Universidade de Brasília
Departamento de Ciência da Computação

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma A 2015/2
Prof. Marcus Vinicius Lamar
Equipes de até 5 pessoas.
Entrega do relatório e fontes em um arquivo zip pelo Moodle até dia 07/10/2015

Laboratório 1
- Assembly MIPS –

Objetivos:

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

(3.0) 1) Cálculo das raízes da equação de segundo grau:

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

a) (1.0) 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.
b) (0.5) 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

c) (0.5) 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.
d) (1.0) Escreva as saídas obtidas para os seguintes polinômios [a, b, c] e, considerando um processador MIPS de 1GHz, onde instruções tipo-J são executadas em 1 ciclo, tipo-R em 2 ciclos, tipo-I em 3 ciclos, tipo-FR e FI em 4 ciclos de clock, calcule os tempos de execução da sua rotina baskara (otimizada).
d.1) [1, 0, -9.86960440] d.2) [1, 0, 0] d.3) [1, 99, 2459] d.4) [1, -2468, 33762440] d.5) [0, 10, 100]

(4.0) 2) Implementação de operações aritméticas inteira em software

Considerando que vc possui um processador MIPS de 32bits, implemente os seguintes procedimentos para aritmética inteira de 64 bits. Considere x={$a1,$a0} e y={$a3,$a2} números de 64 bits (long long int).

a) (0.5) long long int addl(long long int x, long long int y); # {HI,LO}=x+y
b) (0.5) long long int subl(long long int x, long long int y); # {HI,LO}=x-y
c) (0.5) long long long long int multl(long long int x, long long int y); # {HI,LO,$v1,$v0}=x*y
d) (0.5) {long long int, long long int} divl(long long int x, long long int y); # floor(x/y)={Hi,LO} e (x%y)={$v1,$v0}
e) (1.0) Crie os procedimentos print64(long long int), que apresente o numero de 64 bits {$a1,$a0} na tela, e print128(long long long long int x), que apresente o numero de 128 bits {$a3,$a2,$a1,$a0}.
f) (1.0) Crie um programa main que leia do teclado dois valores long long int, calcule as 4 operações definidas, imprima na tela seus resultados e volte a ler outros valores.

(4.0) 3) Compilador GCC

Instale na sua máquina o *cross compiler* MIPS GCC disponível no Moodle.
Forma de utilização: mips-sde-elf-gcc –S teste.c #diretiva –S para gerar o arquivo Assembly teste.s
Inicialmente, teste com programas triviais em C para entender a convenção utilizada para a geração do código Assembly.

a) (0.5) Crie as versões em C dos programas main dos itens 1.c e 2.f.
b) (1.0) Compile-os para Assembly MIPS e comente o código obtido indicando a função de cada uma das instruções e diretivas do montador usadas no código Assembly.
c) (1.0) Indique as modificações necessárias no código Assembly gerado para que possa ser executado corretamente no Mars.
d) (1.0) Compare o número de instruções executadas (através do Mars) e o tamanho dos códigos em linguagem de máquina (em bytes) gerados pelo compilador com a utilização das diretivas de otimização da compilação {-O0, -O1, -O2 e -O3}.
e) (0.5) Pesquise quais são as otimizações realizadas pelo compilador gcc para cada diretiva. Existem outras diretivas de otimização? Os resultados obtidos conferem com os resultados esperados?