PUC – Pontifícia Universidade Católica – Campinas/SP
Mestrado em Sistemas de Computação
Disciplina – Arquitetura de Sistemas Computacionais
Docente: Prof. Dr. Ricardo Pannain
Aluno: André Luiz da Silva

PCSPIM: Simulador de uma Arquitetura MIPS para Windows

O simulador PCSPIM encontra-se em desenvolvimento continuado (versão atual 6.3 de 24/10/2001), pelo que se podem encontrar situações em que o seu comportamento poderá apresentar algumas inconsistências.

Introdução

O SPIM é um simulador de uma arquitetura baseada nos processadores MIPS R2000/R3000 desenvolvido pelo Departamento de Ciências de Computação da Universidade de Wisconsin. O PCSPIM é uma aplicação para plataformas Windows 95 ou Windows WIN32S que implementa a versão 6.3 deste simulador. A aplicação é carregada contendo um programa em linguagem *assembly* do MIPS, fornecendo os mecanismos que permitem fazer a sua execução ou depuração (*debugging*). O interface com o utilizador é baseado em janelas.

A arquitetura simulada pelo PCSPIM é muito simplificada relativamente a uma arquitetura real baseada no MIPS, sendo constituída essencialmente pelo processador mais memória. Não existe simulação de entrada/saída (E/S): a comunicação com o exterior faz-se através de um conjunto de funções do tipo chamada ao sistema operacional que o simulador disponibiliza. Do processador propriamente dito nem todas as partes são cobertas. Dos quatro coprocessadores que acompanham o MIPS apenas o coprocessador de vírgula flutuante (coprocessador C1) e parte do coprocessador de gestão de memória (coprocessador C0) são simulados.

Simulação da Máquina Virtual

A arquitetura MIPS, à semelhança de muitos computadores RISC, é difícil de programar diretamente devido às particularidades do seu reportório de instruções: saltos com atraso (*delayed branch*), *loads* com atraso (*delayed load*) e modos de endereçamento bastante restritos. Esta dificuldade é normalmente tolerada porque os computadores RISC são projetados para serem programados em linguagens de alto nível, o que significa que o código de máquina é gerado por compiladores e não por programadores.

Um *delayed branch* demora dois ciclos a executar. No segundo ciclo a instrução imediatamente a seguir é executada. Esta instrução pode realizar trabalho útil que normalmente seria realizado antes da instrução de salto, ou pode ser a instrução **nop**. Um *delayed load* também demora dois ciclos a executar, pelo que a instrução que imediatamente a precede não pode usar o valor que está sendo carregado da memória. Os assembladores de MIPS optam normalmente por esconder esta complexidade implementando uma *máquina virtual* que apresenta

saltos e *loads* sem atrasos, cabendo ao montador o papel de reordenar as instruções de forma a preencher os *slots* de atraso.

Por outro lado esta máquina virtual apresenta um reportório de instruções mais rico que o do processador real, quer pela inclusão de novas instruções, quer pelo aumento de modos de endereçamento. Estas instruções adicionais, designadas por pseudo-instruções, são convertidas pelo montador em sequências de instruções nativas.

É habitual designar por pseudo-instruções as directivas para o Assembler. Aqui o seu significado é um pouco diferente, sendo mais próximo de macro-instrução.

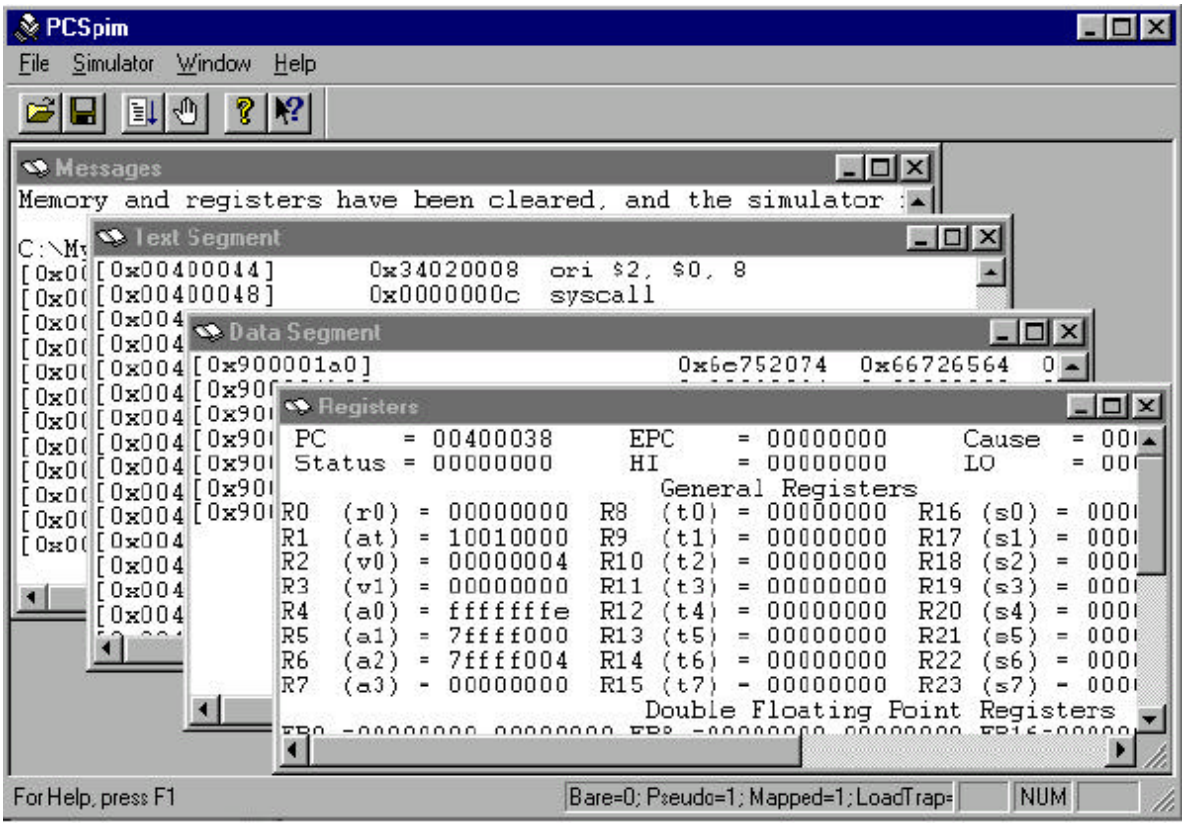


Figura 1 - O Interface do PCSPIM.

Interface do PCSPIM
O PCSPIM, apresenta uma janela principal constituída por uma barra de menus e uma área de trabalho sobre a qual se podem abrir quatro janelas: **Text Segment**, **Data Segment**, **Registers** e **Messages**. (Veja na Figura 1 um exemplo do interface com as quatro janelas abertas). estas quatro primeiras janelas são implementadas por janelas Windows com o mesmo formato e distinto do de uma quinta janela, a janela **Console**. São janelas de tamanho ajustável cuja visibilidade está limitada à área de trabalho da janela principal.

A janela **Console** é implementada por uma janela Windows cuja visibilidade é independente da janela principal. Esta janela tem a particularidade (desagradável) de ser fechada sempre que termina, ao nível do simulador, a execução de um programa Assembly.
A janela **Text Segment** mostra os segmentos de código do utilizador (.text) e de código do *kernel* (.ktext) do programa carregado. Cada linha nesta janela representa uma instrução do MIPS e está formatada em quatro colunas. A primeira coluna (mais à esquerda) indica o endereço da palavra de memória onde a instrução está armazenada; sendo a memória endereçada ao byte (*byte-addressable*) e ocupando uma instrução 4 bytes, ela está armazenada no endereço indicado e nos 3 seguintes. Na segunda coluna aparece o código de máquina da instrução (4 bytes) em notação hexadecimal. Na terceira coluna mostra-se a instrução em *assembly* nativo. Os valores numéricos são, nesta janela, apresentados em decimal. Na última coluna mostrasse a instrução tal como foi escrita no programa fonte. Quando a instrução do programa fonte se expande em mais que uma instrução nativa, a instrução virtual é apresentada na linha correspondente à primeira instrução da expansão.
A janela **Data Segment** mostra o segmento de dados do utilizador (.data), a *stack* e o segmento de dados do *kernel* (.kdata) do programa carregado. A informação é apresentada formatada em duas colunas. Na primeira apresentam-se os endereços de um bloco de memória (normalmente 16 bytes) e na segunda o conteúdo desse bloco em notação hexadecimal. A organização dos bytes no contexto de cada palavra de quatro bytes (*word*) obedece, neste simulador, ao modelo *little endian* (os bytes menos significativos precedem os de maior significância).

