Documento de Arquitetura

Microprocessador Antares-R2

Universidade Estadual de Feira de Santana - UEFS

Versão 1.0

Histórico de Revisões

Data	Descrição	Autor(es)
13/08/2016	• Concepção • Propósito Geral	Khaíck Oliveira Brito, Cássio Silva e Wanderson Silva
14/08/2016	• Explicação da memória • Listagem das instruções • Explicação do Montador • Explicação do Simulador	Khaíck Oliveira Brito, Cássio Silva e Wanderson Silva

SUMÁRIO

1. Introdução

1.1. Propósito do Documento

O objetivo principal deste documento é definir a especificação da implementação, em nível de instrução do Microprocessador Antares-R2. Neste documento, está contido o conjunto de instruções para implementação, as caracteristicas dos registradores bem como os requisitos para a implementação do projeto.

1.2. Descrição do documento

O presente documento é apresentado como segue:

- **Seção 2** –Esta seção apresenta uma visão geral da arquitetura do core Microprocessador Antares-R2, descrevendo as principais características do mesmo
- **Seção 3** –Esta seção apresenta informações quanto a memória utilizada e descreve os registradores presentes na arquiteura
- **Seção 4** –Esta seção diz respeito a listagem das instruções aceitas pelo Antares-R2, bem como os tipos e os formatos das instruçõse aceitas
- **Seção 5** –: Esta seção provê informações sobre o montador desenvolvido para o ANTARES-R2.
- **Seção 6** –Esta seção provê informações sobre o simulador desenvolvido para o ANTARES-R2, capaz de executar os códigos binários gerados pelo seu Montador, e simular o funcionamento do .

1.3. Acrônimos e Abreviações

Acrônimo	Descrição
GRP	Registrador de Proposito Geral
Const	Constante
Imm	Número imediato (constante)
ALU	Unidade Lógica Aritmetica
Shamt	Shift Ammount

Tabela 1: Acrônimos e Abreviações

2. Visão geral do projeto

O Microprocessador Antares-R2 é um *core* de 32-bits projetado para uma arquitetura RISC e baseado no MIPS. Este core apresenta um conjunto de 57 instruções e o seu

Montador suporta outras 7 pseudo-instruções. O Antares-R2 possui um banco de 32 registradores e é estruturado para ser sistema orientado a palavras de 32-bts tanto em sua memória quanto em seus registradores.

O Antares-R2 baseia-se na existência de uma memória global de 64Kb para ser segmentada de acordo com propositos especificos, como será explicado nas próximas sessões.

3. Memória

3.1. Banco de Registradores

O Microprocessador Antares-R2 considera a posse de um banco de 32 registradores, sendo que registrador 0 sempre contém zero e pode ser usado como um operando sempre que um zero for necessitado.

A tabela a seguir apresenta a listagem de todos os registradores presentes na arquitetura.

Número	Nome	Descrição
0	$zero	Fixo com valor zero
1	$at	(Assembler Temporary) Usado em pseudo-instruções
2,3	$v0, $v1	Retorno de valores de funções
4-7	$a0 - $a3	Passagem de parâmetros de funções
8-15	$t0 - $t7	Dados temporários
16-23	$s0 - $s7	Dados Salvos
24,25	$t8, $t9	Dados temporários
26-27	$k0, $k1	byte da memória
28	$gp	Ponteiro Global, aponta para inicio do segmento de dados
29	$sp	Ponteiro da Pilha, aponta para o topo da pilha
30	$fp	Ponteiro de Quadro, aponta para o inicio da pilha de um procedimento
31	$ra	Armazena o endereço de retorno

Tabela 2: Banco de registradores

3.2. Memória Compartilhada

O Microprocessador Antares-R2 considera a existência de uma memória compartilhada de 64Kb endereçada em bytes. Cada palavra na arquitetura deste *core* é baseado em um sistema de 32-bits.

Esta memória é utilizada para endereçar tanto as instruções interpretadas quanto os dados armazenados. Para isso, a memória é segmentada seguindo a proporção de 1:1:2 para respetivamente as instruções, dados estáticos e a pilha.

Fazendo estas considerações, a memória compartilhada deve suportar um total de 16384 palavras para serem amazenadas, das quais 4096 são para instruções, 4096 para dados estáticos e 8192 para a pilha de execução.

