André Oliveira Cunha

2. A máquina Virtual MARS e as Variáveis de Programa[1]

Objetivos

Após esse lab, você deverá ser capaz de verificar:

- Como as instruções sintéticas ou pseudoinstruções MIPS podem ser reconhecidas no código; • Como as instruções sintéticas se expandem para sequências de instruções nativas de máquina; • Como o MIPS endereça a memória de dados.

O MIPS é uma arquitetura RISC "típica": possui um conjunto de instruções simples e regular, com apenas um endereçamento de memória modo (base mais deslocamento) e instruções são de tamanho fixo (32 bits). Surpreendentemente, não é muito simples escrever programas de montagem para MIPS (e para qualquer máquina RISC).

A razão é que a programação não deve ser feita em linguagem *assembly*. A instrução conjuntos de máquinas RISC foram projetados de tal forma que:

- Simplifique o trabalho dos criadores de compiladores
- Permita uma implementação em hardware muito eficiente

Em algumas situações particulares, o programador poderá utilizar uma instrução em linguagem *assembly* MIPS que não têm correspondência direta com uma instrução de máquina da CPU MIPS. Estas instruções que pertencem ao conjunto de instruções para a máquina virtual MIPS e que não pertencem ao conjunto de instruções da máquina MIPS, implementadas pela CPU MIPS, são chamadas de instruções **sintéticas** ou **pseudoinstruções**. O único objetivo destas instruções é "simplificar" a programação. Não há representação binária direta para as pseudoinstruções. Na verdade, o montador as substitui por uma ou mais instruções nativas equivalentes e que executam a lógica desejada.

Um exemplo bastante usado em programas é a carga de um endereço do segmento de dados em um registrador da CPU, como no caso da pseudoinstrução *la $t0, str1*. Esta instrução carrega o endereço reservado para *str1* (*la=load address*) no segmento de dados para o registrador *$t0*. Observe que se *str1* é um endereço, ele é representado em 32 bits e, portanto, não há como a instrução citada ser uma instrução nativa. Por quê?

ETAPA 1 – Usando instruções sintéticas ou pseudoinstruções

Este exercício fará com que você se familiarize com instruções sintéticas no conjunto de instruções da máquina MIPS virtual. Existem muitas instruções sintéticas, mas só exploraremos algumas delas durante este laboratório. Veremos mais deles enquanto continuamos a explorar o conjunto de instruções em outras sessões de laboratório. A arquitetura MIPS é do tipo *load*/*store* e por isso:

- Todas as operações são realizadas em registros. O acesso à memória (de dados) é feito apenas pelas instruções *load* e *store*. Não há instrução aritmética ou lógica que tenha parte de operandos em registradores e parte na memória: operandos para tais instruções estão sempre em registradores da CPU;
- Todas as instruções aritméticas e lógicas possuem três operandos, um registrador de destino que é sempre listado imediatamente após o nome da instrução. Existem sempre dois registradores de origem. A instrução de máquina *add $t0, $t1, $t2* adiciona os registros de origem $t1 e $t2 e armazena o resultado no registro de destino $t0.

Considere o programa MIPS a seguir. Substitua a variável X pelos 3 últimos dígitos de sua

[*www.cs.iit.edu/~**virgil**/cs470/**Labs**/*]

```
 .data 0x10010000
var1: .word 0x55 # var1 is a word (32 bit) with the initial value 0x55 var2: .word 0xaa

 .text
 .globl main
main: addu $s0, $ra, $0 # save $31 in $16
li $t0, X
move $t1, $t0
la $t2, var2
lw $t3, var2
sw $t2, var1

# restore now the return address in $ra and return from main
addu $ra, $0, $s0 # return address back
jr $ra # return from main
```

Salve o programa anterior como *lab2.asm* e o execute no MARS. Identifique instruções sintéticas e preencha a tabela a seguir. Lembre-se de que as instruções sintéticas são aquelas que estão no conjunto de instruções da máquina virtual, mas não no conjunto de instruções nativas (ou seja, não no conjunto de instruções da máquina nua). Você pode reconhecê-los porque eles são diferentes na memória do arquivo de origem. Às vezes há uma substituição de um para um (uma instrução sintética é substituída por uma instrução nativa), outras vezes várias instruções nativas são usadas para substituir uma instrução sintética. Importante: o fato de que um nome de registro (como $ t0) é substituído por um número de registro ($8) não denota uma instrução sintética!

