Universidade de Brasília
Departamento de Ciência da Computação

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma A 2018/1
Prof. Marcus Vinicius Lamar
Equipes de até 4 pessoas.
Entrega do relatório (pdf) e fontes em um arquivo zip pelo Moodle até às 23h55 do dia 30/04/2018

Laboratório 1
- Assembly RISC-V –

Objetivos:

- Familiarizar o aluno com o Simulador/Montador Rars;
- Desenvolver a capacidade de codificação de algoritmos em linguagem Assembly;
- Desenvolver a capacidade de análise de desempenho de algoritmos em Assembly;

(1.0) 1) Simulador/Montador Rars

Faça o download e deszipe o arquivo Lab1.zip disponível no Moodle. Serão criados 3 diretórios.

(0.0) 1.1) No diretório System_Rars, abra o programa `sort.s` no Rars. Dado o vetor: V[10]={5,8,3,4,7,6,8,0,1,9}, ordená-lo em ordem crescente e contar o número de instruções por tipo, por estatística e o número total exigido pelo algoritmo. Qual o tamanho em bytes do código executável?

(0.2) 1.2) Compare o desempenho dos processadores RISC-V e MIPS, para implementações semelhantes com mesma frequência de *clock* e mesma CPI, para este algoritmo de ordenação. É possível melhorar o desempenho do RISC-V? Quais as suas sugestões (sem mudar o algoritmo de ordenação nem o hardware)? Qual o fator de desempenho obtido?

(0.8) 1.3) Considere a execução deste algoritmo em um processador RISC-V com frequência de *clock* de 50MHz que necessita 1 ciclo de *clock* para a execução de cada instrução (CPI=1). Para os vetores de entrada de n elementos já ordenados $v_o[n]$={1,2,3,4...n} e ordenados inversamente $v_i[n]${n, n-1, n-2,...,2,1}, obtenha o número total de instruções necessárias, calcule o tempo de execução para n={1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100} e plote esses dados em um mesmo gráfico $n \times t_{exec}$. Comente os resultados obtidos.

(2.0) 2) Compilador GCC

Instale na sua máquina o *cross compiler* RISC-V (MIPS, ARM e x86) gcc disponível no Moodle.

Na linhas de comando do Toochain do gcc a diretiva –S gera o arquivo com o Assembly .s
Para processadores de 32 bits

```
riscv64-unknown-elf-gcc -S -march=rv32imf  -mabi=ilp32f     # RISC-V RV32IMFD
mips-sde-elf-gcc -S -march=mips32                           # MIPS MIPS32
arm-linux-gnueabihf-gcc -S -march=armv7                     # ARM ARMv7
gcc -S –m32                                                 # x86
```

Para processadores de 64 bits

```
riscv64-unknown-elf-gcc -S -march=rv64imafd                 # RISC-V RV64IMAFD
mips-sde-elf-gcc -S -march=mips64                           # MIPS MIPS64
arm-linux-gnueabihf-gcc -S -march=armv8-a                   # ARM ARMv8
gcc -S -march=x86-64                                        # x86-64 (default)
```

(0.0) 2.1) Inicialmente, teste a compilação para Assembly RISC-V com programas triviais em C disponíveis no diretório 'ArquivosC', para entender a convenção do uso dos registradores e memória utilizada pelo gcc para a geração do código Assembly.

(0.5) 2.2) Dado os programas sortc.c e sort2c.c, compile-os, comente os códigos em Assembly obtidos indicando a função de cada uma das diretivas do montador usadas no código Assembly (.file .option .align .globl .type .size .ident etc.). Qual implementação possui melhor desempenho? Por quê?

(0.5) 2.3) Indique as modificações necessárias no código Assembly gerado pelo gcc para que possa ser executado corretamente no Rars.

(1.0) 2.4) Compile os programas sortc.c e sortc2.c e, com a ajuda do Rars, monte uma tabela comparativa com o número de instruções executadas, o tamanho em bytes dos códigos em linguagem de máquina gerados, e os mesmos resultados obtidos no item 1.1) (programa sort.s), para cada diretiva de otimização da compilação {-O0, -O1, -O2, -O3, -Os}. Analise os resultados obtidos.