3.3. Principais características

- **Segue o padrão MIPS**
- **Opera com uma palavra de 32-bits**
- **Espaços de memória reservados em 8-bits**

4. Conjunto de instruções da arquitetura

4.1. Tipos de Instruções

As instruções interpretadas pelo Antares-R2 consiste em uma única palavra de 32 bits e são divididas entre três grupos dependendo do seus formatos: R, J e I.

Instruções tipo R

As instruções tipo R operam apenas em registradores e são determinadas pelo seu campo *opcode* que pode ser "000000"ou "011100". Quando montados em código de máquina as instruções tipo R seguem o seguinte formato:

Bits:31-26	Bits:25-21	Bits:20-16	Bits:15-11	Bits:10-6	Bits:5-0
opcode	RS	RT	RD	shamt	function

Tabela 3: Campos de instruções tipo R

Instruções tipo J

As instruções tipo J determinam saltos e são determinadas pelo seu campo *opcode* que pode ser "000010"ou "000011". Quando montados em código de máquina as instruções tipo J seguem o seguinte formato:

Bits:31-26	Bits:25-0
opcode	Adress

Tabela 4: Campos de instruções tipo J

Instruções tipo I

As instruções tipo I operam sobre Imediatos(Constantes) de 16 bits. Quando montados em código de máquina as instruções tipo J seguem o seguinte formato:

Bits:31-26	Bits:25-21	Bits:20-16	Bits:15-0
opcode	RS	RT	Imm

Tabela 5: Campos de instruções tipo I

4.2. Instruções de Load e Store

As instruções de Load e Store utilizam do acesso a memoria para armazenar ou carregar instruções a partir de operandos contidos nos registradores do processador.

Mnemonico	Descrição
LB	Carrega um byte da memória
LW	Carrega uma palavra da memória
LH	Carrega meia palavra da memória
SB	Armazena um byte na memória
SW	Armazena uma palavra na memória
SH	Armazena meia palavra na memória

Tabela 6: Conjunto de instruções de Load e Store

4.3. Instruções Aritmeticas

Mnemonico	Operandos	Realização	Descrição
ADD	$d, $s, $t	$d ← $s + $t	Soma duas palavras
ADDI	$d, $s, Imm.	$d ← $s + Imm.	Soma um imediato a um registrador
ADDIU	$d, $s, Imm.	$d ← $s + Imm.	Soma um imediato a um registrado (Sem overflow)
ADDU	$d, $s, $t	$d ← $s + $t	Soma duas palavras (Sem overflow)
CLZ	$d, $s	$d ← c_zeros($s)	Conta a quantidade de 0's em um registrador
CLO	$d, $s	$d ← c_uns($s)	Conta a quandtidade de 1's em um registrador
LUI	$t, Imm.	$t ← imm..0	Desloca-se o imediato a esquerda 16 bits e concatena-o com 16 bits zero
SEB	$d, $t	$t ← signal::t(7-0)	Extende-se o sinal do byte menos significativo de um registrador
SEH	$d, $t	$t ← signal::t(15-0)	Extende-se o sinal de meia palavra menos significativa de um registrador
SUB	$d, $s, $t	$d ← $s - $t	Subtrai duas palavras
SUBU	$d, $s, $t	$d ← $s - $t	Subtrai duas palavras. Sem sinal.

Tabela 7: Conjunto de operações aritméticas

4.4. Instruções Logicas

Mnemonico	Operandos	Realização	Descrição
AND	$d, $s, $t	$d ← $s ⊙ $t	Efetua uma operação logica AND entre dois registradores
ANDI	$t, $s, Imm	$d, ← $s ⊙ Imm.	Efetua uma operação logica AND entre um registrador e um imediato
NOR	$d, $s, $t	$d ← $\overline{\$s + \$t}$	Efetua uma operação lógica NOR entre dois registradores
OR	$d, $s, $t	$d ← $s + $t	Efetua uma operação logica OR entre dois registradores
ORI	$t, $s, c	$d ← $s + Imm.	Efetua uma operação logica OR entre um registrador e um imediato
XOR	$d, $s, $t	$d ← $s ⊕ $t	Efetua uma operação logica XOR entre dois registradores
XORI	$d, $s, Immm.	$d ← $s ⊕ Imm.	Efetua uma operação logica XOR entre um registrador e um imediato