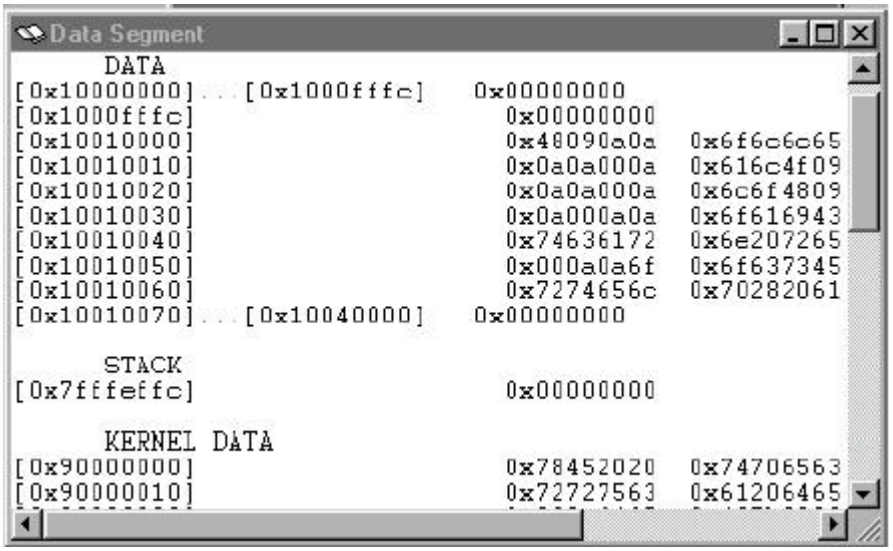


Figura 2- Janela “Data Segment”.

A janela **Registers** mostra o conteúdo (em decimal ou hexadecimal) dos registradores internos do MIPS. Está dividida em quatro paineis de registradores.

O primeiro (no topo) mostra o valor dos registradores especiais **PC**, **EPC**, **Cause**, **BadVAddr**, **Status**, **HI** e **LO**. O segundo, intitulado **General Registers** mostra o conteúdo dos 32 registradores de uso geral do processador. Para cada registo, entre parêntesis, é indicado o seu nome lógico (ver subsecção "registradores da CPU"). Os 2 painéis de registradores seguintes intitulados, respectivamente, **Double Floating Point Registers** e **Single Floating Point Registers** representam as duas leituras possíveis (em precisão dupla ou simples respectivamente) para os registradores do coprocessador de vírgula flutuante. (ver subsecção "Registradores do coprocessador de vírgula flutuante".)
A janela **Messages** é o meio através do qual o simulador envia mensagens ao utilizador, independentemente da sua natureza.
A janela **Console**, finalmente, é o meio através do qual o programa interage com o utilizador. Esta janela comporta-se como um terminal não inteligente, sendo através dela que são levadas a cabo todas as operações de leitura de dados e escrita de resultados resultantes da execução do programa assembly simulado.

Comandos dos Menus

File

Load
Permite carregar para memória um programa em *assembly* do MIPS contido em arquivo. Por defeito considera que esse arquivo tem a extensão **.s**. Esta operação
inicializa o conteúdo da memória e dos registradores internos do simulador.

Save Log File
Permite criar um arquivo de texto com a descrição do estado atual do simulador,
incluindo o conteúdo de cada uma das suas janelas.

Exit
Abandona o simulador.

Simulator

Clear Registers
Inicializa o conteúdo de todos os registradores com o valor 0.

Reinitialize
Reinicializa os registradores, a memória e o motor do simulador.

Reload
Reinicializa os registradores, a memória e o motor do simulador e recarrega e
Assembla o programa fonte corrente.

Go
Executa o programa previamente carregado. A janela de diálogo permite definir o endereço inicial da execução e os parâmetros a introduzir através da linha de comando.

Break/Continue

Quando o programa carregado se encontra em execução, este comando permite a sua interrupção com a consequente atualização dos dados presentes em cada uma das janelas. Quando em modo de interrupção, o comando permite que o programa seja recomeçado a partir do ponto onde foi interrompido.

Simulator

Single Step

Executa a próxima instrução (correspondente ao conteúdo da posição de memória apontada pelo registo **PC** e reentra em modo *Break***.**. O conteúdo das várias janelas é atualizado com os novos dados.

Multiple Step

Semelhante ao comando anterior mas permitindo agora que o operador defina o número de instruções que devem ser executadas antes de o simulador entrar novamente em modo *Break*.

Breakpoints

Evoca uma caixa de diálogo que permite editar (criar ou apagar) os endereços do segmento de texto no qual se pretende colocar *break points*.

Set Value

Evoca uma caixa de diálogo através da qual é possível alterar o conteúdo dos registradores do processador ou da memória. A base de representação dos valores numéricos segue a convenção adoptada na linguagem “C” (e.g. 128 - decimal; 0xfe2 - hexadecimal).

Display symbol table

Apresenta, na janela de mensagens, a tabela de símbolos e os respectivos
endereços em memória.

Settings

Evoca uma caixa de diálogo que permite alterar a configuração do simulador, e bem assim, as opções do interface com o utilizador.

Window

Next

Coloca em primeiro plano a próxima janela activa.

Previous

Coloca em primeiro plano a última janela activa.

Cascade

Redistribui as janelas activas no espaço de trabalho adotando uma organização sobreposta.

Tile

Redistribui as janelas activas no espaço de trabalho adotando uma organização não sobreposta.

Arrange Icons

Reorganiza, no espaço de trabalho, os ícones das janelas minimizadas.

Messages
Selecciona e coloca em primeiro plano a janela de mensagens.

Text Segment
Selecciona e coloca em primeiro plano a janela do segmento de texto.

Data Segment
Selecciona e coloca em primeiro plano a janela do segmento de dados.

Registers
Selecciona e coloca em primeiro plano a janela do registradores internos.

Console
Selecciona e coloca em primeiro plano a janela do console.

Clear Console
Inicializa o conteúdo da janela do console.

Toolbar
Permite ativar/desativar a barra de ferramentas.

Status Bar
Permite ativar/desativar a barra de status.

Sintaxe da Linguagem *Assembly*

Um programa *assembly* é um texto a 3 colunas, em que a primeira representa de uma forma simbólica endereços de memória (*labels*), a segunda instruções e a última comentários. Não existe nenhuma imposição quanto ao posicionamento das colunas no texto a não ser obviamente que a sua ordem deve ser respeitada, embora isso seja feito frequentemente para melhorar a legibilidade do texto.

Exemplo:

```
            .data
Var_1:      .word 1
            .text
            .globl main
main:       li $v0, 4           #Comentário
            lw $a0, Var_1
```

Os endereços são representados por identificadores -- etiquetas (*labels*) -- e devem ser terminados pelo carácter ":". É considerado como comentário tudo o que estiver entre o carácter "#" e o fim da linha, sendo consequentemente ignorado pelo assemblador.

