deec.uc

Sistemas de Microprocessadores

DEP. DE ENG.ª ELECTROTÉCNICA E DE COMPUTADORES
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA



Lab 8 Interface entre linguagem C e Assembly para MIPS

Objetivo: Este trabalho prático tem por objetivo fazer a ligação entre a programação em C e Assembly para MIPS. As tarefas devem ser realizadas no servidor "mips.deec.uc.pt" usando o compilador gcc.

Para aceder ao servidor mips.deec.uc.pt recorde as instruções do trabalho laboratorial #1.

1. Compilação de C para Assembly

Na máquina MIPS use o gcc para compilar o ficheiro **lab8_1.c**, distribuído com a ficha de trabalho, de forma a obter o ficheiro em *assembly* após compilação (*flag* –S, lembra-se?). No ficheiro obtido, tente identificar onde estão a ser efetuadas as instruções escritas em C. Use o gcc uma segunda vez para transformar o código em *assembly* num executável (em caso de necessidade reveja a folha do trabalho laboratorial #1).

2. Chamada de funções em assembly a partir do C

Desenvolver um programa integralmente em *assembly* pode ser trabalhoso devido à necessidade de alguns procedimentos de inicialização. <u>De forma a contornarmos esse problema todas as rotinas em *assembly* que iremos desenvolver nesta ficha de trabalho serão chamadas a partir de uma função main() programada em C.</u>

Considere os ficheiros **lab8_2_main.c** e **lab8_2_mul.s.** Abra-os e observe como a função main(), programada em C, chama a função multiplica() programada em assembly. Use os dois ficheiros para criar um executável e teste o seu funcionamento.

Crie agora uma ficheiro **lab8_2_power.s** que implementa uma função power () de forma a obter um novo programa executável que faz a potência de um número (e.g. power (x, y) devolve x^y). Teste o programa implementado.

Lab8 SMP DEEC-FCTUC

3. Função para encontrar letras

Escreva uma função em *assembly* que percorra uma *string* e determine:

(a) o número de ocorrências de um determinado caracter numa string, usando a função FindChar() implementada em assembly e cujo interface seja como mostrado no código abaixo.

(b) o número de palavras dessa *string* que contêm esse determinado caracter, usando a função FindWordsWithChar() implementada em assembly e cujo interface seja como mostrado no código abaixo.

Teste a sua solução usando o ficheiro **lab8 3.c**:

```
#include <stdio.h>

/*Counts the number of times, a given character
   (toFind) appears in a string */
int FindChar(char *ptr, char toFind);

/*In how many words, appears the character (toFind)? */
int FindWordsWithChar(char *ptr, char toFind);

int main ()
{
   char str[]="Sistemas de Microprocessadores Rocks";
   char tf='s';

   printf("Char '%c' appears %d times in the sentence
sentence.\n", tf, FindChar(str,tf));

   printf("Char '%c' appears in %d words in the
sentence\n", tf, FindWordsWithChar(str,tf));

   return 0;
}
```

4. Máscaras

Nas aulas teóricas aprendeu sobre as instruções do MIPS que fazem operações *bitwise* nomeadamente as instruções *and, or, nor, xor* que fazem operações lógicas bit a bit. Estas instruções permitem a implementação de *máscaras*, em que os bits de um registo são processados independentemente até que se obtém um resultado pretendido de transformação desse valor (por exemplo, a colocação de alguns bits a 0 ou a 1, e a manutenção dos restantes bits inalterados).

Tendo em conta estas instruções *bitwise*, faça uma função main() em C que peça ao utilizador um inteiro. Seguidamente passe esse inteiro a uma rotina par ou impar() que

Lab8 SMP DEEC-FCTUC

devolva 0 se o valor for impar e 1 se for par. A função par_ou_impar() deve ser programada em *assembly* tirando partido das instruções *bitwise* para garantir máxima eficiência computacional.

5. Binariza 2 (Extra)

Lembra-se da função "binariza" do trabalho prático número 2? Na altura foi fornecido um ficheiro **main.c** que chamava uma função bin_img(). A missão foi programar bin_img(), criar o respetivo código objeto, e ligá-lo com **main.o** para obter uma aplicação final. Esta função percorria a imagem e colocava os pixéis a preto e branco conforme o valor de limiar. Pode encontrar em anexo o ficheiro **main.c**.

Pretende-se desta vez que programe a função bin_img() diretamente em *assembly*. Teste a sua solução.



Fig. Imagem original e imagem binarizada correspondente

Se eventualmente tiver necessidade de fazer *debug* à sua rotina em *assembly*, poderá sempre utilizar a opção de compilação '-g' (adiciona informação para *debug*) e utilizar o *gdb* para correr o seu código passo a passo e ver o valor dos registos (ver trabalho prático 2). Por exemplo, o comando 'p \$t0' permitir-lhe-á ver o valor do registo \$t0.