(2.0) 3) Cálculo das raízes da equação de segundo grau:

Dada a equação de segundo grau: $a.x^2 + b.x + c = 0$

(0.5) 3.1) Escreva um procedimento `int baskara(float a, float b, float c)` que retorne 1 caso as raízes sejam reais e 2 caso as raízes sejam complexas conjugadas, e coloque na pilha os valores das raízes. (Exemplo de passagem de parâmetros e resultados pela pilha)

(0.5) 3.2) Escreva um procedimento `void show(int t)` que receba o tipo (t=1 raízes reais, t=2 raízes complexas), retire as raízes da pilha e as apresente na tela, conforme os modelos abaixo:

Para raízes reais:	Para raízes complexas:
R(1)=1234.0000	R(1)=1234.0000 + 5678.0000 i
R(2)=5678.0000	R(2)=1234.0000 – 5678.0000 i

(0.5) 3.3) Escreva um programa `main` que leia do teclado os valores float de a, b e c, execute as rotinas `baskara` e `show` e volte a ler outros valores.

(0.5) 3.4) Escreva as saídas obtidas para os seguintes polinômios [a, b, c] e, considerando um processador RISC-V de 1GHz, onde instruções da ISA de inteiros são executadas em 1 ciclo de *clock* e as instruções de ponto flutuante em 10 ciclos de *clock*,, calcule os tempos de execução da sua rotina `baskara` (portanto, sem considerar I/O).

a) [1, 0, -9.86960440] b) [1, 0, 0] c) [1, 99, 2459] d) [1, -2468, 33762440] e) [0, 10, 100]

(5.0) 4) Tradução de Programas:

Em toda a história da computação sempre se teve a necessidade de atualização de um sistema, seja pela mudança no software seja pela evolução do hardware. Durante mais de 20 anos, a disciplina de OAC se baseou na arquitetura MIPS para apresentação dos conceitos essenciais de Organização e Arquitetura de Computadores, tendo sido desenvolvida uma plataforma computacional completa usando o kit de desenvolvimento DE2-70 da Altera. A modernização levou a troca da plataforma de hardware para o kit DE1-SoC da Intel e o processador para o RISC-V. Uma das partes mais importantes de um sistema computacional são as rotinas de serviço do sistema operacional, que facilitam muito a vida do programador.

(4.0) 4.1) Dada a rotina de tratamento de exceções (exception handler) para a ISA MIPS e executável no Mars, fornecida no arquivo SYSTEMv63.s, construa a rotina SYSTEMv1.s com os serviços do sistema listados abaixo, traduzindo do Assembly MIPS para a arquitetura RV32IMF.
Dica: Pode retirar toda a parte de identificação e tratamento de interrupções e exceções.

Serviço	$v0/a7	Argumentos	Resultados
Print Integer	1 101	$a0/a0=inteiro $a1/a1=coluna $a2/a2=linha $a3/a3=cores	Imprime o número inteiro complemento de 2 $a0/a0 na posição ($a1/a1,$a2/a2) com as cores $a3/a3={0...0BBGGGRRRbbgggrrr} BGR fundo; bgr frente
Print Float	2 102	$f12/fa0=float $a1/a1=coluna $a2/a2=linha $a3/a3=cores	Imprime o número float em $f12/fa0 na posição ($a1/a1,$a2/a2) com as cores $a3/a3
Print String	4 104	$a0/a0=endereço string $a1/a1=coluna $a2/a2=linha $a3/a3=cores	Imprime a string terminada em NULL (.string) presente no endereço $a0/a0 na posição ($a1/a1,$a2/a2) com as cores $a3/a3
Read Int	5 105		Retorna em $v0/a0 o valor inteiro com sinal lido do teclado.
Read Float	6 106		Retorna em $f0/fa0 o valor float lido do teclado.
Read String	8 108	$a0/a0 endereço do buffer de entrada $a1/a1 número de caracteres máximo	Retorna no endereço $a0/a0 o conjunto de caracteres lidos, terminando com /0.
Print Char	11 111	$a0/a0=char (ASCII) $a1/a1=coluna $a2/a2=linha $a3/a3=cores	Imprime o caractere $a0/a0 (ASCII) na posição ($a1/a1,$a2/a2) com as cores $a3/a3
Exit	10 110		Retorna ao sistema operacional. Na DE2/DE1-SoC trava o processador.
Read Char	12 112		Retorna em $v0/a0 código ASCII do caractere da tecla pressionada