Os identificadores são sequências de caracteres alfanuméricos, barras horizontais, "_**",** e pontos, ".", que não comecem por um carácter numérico. As mnemónicas (*opcodes*) das instruções são palavras reservadas, **não** podendo consequentemente ser usadas como identificadores. Um programa *assembly* pode estar distribuído por mais que um arquivo; os identificadores podem ser locais,

sendo assim apenas válidos no contexto desse arquivo, ou globais, sendo assim válidos em todos os arquivos que constituem o programa.
As *strings* são sequências de caracteres delimitadas por aspas (“). A utilização de caracteres especiais segue a convenção da linguagem de programação C.
Os caracteres válidos são:

“\t” => tab; “\n” => new line; “\”” => quote

O campo instrução corresponde a uma instrução (nativa ou virtual) do MIPS ou a uma directiva. As directivas são mecanismos que permitem controlar a forma como a assemblagem é feita, não resultando assim em código executável. São constituídas por um identificador (cujo primeiro caracter é sempre o símbolo “.”) e em alguns casos por um ou mais parâmetros. O PCSPIM suporta o seguinte subconjunto das directivas definidas para o montador de MIPS:

- Para controle dos segmentos:

  .data <address>
  Os valores definidos a seguir devem ser colocados no segmento de dados do utilizador, opcionalmente a partir do endereço <address>.
  .text <address>
  Os valores definidos a seguir devem ser colocados no segmento de texto do utilizador, opcionalmente a partir do endereço <address>. Todos os valores devem medir 32 bits, ou seja, serem instruções ou palavras (words).
  .kdata <address>
  Os valores definidos a seguir devem ser colocados no segmento de dados do kernel, opcionalmente a partir do endereço <address>.
  .ktext <address>
  Os valores definidos a seguir devem ser colocados no segmento de texto do kernel, opcionalmente a partir do endereço <address>.

- Para criação de constantes e variáveis em memória:

  .ascii str
  armazena uma *string* em memória sem lhe acrescentar o terminador NULL.
  .asciiz str
  armazena uma *string* em memória acrescentando-lhe o terminador NULL.
  .byte b1, ..., bn
  armazena as grandezas de 8 bits **b1, ..., bn** em sucessivos bytes de memória.
  .half h1, ..., hn
  armazena as grandezas de 16 bits **h1, ..., hn** em sucessivas meias-palavras de memória.
  .word w1, ..., wn
  armazena as grandezas de 32 bits **w1, ..., wn** em sucessivas palavras de memória.
  .float f1, ..., fn

armazena os números em vírgula flutuante com precisão simples (32 bits) **f1, ..., fn** em posições de memória sucessivas.

.double d1, ..., dn

armazena os números em vírgula flutuante com precisão dupla (64 bits) **d1, ..., dn** em posições de memória sucessivas.

.space n

reserva **n** bytes (SPIM só deixa usar esta directiva no segmento **.data**).

- Para controle do alinhamento:

.align n

alinha o próximo item num endereço múltiplo de 2n . Por exemplo **.align 2** seguido de **.word xpto** garante que a palavra **xpto** é armazenada num endereço múltiplo de 4.

.align 0

desliga o alinhamento automático das directivas **.half**, **.word**, **.float**, e **.double** até à próxima directiva **.data** ou **.kdata**.

- Para referências externas:

.globl sym

declara que o símbolo **sym** é global e pode ser referenciado a partir de outros arquivos.

.extern sym size

declara que o item associado a **sym** ocupa **size** bytes e é um símbolo global. Esta directiva permite ao montador armazenar o item numa porção do segmento de dados que seja eficientemente acedido através do registo $gp.

Chamadas ao Sistema (*System Calls*)

O SPIM dispõe de um conjunto de funções típicas de um sistema operacional (ver Tabela I).

Função	Sv0	Parâmetros	Retorno
print_int(int)	1	$a0 = int	
print_float(float)	2	$f12 = float	
print_double(double)	3	$f12 = double	
print_string(string)	4	$a0 = string	
read_int(int)	5		$v0 = integer
read_float(float)	6		$f0 = float
read_double(double)	7		$f0 = double
read_string(string)	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	$v0 = address
exit	10		

A sua utilização é feita através da instrução de chamada ao sistema ***syscall***. Para invocar uma determinada função o programa deve carregar o código respectivo no registo **$v0** e os argumentos nos registradores **$a0** a **$a1** (ou **$f12** para valores em vírgula flutuante). O valor de retorno (caso exista), é devolvido no registo **$v0** (ou **$f0** no caso de ser em vírgula flutuante).

Exemplo de utilização da *System Call* print_string:

```
        .data
str:    .asciiz "Uma string!"
        .text
        .globl main
main:  li $v0, 4
        la $a0, str
        syscall                    # print_string (“Uma string”);
…
```

As funções *“print”* imprimem um valor (**int** – inteiro, **float** – real com precisão simples, **double** – real com precisão dupla, **string** – sequência de caracteres terminada por um NULL) na janela *Console*.
As funções “*read”* lêem valores a partir do teclado. O valor lido deve ser terminado pela tecla *“Enter*”. Nas funções de leitura de valores numéricos, são ignorados todos os caracteres lidos a partir (e incluindo) o primeiro caracter não numérico. A função “*read_string*” lê **n-1** caracteres para o *buffer* de entrada e termina a *string* com um caracter NULL. A função termina quando o número de caracteres lido for igual a **n-1** ou quando receber um *new line* (tecla “*Enter*”).
“*sbrk”* devolve um ponteiro para um bloco de memória contendo *n* bytes adicionais.
“*exit”* termina a execução de um programa.

Passagem dos parâmetros da linha de comando

O PCSPIM permite que o programa Assembly recupere do stack os parâmetros que lhe sejam passados a partir da linha de comando. Para isso, os primeiros 32 bytes do segmento de texto são inicializados automaticamente pelo PCSPIM com uma porção de código que inicializa os ponteiros para os locais próprios.
O valor de argc, variável que em linguagem C contém o número de parâmetros da linha de comando, é passado ao programa através do registo $a0. $a1, por sua vez, contém o valor de argv, variável que contém um ponteiro para uma lista de ponteiros. Cada um dos ponteiros dessa lista (num total de argc ponteiros) aponta para uma string que contém o conteúdo de um dos parâmetros. $a2, finalmente, contém o ponteiro para a lista de variáveis do environment. Note que, ao contrário do que acontece em C, a lista de parâmetros não inclui o nome do programa. Consequentemente, Argv[0] aponta diretamente para o primeiro parâmetro da lista.

Descrição da Arquitetura

Um processador MIPS é constituído por uma unidade de processamento de inteiros (o CPU) e vários coprocessadores que realizam tarefas subsidiárias ou operam sobre outros tipos de dados como sejam números em vírgula flutuante (Figura 3). O coprocessador de controle de sistema (CP0) é responsável pela gestão de *traps*, excepções e o sistema de gestão da memória virtual. O coprocessador de vírgula flutuante (CP1), por sua vez, realiza operações aritméticas e lógicas sobre números reais representados em vírgula flutuante no formato IEEE-754. O PCSPIM simula o funcionamento da CPU, do CP1 e das *traps* e excepções do CP0.