Endereço	Instrução sintética	Instruções nativas	Efeito
0x00400004	li $t0, 756	addiu $8, $0, 0x000002f4	$t0 <- $0 + 0x000002f4
0x00400008	move $t1, $t0	addu $9, $0, $8	$t1 <- 0x000002f4
0x0040000c	la $t2, var2	lui $1, $0x00001001 ori $10, $1, $0x00000004	$t2 <- 0x10010004
0x00400014	lw $t3, var2	lui $1, $0x00001001 lw $11, 0x00000004 ($1)	$t3 <- 0x000000aa
0x0040001c	sw $t2, var1	lui $1, 0x00001001 sw $10, 0x00000000 ($1)	0x10010000 <- $t3

Na coluna "Efeito", indique qual é operação realizada em cada instrução nativa.
Ex: *add $t0, $t1, $t2* Efeito # *$t0 <- $t1 + $t2*

ETAPA 2 – Carregando um endereço em um registrador

A instrução *lui* é usada para carregar uma constante de 32 bits em um registrador. Como todas as

instruções são do mesmo tamanho (32 bits), não há como uma instrução inicializar um registrador com um valor imediato de 32 bits, pois a instrução "não caberia" nos 32 bits disponíveis para codificar a instrução de máquina.

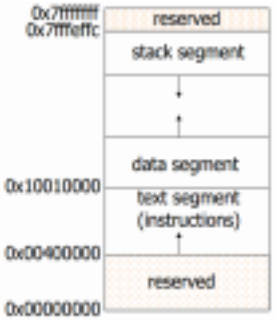
Sendo assim, foi criado um mecanismo para permitir a carga carregar uma constante de 32 bits em um registrador, baseado numa execução em duas etapas. Na primeira etapa, o mecanismo usa uma instrução do tipo *lui* para carregar 16 bits na parte superior do registrador (geralmente usa-se o registrador $1, que é um registro reservado para o montador). Estes 16 bits correspondem à parte "superior" da constante a ser armazenada no registrador. Na segunda etapa, uma instrução *ori* é utilizada para que se completem os 16 bits "inferiores" do valor da constante a ser armazenada no registrador.

Na tabela abaixo, insira a constante que você encontra com os primeiros *lui* e *ori* em seu programa, em formato hexadecimal. Explique o que estas constantes representam no seu programa.

Instrução	Constante	Significado da constante
lui	0x00001001	Ocorre o deslocamento para a esquerda dos 4 bits mais significativos de var2
ori	0x00000004	São os 4 bits menos significativos do endereço de var2

ETAPA 3 – Endereçamento em MIPS

A única maneira pela qual a CPU pode acessar a memória no MIPS é por meio de instruções do tipo *load/store*. Existe apenas um modo de endereçamento para os dados: base + deslocamento. Ter apenas um modo de endereçamento faz parte da filosofia RISC de manter as instruções simples, permitindo assim uma estrutura de controle simples e uma execução eficiente. A figura abaixo mostra o layout da memória para sistemas MIPS. O espaço do usuário é reservado para programas do usuário:



O espaço do *kernel* não pode ser acessado diretamente pelos programas do usuário e é reservado para o uso do Sistema Operacional. Note que o espaço do *kernel* é metade do espaço de endereçamento disponível para o programa do usuário (contém aqueles endereços que possuem o bit 1 mais significativo). O espaço do *kernel* pode ser organizado da mesma maneira que o espaço do usuário (com um segmento de texto, segmento de dados, pilha), embora isso não seja mostrado na figura.