Time	30 130		Retorna o tempo do sistema (número de ciclos de *clock*) $a0/a0 = parte menos significativa $a1/a1 = parte mais significativa (zero na DE2/DE1-SoC)
MIDI Out	31 131	$a0/a0 = pitch $a1/a1 = duração ms $a2/a2 = instrumento (1) $a3/a3 = volume	Gera o som definido e retorna imediatamente
Sleep	32 132	$a0/a0=tempo(ms)	Aguarda $a0/a0 milissegundos
MIDI Out sincrono	33 133	$a0/a0 = pitch $a1/a1 = duração ms $a2/a2 = instrumento (1) $a3/a3 = volume	Gera o som definido e retorna apenas após o término
print integer hexadecimal	34 134	$a0/a0=inteiro $a1/a1=coluna $a2/a2=linha $a3/a3=cores	Imprime, em hexadecimal, o número inteiro de 32 bits em $a0/a0 na posição ($a1/a1,$a2/a2) com as cores $a3/a3
Rand	41 141		$a0/a0 = número randômico de 32 bits
Clear Screen	48 148	$a0/a0 = cor	Limpa a tela com a cor $a0/a0

O serviço original e o serviço original+100 são usados para compatibilizar os serviços originais do Rars com os serviços equivalentes que usam o Bitmap Display e o MMIO Keyboard and Display MMIO Simulator tools.

Na arquitetura RISC-V as rotinas no sistema são chamadas pela instrução `ecall`, que coloca o processador no modo privilegiado S/M (deveria) e executa a rotina de tratamento de exceções/interrupções do sistema definida no registrador `utvec`.

Exemplo de uso de Macro: macros.s

```
.macro M_SetEcall(%label)
       la t0,%label        # carrega em t0 o endereço base das rotinas do sistema ECALL
       csrrw zero,5,t0     # seta utvec (reg 5) para o endereço t0
       csrrsi zero,0,1     # seta o bit de habilitação de interrupção em ustatus (reg 0)
 .end_macro
```

Exemplo de rotinas do sistema ECALL.s

```
.data
.align 2                          # alinha o dado seguinte em word
       E_STR: .string "Isto e so para teste:"
       E_NL:  .string "\n"
.text
ECALL: li t0,104           # carrega t0 = 104
       bne a7,t0,E_FIM     # Se não for o serviço 104 então FIM

       mv t0,a0            # salva argumento a0 em t0
       la a0,E_STR         # ponteiro para a STR
       li a7,4             # serviço original de print string
       ecall

       mv a0,t0            # recupera o número a ser impresso
       li a7,1             # serviço original print int
       ecall

       la a0,E_NL          # ponteiro para a NL
       li a7,4             # serviço original de print string
       ecall

E_FIM: csrrw t0, 65, zero  # le o valor de EPC salvo no registrador uepc (reg 65)
       addi t0, t0, 4      # soma 4 para obter a instrução seguinte ao ecall
       csrrw zero, 65, t0  # coloca no registrador uepc
       uret                # retorna PC=uepc
```

Exemplo de utilização: exemplo_ecall.s

```
.include "macros.asm"      # inclui arquivos de macros no inicio do programa
.data
       STR: .string "Digite um Numero:"

.text
main:  M_SetEcall(ECALL)   # Macro de SetEcall
```

```
        la a0,STR           # Define a0 = endereço STR
        li a7,4             # Define a7 = 4
        ecall               # Chama o serviço original Print String

        li a7,5             # Chama o serviço original Read Int
        ecall

        li a7,104           # Chamada para o novo serviço 104
        ecall

        li a7,10            # Define a7 = 10
        ecall               # chama o serviço de Exit

.include "ECALL.s"          # inclui arquivo ECALLv1.s no final
```

(1.0) 4.2) Demonstre o correto funcionamento das chamadas do sistema implementadas em SYSTEMv1.s executando o programa testeSYSTEMv1.s usando o Bitmap Display tool e o Keyboard and Display MMIO Simulator. Filme a execução e os resultados obtidos.