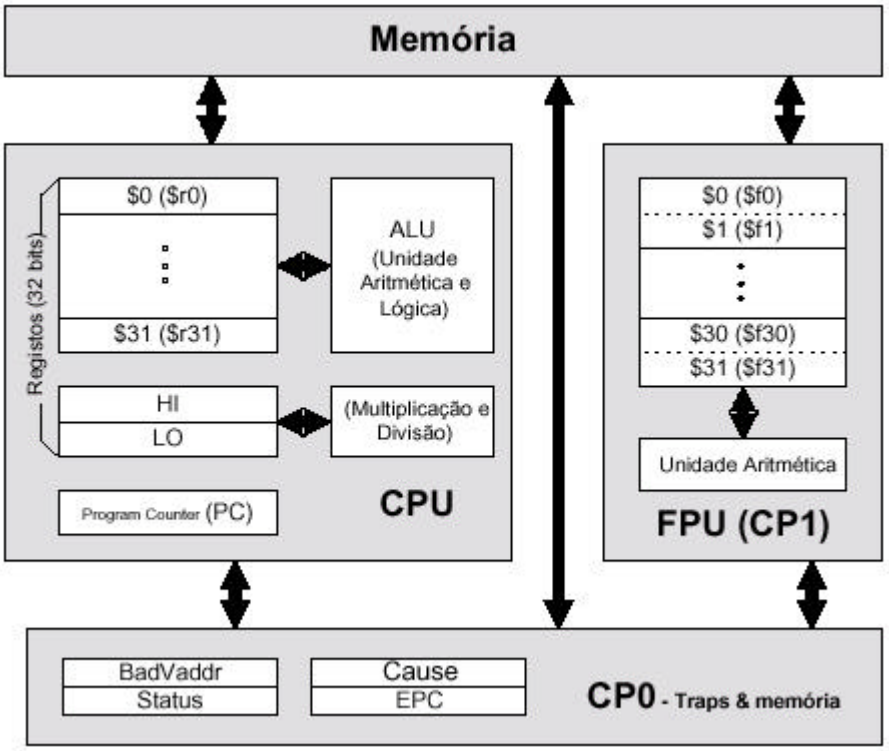


Figura 3 - *Arquitetura simplificada do MIPS*

Registradores da CPU

A unidade de processamento central (CPU) do MIPS possui um *program counter* de 32 bits (**PC**), dois registradores de 32 bits para armazenar os resultados de multiplicações e divisões inteiras (**HI** e **LO**) e 32 registradores de uso geral de 32 bits,
numerados de 0 a 31 (**$0** a **$31**). Dois destes registradores, embora de uso geral, apresentam particularidades de registo: o registo **$0** contém sempre o valor 0 (*hardwired*) e o registo **$31** (*link register*) é usado pelas instruções de *Jump and Link* para armazenar o endereço de retorno.
O PCSPIM simula todos estes registradores.

Relativamente à utilização, por parte dos programas *Assembly*, dos registradores de uso geral, a MIPS estabeleceu um conjunto de regras ou convenções que devem ser seguidas pelos programadores. Embora estas convenções não correspondem a imposições do *hardware*, a sua adopção é fortemente recomendada por
forma a garantir que o código gerado funcione correctamente em outras plataformas. Com vista a facilitar o seguimento da convenção estabelecida pela MIPS, a cada um dos registradores foi atribuído um nome lógico que evidencia o seu tipo de uso. Na Tabela II são identificados, para cada registo de uso geral, a

correspondente designação lógica e o tipo de uso que lhe está destinado pela convenção.

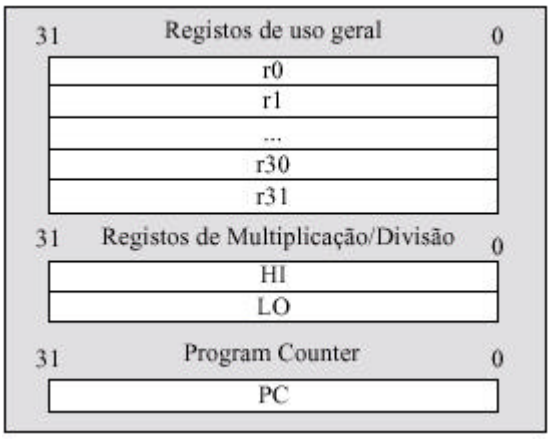


Tabela 4 – Os registradores da CPU

Nome Lógico	Nome Real	Uso
$zero	$0	Constante 0
$at	$1	Reservado pelo assemblador
$v0..$v1	$2..$3	Cálculo de expressões e valor de retorno das funções.
$a0..$a3	$4..$7	Primeiros 4 parâmetros das funções
$t0..$t7	$8..$15	Geral (pode não ser preservado pelas funções)
$s0..$s7	$16..$23	Geral (deve ser preservado pelas funções)
$t8..$t9	$24..$25	Geral (pode não ser preservado pelas funções)
$k0..$k1	$26..$27	Reservado pelo *kernel* do S.O.
$gp	$28	Ponteiro para área global (*Global Pointer*)
$sp	$29	*Stack Pointer*
$fp	$30	*Frame Pointer*
$ra	$31	Endereço de retornos das funções (*Return Address*)

Tabela II – Registradores do MIPS e convenção de uso.

Os registradores **$at** (**$1**) e **$k0-$k1** (**$26-$27**) estão reservados para uso pelo montador e sistema operativo.
Os registradores **$a0-$a3** (**$4-$7**) são usados para passar às subrotinas 3 os seus quatro primeiros parâmetros.
Se for necessário passar mais do que quatro parâmetros os remanescentes deverão ser passados na *stack*.
Os registradores **$v0-$v1** (**$2-$3**) são usados para armazenar temporariamente o resultado de expressões aritméticas ou lógicas ou para o retorno de valores das funções.
Os registradores **$t0-$t9** (**$8-$15**, **$24**, **$25**) são usados para armazenar informação temporária que não precise de ser preservada entre chamadas a

subrotinas; são designados por registradores "*caller-saved*", i.e., cabe ao programa em execução preservar os seus valores no caso de necessitar do seu valor após uma chamada a uma subrotina.
Os registradores **$s0-$s7** (**$16-$23**) são usados para armazenar valores com um tempo de vida longo que precisam de ser preservados entre chamadas a subrotinas; são designados por registradores "*callee-saved*", i.e., cabe à sub-rotina em execução preservar os seus valores no caso que necessitar de recorrer à sua utilização
temporária.

Alguns compiladores, por forma a generalizarem a geração de código, optam por passar todos os parâmetros na *stack*.

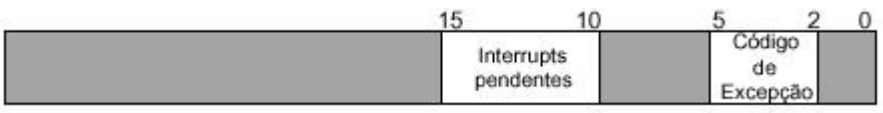
O registo **$sp** (**$29**), ou *stack-pointer*, aponta para a última posição ocupada na *stack*. Embora o MIPS não tenha instruções próprias para manipular a *stack*, convenciona-se que aquela cresce no sentido decrescente dos endereços e que o conteúdo de **$sp** é um ponteiro para a última posição ocupada da mesma.
O registo **$fp** (**$30**), ou *frame-pointer*, é usado por alguns compiladores para acesso à informação na *stack4*.
O registo **$gp** (**$28**) é um apontador global (*global pointer*) que aponta para o meio de um bloco de memória de 64 Kbytes definido na memória *heap* e usado para definir constantes e variáveis globais.
O registo **$ra** (**$31**), finalmente, é usado pela instrução **jal** para armazenar o endereço de retorno das subrotinas.

Registradores do Coprocessador CP0

O coprocessador CP0 contém registradores necessários para a gestão das excepções, do sistema de memória virtual e das *caches* de dados e instruções. Contém ainda um registo com a indicação da versão do processador. O PCSPIM simula apenas 4 destes registradores:

BadVAddr contém o endereço de memória onde ocorreu a excepção de memória.

Cause indica o tipo de excepção ocorrida. Apenas os bits 2 a 5 e 10 a 15 têm significado, conforme se pode observar na figura seguinte.



Os compiladores da MIPS não usam *frame-pointer* pelo que acrescentam este registo ao grupo dos "callee-saved" ($s8). Em contra partida os compiladores da GNU já o usam como *frame-pointer*.

Tabela II: Registradores do MIPS e convenção de uso.

Os cinco bits que identificam os *interrupts* pendentes (**b15..b10**) correspondem aos cinco níveis de interrupção possíveis. Um bit com o valor lógico '1' neste campo indica a ocorrência de um *interrupt* a esse nível que ainda não foi servido. O código de excepção, correspondente aos bits **b5..b2**, contém um código de identificação de interrupção de acordo com a Tabela III.

Número	Nome	Descrição
0	INT	*Interrupt* externo
4	ADDRL	Erro de endereço (em instrução de *load* ou na operação de *fetch*)
5	ADDRS	Erro de endereço (em instruções de *store*)
6	IBUS	Erro de *Bus* em operação de *fetch* de instrução
7	DBUS	Erro de *Bus* em operação de *Load* ou *Store* de dados
8	SYSCALL	Excepção de chamada ao sistema
9	BKPT	*Break Point*
10	RI	Excepção de instrução reservada
12	OVF	Excepção provocada por *overflow* aritmético

Tabela III – Códigos de exceção

Status contém um conjunto de bits de *status* e controle dos quais são implementados pelo PCSPIM os indicados na figura seguinte.

