010	R	ROTR $RD, $RA, sa	$RD = $RA :: sa
000000	000110	R	ROTRV $RD, $RA, $RB	$RD = $RA :: $RB
000000	101010	R	SLT $RD, $RA, $RB	if $RA < $RB, $RD = 1 (else $RD = 0)
000000	101011	R	SLTU $RD, $RA, $RB	if $RA < $RB, $RD = 1 (else $RD = 0)
000000	001011	R	MOVN $RD, $RA, $RB	if $RB != 0, $RD = $RA
000000	001010	R	MOVZ $RD, $RA, $RB	if $RB == 0, $RD = $RA
000000	010000	R	MFHI $RD	$RD = HI
000000	010010	R	MFLO $RD	$RD = LO
000000	010001	R	MTHI $RA	HI = $RA
000000	010011	R	MTLO $RA	LO = $RA
000000	001000	R	JR $ra	go to $ra(Endereço)
000000	001001	R	JALR $RD,$R A	$RD = Endereço de retorno; pc=$RA
011111	100000	R (special3)	SEB $RD, $RB	$RD = SignExtend($RB)
011111	100000	R (special3)	SEH $RD, $RB	$RD = SignExtend($RB)
011111	100000	R (special3)	WSBH $RD, $RB	$RD = WSBH($RB)
000010	——	J	J L	go to L
000011	——	J	JAL L	$ra=PC + 1, go to L
001000	——	I	ADDI $RD, $RA, CONST	$RD = $RA + CONST
001001	——	I	ADDIU $RD, $RA, CONST	$RD = $RA + CONST
001100	——	I	ANDI $RD,$RA,CONST	$RD=$RA and CONST
001101	——	I	ORI $RD, $RA, CONST	$RD=$RA | CONST
001110	——	I	XORI $RD, $RA, CONST	$RD=$RA ĈONST
001010	——	I	SLTI $RD, $RA, CONST	if $RA < CONST, $RD =1 (else $RD = 0)
001011	——	I	SLTIU $RD, $RA, CONST	if $RA < CONST, $RD = 1 (else $RD = 0)
000100	——	I	BEQ $RA, $RB, L	if $RA == $RB, go to L
000111	——	I	BGTZ $RA, offset	if $RA>0, pule

Tabela 3.5: Instruções da Arquitetura - parte 2

Opcode	Func	Tipo	Mnemônico	Operação
000101	——	I	BNE $RA, $RB, L	if $RA != $RB, go to L
000001	——	I	BLTZ $RA,offset	if GPR[rs] < 0, pule
100000	——	I	LB $RB, CONST($RA)	$RD = offset(base)
100011	——	I	LW $RB, CONST($RA)	$RD = offset(base)
100001	——	I	LH $RB, CONST($RA)	$RD = offset(base)
101000	——	I	SB $RB, CONST($RA)	memory[base+offset] = $RD
101001	——	I	SH $RB, CONST($RA)	memory[base+offset] = $RD
101011	——	I	SW $RB, CONST($RA)	memory[base+offset] = $RD

Tabela 3.6: **Instruções da Arquitetura - parte 2**

4.3. PC - Contador de Programa

Contador de Programa, é um registrador de 32 bits que guarda o endereço de memória da próxima instrução e envia para a memória de instruções.

4.4. Somador

Este componente é responsável por receber o valor do PC e somar mais 1, gerando o endereço da próxima instrução.

4.5. Extensor de Sinal

Componente que recebe um operando e o transforma em 32 bits, o bit mais significativo é replicado até completar os 32 bits.

4.6. Banco de Registradores

Banco com 32 registradores que guarda os valores das variáveis usadas no programa que está a executar. Para serem acessados é necessário o endereço dos mesmos.

4.7. Mux

Multiplexador que recebe duas entradas e repassa na sua saída apenas uma delas, definida pelo sinal de controle.

4.8. ULA

Unidade lógica e aritmética, responsável pelas operações feitas pelo processador. A ULA desta arquitetura opera com duas palavras, o tipo de operação que é um sinal enviado pela unidade de controle e realiza a operação. A sua saída é de 32 bits. Algumas operações da ULA atualiza uma ou mais Flags de acordo o resultado.

4 | Montador

1. Defínção:

O montador é responsável por fazer a tradução do código fonte em assembly para a linguagem de maquina aceita por este processador. O montador faz uso da tabela de instruções para poder traduzir todas as instruções conforme o OPCODE e outros campos definos pela arquitetura.

Para que a tradução seja feita corretamente é necessário que o código fonte esteja conforme o especificado nos Mnemónicos na tabela de instruções, porque só assim será obtido o resultado esperado. Além disso é importante observar o uso das diretivas para que o código seja traduzido corretamente. O montador não é sensitive case, para manter um padrão é aconselhavel que as isntrução sejam escritas em caixa alta e as variavels sejam em caixa baixa. Por exemplo; ADD $s0,$t1,$s2

O montador aprensentado nesta secção possui 4 diretivas basicas que irão auxiliar ao programador como escrever o código assembly para o processador prosposto. As diretivas são:

.module nome do programa

Essa diretiva é responsável somente para o programador definir o nome do seu programa.

.end

Essa diretiva serve para definir o final do seu programa assembly.

.pseg

Essa diretiva serve para definir o inicio das instruções do seu programa assembly.

.dseg

Essa diretiva serve para definir o inicio dos dados estáticos seu programa.

.word

Essa diretiva serve para definir um dado estático do seu programa.
Para definir uma variável é necessário definir o valor apois a label, por exemplo:
variavel_X: . word 10

Para definir um vetor é necessário definir os valores abaixo da label, segue o exemplo abaixo.
Exemplo:
vetor_X:
. word 10
. word 10

Labels

Para criar uma label é necessário atentar para as seguintes informações. O montador não aceita duas labels com mesmo nome, labels devem conter depois pontos no final e a instrução seguinte ser escrita na próxima linha.
Exemplo:
Label_x:
ADD $t0,$t5,$t0

1.1. Estrutura do Código

O código fonte assembly deve conter a estrutura apresentada abaixo. Desta forma o montador conseguirá traduzir o código fonte para linguagem de máquina.

```
.module nome_programa

    .pseg

    INSTRUÇÕES E LABELS

    .dseg

    var: .word Inteiro
    var_verto:
    .word Inteiro
    .word Inteiro

.end
```

Figura 4.1: Estrutura do Código

2. Simulador