UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

116394 ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Trabalho I: Simulador MIPS - 1a Parte

OBJETIVO

Este trabalho consiste na implementação da primeira parte de um simulador da arquitetura MIPS em linguagem de alto nível (C/C++). Este primeiro trabalho aborda a carga de código binário e dados gerados a partir do montador MARS. O simulador deve ler arquivos binários contendo o segmento de código e o segmento de dados para sua memória e exibir seu conteúdo na tela. Nomear os arquivos de entrada como *text.bin* e *data.bin*.

DESCRIÇÃO

Geração dos arquivos

As instruções e dados de um programa MIPS para este trabalho devem vir necessariamente de arquivos montados pelo MARS. Para ilustrar o procedimento, considere o exemplo a seguir:

```
.data
primos: .word 1, 3, 5, 7, 11, 13, 17, 19
size:   .word 8
msg:    .asciiz "Os oito primeiros numeros primos sao : "
space:  .ascii " "

.text
        la $t0, primos          #carrega endereço inicial do array
        la $t1, size            #carrega endereço de size
        lw $t1, 0($t1)          #carrega size em t1
        li $v0, 4               #imprime mensagem inicial
        la $a0, msg
        syscall

loop:   beq $t1, $zero, exit    #se processou todo o array, encerra
        li $v0, 1               #serviço de impressão de inteiros
        lw $a0, 0($t0)          #inteiro a ser exibido
        syscall
        li $v0, 4               #imprime separador
        la $a0, space
        syscall
        addi $t0, $t0, 4        #incrementa indice array
        addi $t1, $t1, -1       #decrementa contador
        j loop                  #novo loop
exit:   li $v0, 10
        syscall
```

Montagem do programa

Antes de montar o programa deve-se configurar o MARS através da opção:

Settings->Memory Configuration, opção Compact, Text at Address 0

Ao montar o programa (F3), o MARS exibe na aba “Execute” os segmentos *Text* e *Data*, apresentados abaixo. O segmento de código (*Text*) deste programa começa no endereço `0x00000000` de memória e se encerra no endereço `0x00000044`, que contém a instrução *syscall*. O segmento de dados começa na posição `0x00002000` e termina na posição `0x000204c`. Verifique a ordem dos caracteres da mensagem *msg* no segmento de dados usando a opção ASCII de visualização.

O armazenamento destas informações em arquivo é obtido com a opção:

File -> Dump Memory...

As opções de salvamento devem ser:

Código:

.text (0x00000000 - 0x00000044) - que é o valor *default* para este exemplo

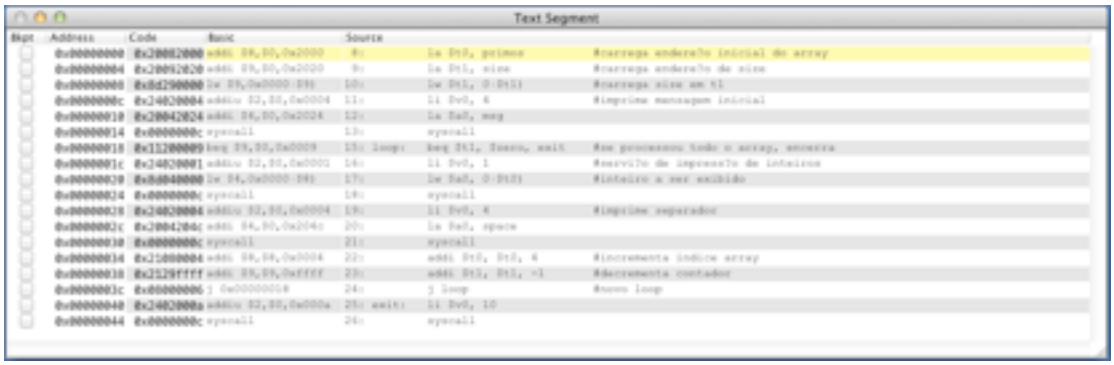
Dump Format: binary

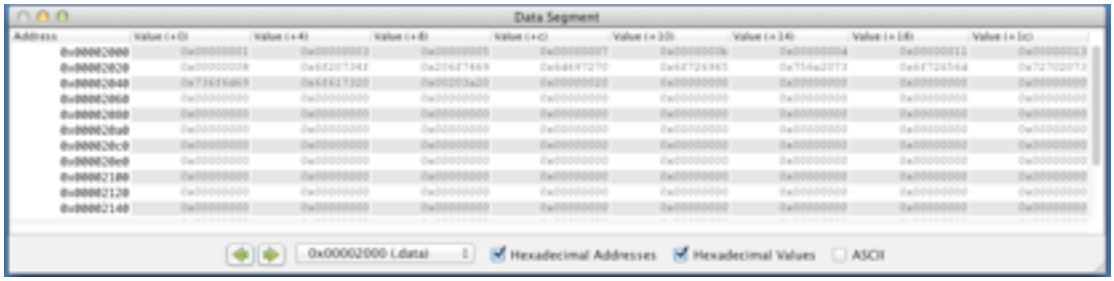
Dados:

.data (0x0000200 - 0x0000204c) - que contém os dados de interesse para o exemplo.

Dump Format: binary

Gere os arquivos com nomes *text.bin* e *data.bin*.





Leitura do código e dos dados

O código e os dados contidos nos arquivos devem ser lidos para a memória do simulador.

A memória deve ser modelada como um arranjo de inteiros:

```
#define MEM_SIZE 4096
int32_t mem[MEM_SIZE];
```

Ou seja, a memória é um arranjo de 4KWords, ou 16KBytes.

A leitura dos arquivos gerados pode ser feita com as operações sobre arquivos de C:

```
FILE * fopen ( const char * filename, const char * mode );
size_t fread ( void * ptr, size_t size, size_t count, FILE *
stream );
int feof ( FILE * stream );
int fclose ( FILE * stream );
```

A leitura deve ser no modo binário.

Acesso à Memória

Desenvolver as funções:

```
int32_t  lw(uint32_t address, int16_t kte);
      => lê um inteiro alinhado - endereços múltiplos de 4
int32_t  lh(uint32_t address, int16_t kte);
      => lê meia palavra, 16 bits - retorna inteiro com sinal
int32_t  lb(uint32_t address, int16_t kte);
      => lê um byte - retorna inteiro com sinal
void  sw(uint32_t address, int16_t kte, int32_t dado);
      => escreve um inteiro alinhado na memória - endereços múltiplos de 4
void  sh(uint32_t address, int16_t kte, int16_t dado);
      => escreve meia palavra, 16 bits - endereços múltiplos de 2
void  sb(uint32_t address, int16_t kte, int8_t dado);
      => escreve um byte na memória
```

Os endereços são todos de *byte*. A operação de leitura de *byte* retorna um inteiro com o *byte* lido na posição menos significativa. A escrita de um *byte* deve colocá-lo na posição correta dentro da palavra de memória.

Verificação do Simulador

1. Criar um código ASM no MARS com *arrays* de *byte*, *half-word* e *word*.
2. Salvar em arquivos binárias as instruções e dados.
3. Ler as instruções e dados dos arquivos para a memória do simulador.
4. Imprimir os dados (*byte, half-word e word*) e as instruções em formato hexadecimal e comparar com os dados gerados pelo MARS.

Entrega

Entregar:
- relatório da implementação:
    - descrição do problema
    - descrição sucinta das funções implementadas
    - testes e resultados
- o código fonte do simulador, com a indicação da plataforma utilizada:
    - qual compilador empregado
    - sistema operacional
    - IDE (Eclipse, XCode, etc)

Entregar no Moodle em um arquivo compactado.