A máscara de *interrupts* (**$b_{15}..b_{10}$**) contém um bit para cada um dos cinco níveis de interrupção.
Quando um desses bits toma o valor lógico '**1**', o nível de interrupção correspondente encontrasse activo. Os seis bits menos significativos deste registo implementam um *stack* de três níveis para os bits de identificação do *kernel/user* e *interrupt enable*. Um valor lógico '**0**' no bit *kernel/user* indica que o programa estava a correr parte do *kernel* no momento em que ocorreu a interrupção. Um valor lógico '**1**' indica que a execução pertencia ao utilizador.

EPC guarda o endereço da instrução que causou a excepção.

Registradores do Coprocessador de Vírgula Flutuante (CP1)

O MIPS possui um coprocessador de vírgula flutuante (o coprocessador CP1) que manipula números reais representados em vírgula flutuante com precisão simples (32 bits) e precisão dupla (64 bits) de acordo com a norma IEEE-754. Os operandos para as operações de cálculo disponibilizadas pelo coprocessador pertencem a um conjunto específico de registradores designados por **$f0-$f31**. Estes registradores desempenham um papel duplo.
Por um lado podem ser vistos como um conjunto de 32 registradores independentes com capacidade para armazenar quantidades em vírgula flutuante com precisão simples. Por outro lado, cada registo par pode ser agrupado ao seguinte para formar um registo com capacidade para armazenar quantidades em vírgula flutuante com precisão dupla. Por forma a simplificar a sua utilização, as operações de cálculo, sejam de precisão simples ou dupla, usam apenas os registradores de índice par.

Este coprocessador possui ainda dois registradores de controle de 32 bits que são usados para controle do modo de arredondamento, tratamento de excepções, e para salvaguarda do estado. Uma *flag* interna ao coprocessador, à frente designada por fp_flag ou c1_flag, serve de suporte às instruções de comparação e salto condicional relativas a entidades em vírgula flutuante.
Apenas a título de resumo, a norma IEEE -754 estabelece as seguintes características para os formatos de representação numérica em vírgula flutuante:

- Precisão simples

Os números reais representados em vírgula flutuante com precisão simples são representados por palavras de 32 bits organizados de acordo com a figura seguinte:

O valor do número em vírgula flutuante é dado pela expressão:

$$F = (-1)^S 1.f\, 2^{E-127}$$

sendo ***S*** o valor do sinal representado pelo bit $\mathbf{b_{31}}$ (‘**0**’ representa um número positivo; ‘**1**’ representa um número negativo), ***f*** (bits $\mathbf{b_{22}}$…$\mathbf{b_0}$) o valor binário da mantissa no qual se pressupõe a existência de um *hidden* bit de valor 1 à esquerda da vírgula, e ***e*** (bits $\mathbf{b_{30}}$…$\mathbf{b_{23}}$) o valor do expoente codificado em excesso 127.
Existem um conjunto de valores particulares que representam excepções à regra e que se encontram sumariados na Tabela IV.

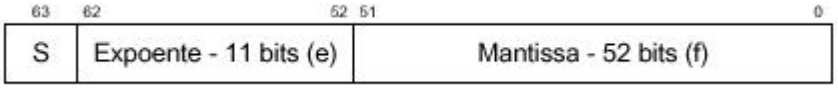
E	*f*	**Significado**
0	0	Representa o valor numérico 0 (zero)
0	≠0	Representação desnormalizada de acordo com a equação $F = (-1)^S 0.f\,2^{-126}$
255	0	Representa ±∞
255	≠0	Representa NAN (Not A Number)

Tabela IV – Exceções no formato IEEE-754

Refira-se em particular que, na representação não normalizada, o *hidden* bit não existe e o expoente é fixo e igual a -126. Esta representação permite diminuir o intervalo entre a menor representação possível e zero, minimizando, consequentemente, o potencial de *underflow*.

- Precisão dupla

Os números reais representados em vírgula flutuante com precisão dupla são representados por palavras de 64 bits organizados de acordo com a figura seguinte:



O valor do número em vírgula flutuante é dado pela expressão:

$$F = (-1)^S 1.f\,2^{E-1023}$$

sendo ***S*** o valor do sinal representado pelo bit $\mathbf{b_{63}}$ (‘**0**’ representa um número positivo; ‘**1**’ representa um número negativo), ***f*** (bits $\mathbf{b_{51}}$…$\mathbf{b_0}$) o valor binário da mantissa no qual se pressupõe a existência de um *hidden* bit de valor 1 à esquerda da vírgula, e ***e*** (bits $\mathbf{b_{62}}$…$\mathbf{b_{52}}$) o valor do expoente codificado em excesso 1023.

A Tabela V apresenta um resumo das principais características destes formatos em precisão simples e dupla respectivamente.

	Precisão simples	Precisão dupla
Comprimento de palavra	32 bits	64 bits
Mantissa + *Hidden* bit	23 + 1 bits	52 + 1 bits
Expoente	8 bits	11 bits
Offset do expoente	127	1023
Gama de representação aproximada	$2^{128} \approx 3.8 \times 10^{38}$	$2^{1024} \approx 9 \times 10^{307}$
Menor número normalizado	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$
Precisão aproximada	$2^{-23} \approx 10^{-7}$	$2^{-52} \approx 10^{-15}$

Tabela V – Mapa comparativo do formato IEEE-754 em precisão simples e dupla.

Modos de Endereçamento

Os processadores MIPS seguem uma arquitetura do tipo "*load/store*", o que significa que apenas as instruções de *load* e de *store* podem aceder à memória. As instruções de cálculo apenas operam sobre o conteúdo de registradores. Além disso o MIPS possui apenas um modo de endereçamento da memória, representado por **imm(reg)**, em que o endereço de memória é determinado pela soma do conteúdo do registo **reg** (qualquer dos registradores gerais da CPU) com a constante **imm** (limitada a uma quantidade de 16 bits).

Modo de Endereçamento	Cálculo do Endereço
`(reg)`	conteúdo do registo `reg`
`imm`	constante `imm`
`imm(reg)`	conteúdo do registo `reg` + constante `imm`
`sym`	endereço do símbolo (*label*) `sym`
`sym +/- imm`	endereço do símbolo `sym` +/- constante `imm`
`sym +/- imm (reg)`	conteúdo do registo `reg` + endereço do símbolo `sym` +/- constante `imm`

Tabela VI – Modos de endereçamento (virtuais) do simulador.

A máquina virtual implementada pelo simulador alarga o leque de modos de endereçamento disponíveis para os que são apresentados na Tabela VI. De notar que, nesta tabela, **imm** pode ser uma quantidade de 32 bits.

Input* e *Output

Para além de simular as operações básicas da CPU do MIPS e do respectivo sistema operacional, o PCSPIM também simula operações de entrada/saída (*Input/Output* ou *I/O*) relativamente a uma ligação a um terminal virtual mapeado em memória. Quando um programa Assembly se encontra em execução, o simulador "liga" o seu próprio terminal virtual ao processador simulado. O programa pode assim ler caracteres diretamente do teclado à medida que estes vão sendo digitados pelo operador. Da mesma forma, é possível executar

instruções para escrever caracteres diretamente no tela do console virtual. A única excepção a esta regra diz respeito à associação de teclas [Control+C]. O resultado não é passado para o programa mas, alternativamente, este é colocado em modo "*break*" devolvendo o comando ao PCSPIM e atualizando o conteúdo das várias janelas. Para poder usar este modo especial de I/O, é necessário que o campo "*Mapped I/O*" da janela de diálogo iniciada pelo comando "*Simulator* | *Settings*" esteja ativado. O dispositivo terminal correspondente à consola virtual consiste em duas unidades independentes: um receptor e um emissor. O receptor lê caracteres a partir do teclado À medida que estes vão sendo digitados. O
emissor, por sua vez, escreve caracteres no tela do console virtual. As duas unidades são completamente independentes. Isso significa, por exemplo, que não é executado o eco automático dos caracteres digitados no teclado. Pelo contrário, o processador deve ler um caracter de entrada a partir do receptor e retransmití-lo para o emissor para obter o seu eco visual.
O programa acede ao terminal virtual usando quatro registradores especiais mapeados em memória (Figura 5).