O segmento de texto contém o código do usuário que o sistema está executando. O segmento de dados tem duas seções:

• Dados alocados de forma estática, que contém o espaço para variáveis estáticas e globais; • Dados alocados de forma dinâmica, que é o espaço alocado para objetos de dados em tempo de execução (geralmente usando instruções do tipo *malloc*)

Tarefas

1) Criar um programa em assembly MIPS que tenha as seguintes características:

- Reserve espaço na memória para quatro variáveis chamadas *var1* a *var4* do tamanho da palavra.
- Defina valores iniciais para estas variáveis
- Reserve espaço na memória para duas variáveis chamadas primeiro e ultimo de tamanho do byte. O valor inicial de primeiro deve ser a primeira letra do seu primeiro nome e o valor inicial do último deve ser a primeira letra do seu último nome.
- O programa troca os valores das variáveis na memória: o novo valor de *var1* será o valor inicial de *var4*, o novo valor de *var2* será o valor inicial de *var3*, *var3* receberá o valor inicial de *var2*, e finalmente o *var4* obterá o valor inicial de *var1*.

Você deve priorizar o uso dos registradores $t0 a $ t8 em seu programa. Você pode usar o conjunto de instruções estendido com pseudoinstruções MIPS. Comente cada linha no programa com comentários indicando o que a instrução MIPS faz.

2) Encontre o valor do deslocamento usado para determinar o endereço de cada variável no seu programa desde o início do segmento de dados. O deslocamento será a distância em bytes entre o início do segmento de dados e o local onde a variável é armazenada. Compare estes valores de deslocamento com os que aparecem na tabela de labels do seu programa no MARS. Use as chamadas de sistema (*syscall*) para verificar o que está armazenado na memória no segmento de dados (basta imprimir o que está na memória, comparando com os valores armazenados na tabela "Data segment" do MARS).

variável	endereço	deslocamento
var1	0x10010000	0 bytes
var2	0x10010004	4 bytes
var3	0x10010008	8 bytes
var4	0x1001000c	12 bytes
primeiro	0x10010010	16 bytes
ultimo	0x10010011	17 bytes

3) Quantas instruções assembly e quantas instruções de máquina foram utilizadas no seu programa? Como você determina o tamanho do programa, em Bytes, a partir destas informações?

R: Ao compilar e executar o programa é possível perceber que 8 instruções sintéticas e 16 instruções de máquina. Sabendo que cada instrução tem 4 bytes, tamanho do programa será 64 bytes (16*4).

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	11: lw $t1, var1
	0x00400004	0x8c290000	lw $9,0x00000000($1)	
	0x00400008	0x3c011001	lui $1,0x00001001	12: lw $t2, var4
	0x0040000c	0x8c2a000c	lw $10,0x0000000c($1)	
	0x00400010	0x3c011001	lui $1,0x00001001	13: sw $t2, var1
	0x00400014	0xac2a0000	sw $10,0x00000000($1)	
	0x00400018	0x3c011001	lui $1,0x00001001	14: sw $t1, var4
	0x0040001c	0xac29000c	sw $9,0x0000000c($1)	
	0x00400020	0x3c011001	lui $1,0x00001001	16: lw $t1, var2
	0x00400024	0x8c290004	lw $9,0x00000004($1)	
	0x00400028	0x3c011001	lui $1,0x00001001	17: lw $t2, var3
	0x0040002c	0x8c2a0008	lw $10,0x00000008($1)	
	0x00400030	0x3c011001	lui $1,0x00001001	18: sw $t2, var2
	0x00400034	0xac2a0004	sw $10,0x00000004($1)	
	0x00400038	0x3c011001	lui $1,0x00001001	19: sw $t1, var3
	0x0040003c	0xac290008	sw $9,0x00000008($1)	