Tabela 8: Conjunto de operações lógicas

4.5. Instruções de Multiplicação e Divisão

Mnemonico	Operandos	Realização	Descrição
DIV	$s, $t	(LO,HI) ←$s/$t	Divide duas palavras
DIVU	$s, $t	(LO,HI) ←$s/$t	Divide duas palavras. Sem sinal
MADD	$s, $t	(LO,HI) ←$s × $t	Multiplica os registradores e adiciona ao acumulador
MADDU	$s, $t	(LO,HI) ←$s × $t	Multiplica os registradores (sem sinal) e adiciona ao acumulador
MSUB	$s, $t	(LO,HI) ←$s × $t	Multiplica os registradores e subtrai ao acumulador
MSUBU	$s, $t	(LO,HI) ←$s × $t	Multiplica os registradores (sem sinal) e subtrai ao acumulador
MUL	$d, $s, $t	$d ←$s × $t	Multiplica os registradores
MULT	$s, $t	(LO,HI) ←$s × $t	Iguala o acumulador ao valor da multiplicação dos registradores
MULTU	$s, $t	(LO,HI) ←$s × $t	Iguala o acumulador ao valor da multiplicação dos registradores (sem sinal)

Tabela 9: Conjunto de operações de multiplicação/divisão

4.6. Instruções de Deslocamento e Rotação

Mnemonico	Operandos	Realização	Descrição
ROTR	$d, $s, sa	$d← $s ↔ (direita) sa	Executar uma rotação logica a direita de uma palavra por um número fixo de bits
ROTRV	$d, $s, $t	$d← $s ↔ (direita) $t	Executar uma rotação logica a direita de uma palavra por um número variável de bits
SLL	$d, $t, c	$d←$t«c	Aplica deslocamento no registrador a partir de um número fixo de bits
SLLV	$d, $s, $t	$d←$t«$t	Aplica deslocamento no registrador a partir de um número variável de bits
SRA	$d, $s, c	$d→$s»c	Aplica deslocamento a direita numa palavra por um número fixo de bits.
SRAV	$d, $s, $t	$d→$s»$t	Aplica deslocamento a direita numa palavra por um número variável de bits.
SRL	$d, $s, c	$d→$s»c	Executar um deslocamento lógico a direita de uma palavra por um número fixo de bits
SRLV	$d, $s, $t	$d→$s»$t	Executar um deslocamento lógica a direita de uma palavra por um número variável de bits

Tabela 10: Conjunto de operações de Rotação e Deslocamento

4.7. Branchs e Jumps

Mnemonico	Operandos	Realização	Descrição
BEQ	$s, $t, L	se $s = $t, PC← PC + L	Desvia para a instrução sinalizada caso os registradores sejam de valores iguais
BNE	$s, $t, L	se $s != $t, PC← PC + L	Desvia para a instrução sinalizada caso os registradores sejam de valores diferentes
J	L	PC← L	Desvia para a instrução do endereço de destino especificado
JAL	L	$ra← PC+4, PC← L	Chamada de procedimento especificado no endereço
JALR	$d, $s	$d← PC+4, PC← $s	Executa um procedimento chamando uma instrução no endereço de um registrador
JR	$s	PC← $s	Retorno de procedimento

Tabela 11: Conjunto de operações de Branch e Jump

4.8. Testes Condicionais

Mnemonico	Operandos	Realização	Descrição
MOVN	$d, $s, $t	if ($t != 0) then $d ← $s	Mover condicionalmente o conteúdo de um registrador após testar seu valor
MOVZ	$d, $s, $t	if ($t = 0) then $d ← $s	Mover condicionalmente o conteúdo de um registrador após testar seu valor
SLT	$d, $s, $t	$d← $s<$t	Armazenar o resultado de uma comparação Menor que
SLTI	$t, $s, c	$d← $s<c	Armazenar o resultado de uma operação Menor que com um imediato
SLTIU	$t, $s, c	$d← $s<c	Armazenar o resultado de uma operação Menor que com um imediato (sem sinal)
SLTU	$d, $s, $t	$d← $s<$t	Armazenar o resultado de uma operação Menor que (sem sinal)

Tabela 12: Conjunto de operações condicionais

4.9. Acesso ao acumulador

Mnemonico	Operandos	Realização	Descrição
MFHI	$d	$d = HI	Copiar o conteudo do registrador HI para um registrador
MFLO	$d	$d = LI	Copiar o conteudo do registrador LO para um registrador
MTHI	$s	$s = HI	Copiar o conteudo de um registrador para o registrador HI
MTLO	$s	$s = HI	Copiar o conteudo de um registrador para o registrador LO