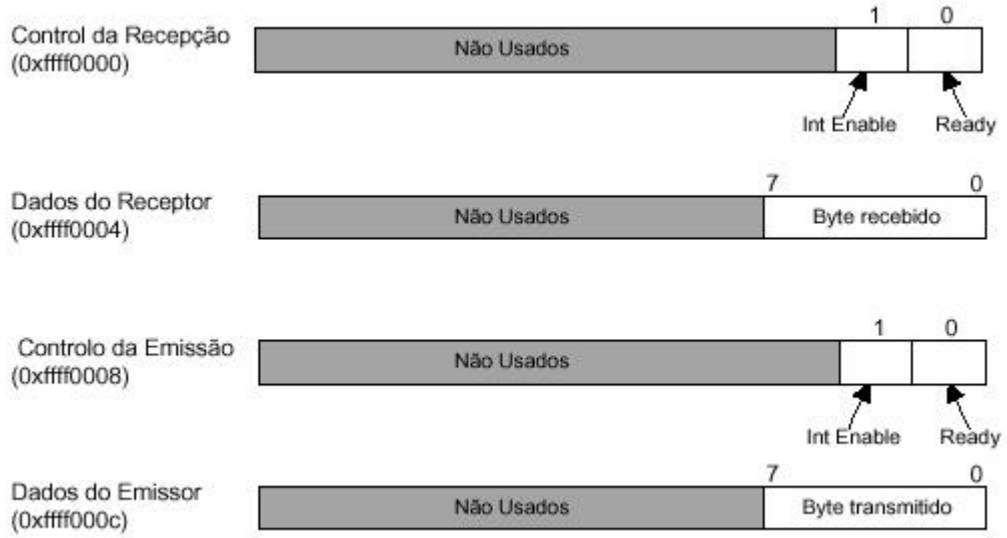


Figura 5 – Os registradores para acesso I/O ao terminal virtual

Mapeados em memória significa que cada um destes registradores surge numa posição de memória especial e dedicada. O "Registo de Controle de Recepção" localiza-se no endereço 0xffff0000. Apenas dois dos seus 32 bits são usados de fato. O bit $\mathbf{b_0}$, designado por "*Ready*", indica, quando o seu valor é '**1**', que um novo
caracter foi recebido a partir do teclado, e que este ainda não foi lido do "Registo de Dados do Receptor".
Este bit é alterado apenas pelo simulador, não podendo consequentemente, ser alterado pelo programa. Este bit muda automaticamente de '**0**' para '**1**' quando um caracter é recebido do teclado, e regressa automaticamente ao valor '**0**' quando o caracter correspondente é lido do "Registo de Dados do Receptor".

O bit **b1** do "Registo de Controle de Recepção" é o "*Interrupt Enable*". Este bit pode ser lido ou escrito pelo programa. Inicialmente, o seu valor é colocado a '**0**'. Se o programa o forçar a '**1**', é gerada uma interrupção de nível zero sempre que o "*Ready*" bit passar de '**0**' para '**1**'. Para que esta interrupção posssendo recebida pelo processador, os *interrupts* devem estar ativos através do registo de **Status** do coprocessador C0.

O segundo registo do dispositivo terminal é o "Registo de Dados do Receptor", o qual existe no endereço 0xffff0004. Os oito bits menos significativos deste registo contêm o último caracter digitado no teclado.

Todos os restantes 24 bits têm o valor '**0**'. Este registo é do tipo "*read-only*". Qualquer tentativa para escrever no mesmo é ignorada pelo simulador. Ler o conteúdo deste registo provoca o "*reset*" do bit "*Ready*" do registo anterior.

O terceiro registo do dispositivo terminal é o "Registo de Controle de Emissão" localizado no endereço 0xffff0008. Apenas os dois bits menos significativos têm significado, e este comportam-se de forma muito semelhante aos bits do "Registo de Controle de Recepção". O bit $\mathbf{b_0}$, quando '**1**', indica que o emissor está pronto o receber mais um caracter. Se o seu valor for '**0**', então o emissor encontra-se ainda ocupado a transmitir o caracter anterior. O bit $\mathbf{b_1}$, por sua vez, é o "*Interrupt Enable*". Um '**1**' neste bit desencadeia uma interrupção de nível um sempre que o "*Ready*" bit passar de '**0**' para '**1**'.

O último registo do dispositivo terminal é o "Registo de Dados do Emissor", o qual existe no endereço 0xffff000c. Quando escrito pelo programa, os seus oito bits menos significativos são interpretados como sendo um caracter ASCII, o qual é enviado e afixado no tela do console. O registo deverá ser escrito apenas quando o bit "*Ready*" do "Registo de Controle de Emissão" se encontrar no seu estado activo ('**1**'). Qualquer operação de escrita efectuada sem ser nessas condições não terá qualquer efeito prático, embora o programa em execução não tenha conhecimento do fato.

Reportório de Instruções

O conjunto de instruções disponibilizadas pelo simulador está enriquecido em relação ao conjunto de instruções da máquina real, não apenas pela existência do maior número de modos de endereçamento mas também pela existência de novas instruções. Muitas destas novas instruções desdobram-se na máquina real em duas ou mais instruções; no entanto, algumas há que representam apenas sintaxes mais apropriadas para operações que na máquina real podem ser realizadas por uma única instrução. Considere-se, a título de exemplo, a instrução virtual **move $s0,$s1**, e as instruções nativas **add $s0,$s1,$0** ou **or $s0,$s1,$0**: embora as três instruções façam rigorosamente o mesmo a instrução virtual é a que retrata de uma forma mais adequada uma operação do tipo "**a** toma o valor de **b**". O uso de instruções virtuais pode, por isso, contribuir para tornar o programa *assembly* mais curto e mais legível.

No texto que se segue faz-se uma apresentação pormenorizada do reportório de instruções dividido por categorias funcionais. As instruções apresentadas podem

ser completamente nativas, completamente virtuais ou possuírem um código de operação (*opcode*) nativo mas modos de endereçamento enriquecidos. Por forma a distinguir facilmente as instruções pertencentes a cada caso optou-se por escrever as pertencentes ao primeiro em letra normal, as pertencentes ao segundo em **negrito** (*bold*) e as pertencentes ao último em letra normal com as partes não nativas em **negrito**. Em todas as instruções que caem nesta última categoria apenas um dos operandos é virtual e apresenta uma das formas seguintes:

Imm
: representa um valor constante com 16 ou 32 bits; o primeiro caso corresponde à instrução nativa

Src
: representa um registo da CPU ou uma constante (imediato); o primeiro caso corresponde à instrução nativa

Addr
: representa um dos modos de endereçamento definidos na Tabela VI; a instrução nativa corresponde ao caso em que **Addr** tem a forma **Imm(Reg)** e **Imm** é uma quantidade de 16 bits.

Organização da Memória

À semelhança do que tipicamente acontece nos sistemas baseados no processador MIPS o PCSPIM divide a memória em três segmentos: texto, dados e *stack*. O segmento texto (**.text**) começa no endereço 0x400000 (o código de utilizador é carregado, por omissão, a partir do endereço 0x400020) e é usado para suportar as instruções do programa. O segmento de dados (.data) começa no endereço 0x10000000 e é usado para armazenar os dados do programa.
Este segmento está por sua vez subdividido em duas partes. A primeira (a partir do endereço 0x10010000) é usada para alocar objectos cujos comprimento e endereço são conhecidos aquando da assemblagem. Imediatamente a seguir são colocados os objectos alocados dinamicamente usando a chamada ao sistema sbrk8. Por fim o segmento *stack* (**.stack**) começa no topo da memória virtual (0x7fffffff) e cresce no sentido contrário ao dos endereços.
Esta divisão da memória em três segmentos não é a única possível. Contudo, ela possui duas características importantes: os dois segmentos dinâmicos estão o mais afastados possível um do outro e podem crescer no sentido de ocuparem a totalidade do espaço de endereçamento da memória.
Além dos três segmentos descritos existem mais dois próprios do *kernel* do sistema. Eles são os segmentos .ktext e .kdata que começam nos endereços 0x80000080 e 0x9000000 respectivamente.

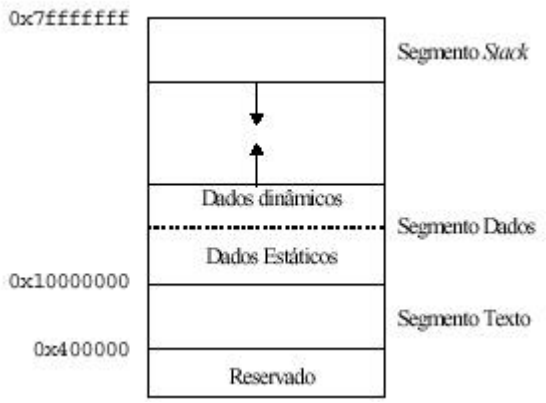


Figura 6 - *A organização da memória.*

Se pensarmos em termos de um programa em C esta é a área de memória onde são alocadas as variáveis definidas usando a função malloc.

Notas

Descrição do MIPS R2000

1. O Processador
2. Organização da memória
3. Características MIPS / SPIM
4. Construção de programas

1. O Processador

1.1 Unidades

O processador MIPS é constituído por:

CPU - unidade principal;

Coprocessadores - unidades auxiliares:

Nº 0 - gestão de excepções, memória virtual;

Nº 1 - unidade de cálculo em vírgula flutuante.

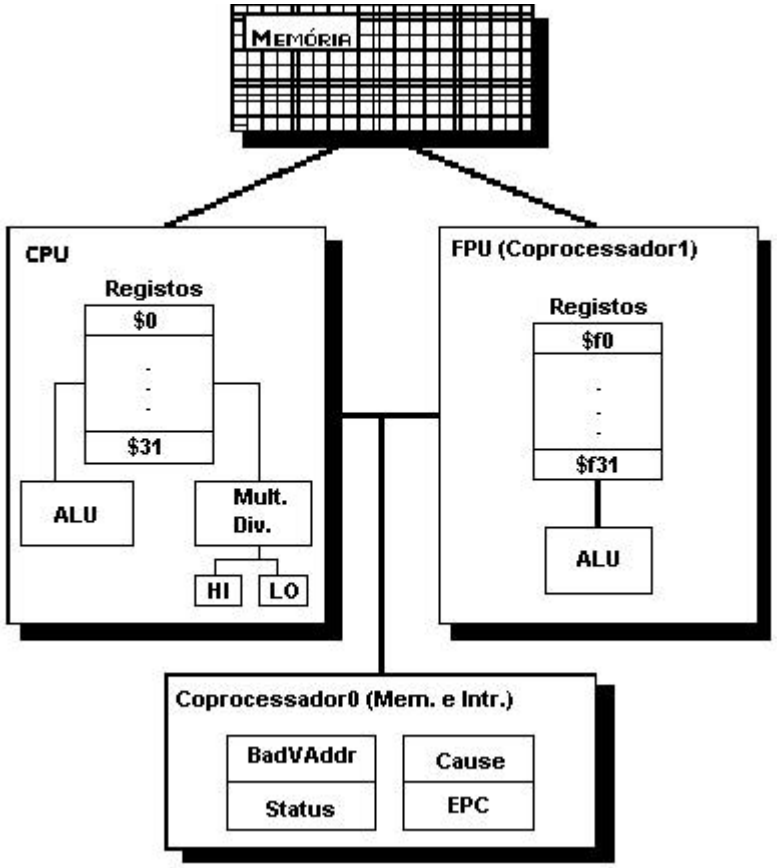


Figura: MIPS R2000 - CPU e FPU

1.2 Registradores
Cada unidade tem os seus próprios registradores.

CPU
A tabela apresenta a convenção para o uso dos registradores que garante a compatibilidade entre módulos desenvolvidos por diferentes programadores.

Registo	Número	Uso
ra	31	Endereço de retorno de procedimentos
zero	0	Constante 0
at	1	Reservado ao assembler
v0-v1	2-3	Expressões e resultados de funções
a0-a3	4-7	Argumentos 1 - 4
t0-t7	8-15	Valores temporários a descartar
s0-s7	16-23	Valores temporários a preservar
t8-t9	24-25	Valores temporários a descartar

k0-k1	26-27	Reservados ao S.O. (kernel)
gp	28	Apontador para a área global de dados
sp	29	Apontador para a pilha
fp	30	Apontador para a estrutura
ra	31	Endereço de retorno

Tabela: Convenções para uso dos registradores do MIPS
Há ainda dois registradores especiais associados às operações de multiplicação / divisão. Estes só podem ser acedidos através de 4 instruções específicas que transferem informação entre esses registradores (HI e LO) e qualquer um dos 32 registradores ($0 a $31).

Coprocessador 0
O SPIM simula os registradores do coprocessador 0 necessários para a gestão de exceções.

Registo	Número	Uso
BadVAddr	8	Endereço de memória onde ocorreu a excepção
Status	12	Máscara de interrupções e bits de activação
Cause	13	Tipo de excepção e bits de interrupção pendentes
EPC	14	Endereço da instrução que desencadeou a excepção

Tabela: Nomes dos registradores do coprocessador 0 do MIPS

Coprocessador 1 - Processador de vírgula flutuante

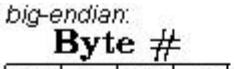
O acesso aos registradores do coprocessador 1 para cálculos em vírgula flutuante, precisão simples ou dupla, é através do número do registo precedido pela letra f. Note que para manipular valores v.f. prec. simples estão disponíveis os 32 registradores. Para manipular valores v.f. prec. dupla apenas 16 registradores estão disponíveis ($f0, $f2, $f4, ..., $f30). Internamente cada um destes registradores está associado ao seguinte para formarem um "registo" de 64 bits.

Registo	Número	Usos
$f0 - $f31	0 - 31	32 registradores genéricos para guardar valores em vírgula flutuante - precisão simples (32 bits)
$f0 - $f30	0 - 30	16 pares de registradores (ex. $0 e $1) para guardar valores em vírgula flutuante - precisão dupla (64 bits)

Tabela: Nomes dos registradores do coprocessador 1 do MIPS

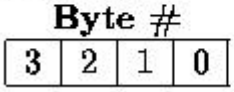
1.3 Ordem dos Octetos
O SPIM numera os octetos de uma palavra obedecendo a uma sequência que depende da convenção usada pelo processador que corre a simulação.

- *big-endian*:



O mais significativo à frente

- *little-endian*:



O menos significativo à frente

2. Organização da Memória

A organização da memória no MIPS é a seguinte:

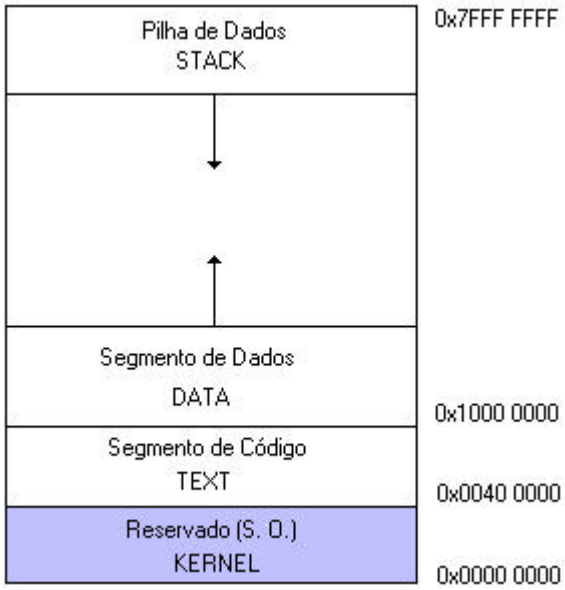


Figura: Organização da Memória

base - segmento de texto (0x0040 0000) contém as instruções.
intermédia - segmento de dados (0x1000 0000)

- estática - objectos cujo tamanho e endereço é conhecido no momento da compilação ou ligação.
- dinâmica - reservada dinamicamente na invocação de funções, (p. ex. malloc).

topo - a pilha inicia-se em (0x7fff ffff) e cresce em direcção ao segmento de dados.