Tabela 13: Conjunto de operações de acesso ao acumulador

4.10. Pseudo-Instruções

Mnemonico	Operandos	Realização	Descrição
LA	$d, L	$d ← Imm(32)	Copia o valor do endereço da label para o registrador
LI	$d, Imm	$d ← $0 + Imm.	Copia o valor do imediato para o registrador
MOVE	$d,$s,$t	$d ← $d + $st	Copia o conteúdo do registrador s para o d
NEGU	$d,$s	$d ← $0 - $d.	Copia o inverso do conteúdo do registrador S para D
NOT	$d,$s	$d ← $\overline{\$s}$	Nega o valor do registrador s e armazena no registrador d
BEQZ	$s L	———-	Vai para a instrução especificada no endereço L se o valor no registrador s for igual a 0
BNEZ	$s L	——–	Vai para a instrução especificada no endereço L se o valor no registrador s for diferente de 0

Tabela 14: **Conjunto de pseudo-instruções**

5. Montador

5.1. Descrição do Montador

O montador(Assembler) é o programa que transforma o código escrito na linguagem Assembly em linguagem de máquina, substituindo as instruções pelos códigos binários e endereços de memória correspondentes. O montador apresentado neste documento foi desenvolvido na linguagem de programação Java, logo é compatível com os sistemas operacionais mais utilizados, precisando apenas da instalação da JVM (Java Virtual Machine) para ser executado.

5.2. Modo de usar

Para executar o montador basta utilizar o comando:

```
java -jar montador.jar
```

Em seguida será exibido uma janela com as opções de seleção do código Assembly e definir o nome e o diretório para salvar o arquivo de saída. Após isso, basta clicar no botão *Assembler* e o código Assembly será convertido para código objeto e salvo no diretório especificado.

5.3. Restrições

- O arquivo de entrada deve estar nos formatos *.asm* ou *.txt.*
- No código não pode haver instrução na mesma linha de uma label, ou duas instruções na mesma linha.
- Deve haver apenas um espaço simples entra o mnemônico e os registradores.Os registradores devem ser separados por vírgula, e deve haver apenas um espaço simples após cada virgula.
- O montador aceita apenas a diretiva *.text*, informando ao montador que as linhas seguintes são códigos da linguagem Assembly. Caso uma outra diretiva seja detectada, será exibido um erro mostrando a linha onde o problema está ocorrendo e o código objeto não será gerado.

5.4. Informações adicionais

- Aplica-se ao montador um conjunto de configurações que são encontradas junto ao diretório do mesmo, o qual determina os caminhos de arquivos fundamentais para o funcionamento do simulador, sendo esses o conjunto de registradores e o conjunto de instruções.
- Utiliza-se o caractere ' **;** ' fazer comentários no código. Todas as informações presentes entre este caractere e o fim da linha serão ignorados pelo montador.
- Caso algum erro seja encontrado durante a leitura do arquivo do código Assembly, a execução do programa será abortada e o usuário poderá ver uma mensagem de erro indicando a linha e qual o erro detectado.

6. Simulador

6.1. Modo de usar

Através da interface do simulador, após escolher o arquivo binário a ser simulado, se é possível visualizar o código binário em questão e escolher o tipo de execução, podendo ela ser direta, ou seja, executar todos os passos até a finalização, ou passo a passo, possibilitando acompanhar minuciosamente a execução. Ao seguir, a tabela de registradores é exibida, essa possuindo o nome de cada registrador, os valores neles armazenados em hexadecimal, decimal e binário. Finalizando a execução, pode-se acessar uma tabela contendo os valores correspondentes as palavras dentro da memória compartilhada associada ao processador.

6.2. Limitações

Esse simulador não é capaz de gerar arquivos contendo um log de execução, também o estado final dos registradores e/ou o estado final da memória compartilhada.

6.3. Requisítos não funcionais

O simulador somente executa códigos binários gerados por montadores que seguem a arquitetura MIPS. É necessário a presença da máquina virtual Java para se fazer uso do simulador.

6.4. Informações adicionais

Aplica-se ao simulador um conjunto de configurações que são encontradas junto ao diretório do mesmo, o qual determina os caminhos de arquivos fundamentais para o funcionamento do simulador, sendo esses o conjunto de registradores e o conjunto de instruções.