3. Características MIPS / SPIM

3.1 MIPS

3.1.1 Complexidade

- possui modos de endereçamento limitados;
- complexidade remetida para os programadores.

3.1.2 Ao nível da Máquina Virtual

- A complexidade é ocultada através da definição de uma máquina virtual e respectivo assembler
- não existem instruções retardadas (delayed);
    - nos saltos (delayed branch);
    - na transferência de valores de e para a memória (delayed loads).
- o conjunto de instruções é enriquecido com pseudo-instruções.

3.2. SPIM

O SPIM S20 é um simulador dos processadores RISC - MIPS R2000.

3.2.1 Possibilidades

- ler e executar programas descritos em linguagem assembly;
- ler e executar ficheiros de código MIPS executável;
- estabelecer um interface básico com o computador hospedeiro;
- monitorizar e depurar (debug) programas.

3.2.2 Razões de Utilização

Equipamento

- hardware específico nem sempre disponível;
- arquitecturas cada vez mais complexas;
- MIPS R2000 é um protótipo de máquina RISC.

Ambiente de Desenvolvimento

- interfaces mais elaborados na versão X (XSPIM);
- melhor detecção de erros;
- facilmente modificável;
- estudo de novos processadores e instruções;
- previsão de comportamento dos programas;

3.2.3 Interface

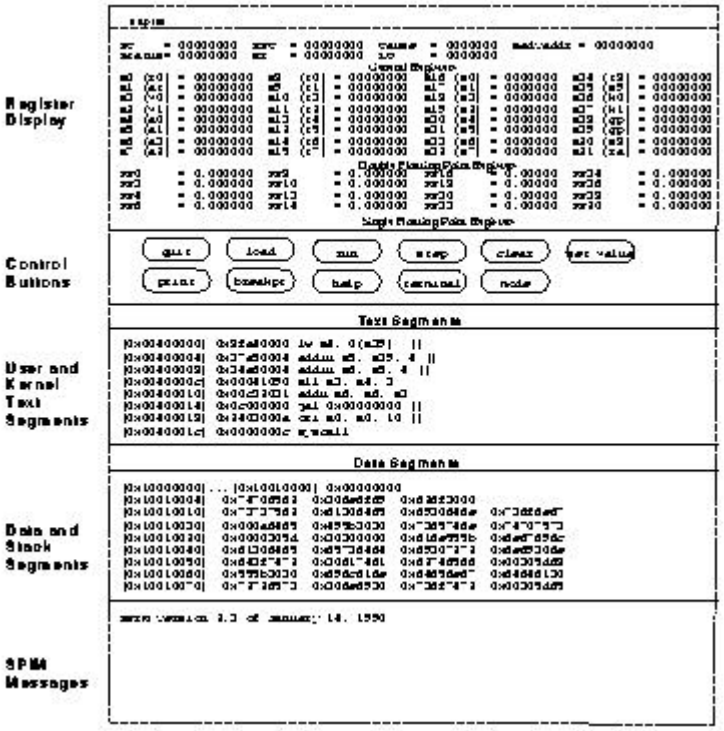
O simulador tem um interface simples spim e um mais sofisticado xspim.

spim(1) spim(1)

NAME
xspim - A MIPS R2000/R3000 Simulator

SYNTAX
xspim [-asm/-bare -trap/-notrap -quiet/-noquiet
-mapped_io/-nomapped_io -file file -execute file
-stext size -sdata size -sstack size -sktext size -skdata
size -ldata size -lstack size -lkdata size
-hexgpr/-nohexgpr -hexfpr/-nohexfpr]

DESCRIPTION
SPIM S20 is a simulator that runs programs for the MIPS R2000/R3000 RISC computers. (For a description of the real machines, see Gerry Kane and Joe Heinrich, MIPS RISC



Architecture, Prentice Hall, 1992.) SPIM can read and immediately execute files containing assembly language or MIPS executable files. SPIM is a self-contained system for running these programs and contains a debugger and interface to a few operating system services.

3.2.4 Interface XSPIM

Na versão X do spim os painéis estão continuamente a ser actualizados, excepto durante a execução dos programas.

Figura: Interface XSPIM.

3.2.5 Funcionalidade

Comando	Descrição
Quit	Abandonar o simulador
load	Ler um ficheiro assembly ou executável para a memória.
run	Iniciar a execução do programa.
step	Executar passo-a-passo.
clear	Reiniciar os valores dos registradores ou da memória.
set value	Atribuir valores aos registradores ou à memória.
print	Mostrar o valor dum registo ou duma posição de memória.
breakpoint	Definir, apagar ou mostrar pontos de paragem.
help	Mostrar a mensagem de ajuda.
terminal	Mostrar ou esconder a consola do programa.
mode	Estabelece os modos de operação do simulador

3.3 SPIM versus MIPS

Diferenças

- tempos das instruções - diferentes do real.

- memória - não existem caches nem atrasos no acesso.
- operações - vírgula flutuante, não duram o tempo previsível.

- Semelhanças
  - pseudo-instruções - expandidas para várias instruções puras.

-

4. Construção de Programas

O SPIM simula a máquina virtual do MIPS seguindo as convenções da sua linguagem Assembler e dos seus compiladores.

4.1 Sintaxe

comentários - iniciam-se com (#) ;

- identificadores - cadeia de caracteres alfanuméricos (_) e (.), não iniciada por um algarismo;
- instruções - símbolos reservados que não podem ser usados como identificadores);
- etiquetas - identificadores declarados no início de uma linha e terminados por (:);
- cadeia de caracteres - definidas entre aspas (") seguindo a convenção:
  - \n - nova linha;

    \t - tabulador;

    \" - aspas;

    \\ - barra invertida.

4.2 Directivas

O SPIM contempla um subconjunto de directivas do assembler do MIPS:

4.2.1 Segmentos e Símbolos

O SPIM não faz qualquer distinção entre as directivas (.data, .rdata, and .sdata).

.text <addr>	Se estiver presente o argumento addr o segmento de texto inicia no endereço addr.
.ktext <addr>	Os próximos itens (instruções ou palavras) são incluídos no segmento de texto. Idêntica à anterior para o segmento de texto do Kernel.
.data <addr>	Os próximos itens são incluídos no segmento de dados. Se estiver presente o argumento addr o segmento de dados inicia no endereço addr.
.kdata <addr>	Idêntica à anterior para o segmento de dados do Kernel.
.globl sym	Declara sym como um símbolo global que também pode ser referenciado noutros ficheiros.

.extern sym size	Declara a estrutura de dados declarada sym como um símbolo global que tem n size octetos de comprimento. Sugere que aquela estrutura seja colocada numa posição de memória acessível através do registo $gp.

4.2.2 Dados

.align n	Alinhamento dos dados em quadros de 2n octetos. 1 - meia palavra (dois octectos); 2 - palavra (quatro octetos); 0 - desliga o mecanismo de alinhamento até uma próxima directiva .data ou .kdata
.ascii str	A cadeia de caracteres não terminada pelo caracter zero é guardada sequencialmente na memória.
.asciiz str	Idêntica à anterior com a cadeia de caracteres terminada pelo caracter zero.
.byte b1,...,bn	Guarda os n octectos sequencialmente na memória.
.space n	Aloca n octectos de espaço no segmento de dados.
.half h1,...,hn	Guarda os n valores de 16-bits (meia palavra) sequencialmente na memória.
.word w1,...,wn	Guarda os n valores de 32-bits (palavras) sequencialmente na memória.
.float f1,...,fn	Guarda os n valores em vírgula flutuante (precisão simples).
.double d1,...,dn	Guarda os n valores em vírgula flutuante (precisão dupla).

4.3 Exemplo

```
            .data
item:       .word 1
            .text
main:       lw $t0, item        # coloca 1 no registo $t0
```

Referências

[1] A. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions”, Phil. Trans. Roy. Soc. London, vol. A247, pp. 529-551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp. 68-73.

[3] de Nikos Drakos, Computer Based Learning Unit, University of Leeds.