universidade de aveiro theoria poiesis praxis

Departamento de Electrónica, Telecomunicações e Informática

Arquitectura de Computadores II

Mestrado Integrado em Engenharia Electrónica e Telecomunicações

Mestrado Integrado em Engenharia de Computadores e Telemática

2º Ano, 2º semestre

Guia das Aulas Práticas

Ano letivo 2012/2013

(janeiro/2013)

José Luís Azevedo, Arnaldo Oliveira, Tomás Oliveira e Silva

Trabalho prático N.º 1

Objetivos

- Conhecer o processo de criação de um programa escrito em *assembly* para correr na placa DETPIC32: compilação, transferência e execução.
- Utilizar os *system calls* disponibilizados na placa DETPIC32.
- Rever os conceitos associados à manipulação de *arrays* de caracteres.

Trabalho a realizar

Parte I

1. Utilizando o editor de texto *gvim*[1], edite e grave o programa de demonstração *assembly* que é apresentado de seguida. Para facilitar a organização dos ficheiros dos vários programas que irão ser feitos ao longo do semestre, sugere-se que seja criado um *folder* (diretório) por trabalho prático, estando o nome do ficheiro relacionado com a alínea a que diz respeito. No nosso caso o ficheiro poder-se-á chamar "prog1.s" (usa-se a extensão ".s" para ficheiros *assembly*) a colocar no *folder* "tp01".

```
#           int main(void)
#           {
#              printStr("AC2 – DETPIC32 primer\n");   // system call
#              return 0;
#           }

            .equ    PRINT_STR,8

            .data
msg:        .asciiz "AC2 - DETPIC32 primer\n"

            .text
            .globl  main
main:       la      $a0,msg
            ori     $v0,$0,PRINT_STR
            syscall                 # printStr("AC2 – DETPIC32 primer\n");
            ori     $v0,$0,0        # return 0;
            jr      $ra
```

2. Compile o programa do ponto 1 introduzindo, na linha de comando (numa janela de terminal do linux), o seguinte comando**:**

   pcompile prog1.s

3. O comando da linha anterior produz os seguintes ficheiros: "prog1.o", "prog1.elf", "prog1.map" e "prog1.hex", sendo os dois primeiros ficheiros binários e os restantes de texto. Analise o conteúdo do ficheiro "prog1.hex" das duas formas seguintes:
   - utilizando o editor de texto *gvim*
   - utilizando o *disassembler hex2asm*, com as seguintes linhas de comando:

     hex2asm prog1.hex (gera o ficheiro "prog1.hex.s")

     gvim prog1.hex.s

   - identifique no ficheiro "prog1.hex.s" os endereços correspondentes aos *labels* **msg** e **main** do programa que editou. Anote no seu *log book* esses endereços.

[1] Embora possam ser usados outros editores de texto, o *gvim* é o recomendado por questões de portabilidade e legibilidade dos ficheiros.

4. Analise o conteúdo do ficheiro "prog1.map" produzido pelo processo de compilação, abrindo-o com o editor de texto *gvim*. Compare os endereços dos *labels* **msg** e **main** com os obtidos no ponto anterior (acrescente "**.globl msg**" ao código anterior para o símbolo "**msg**" passar a aparecer no ficheiro "prog1.map").
5. Transfira o programa "prog1.hex" para a memória FLASH do microcontrolador PIC32 da placa DETPIC32, realizando os seguintes passos:
   - ligue a placa à porta USB do PC
   - introduza o comando: **ldpic32 –w prog1.hex**
   - prima o botão de *reset* da placa DETPIC32 e aguarde que a transferência se processe
   - execute, em linha de comando, o programa **pterm**
6. Execute o programa transferido no ponto anterior premindo novamente o botão de *reset*.
7. Desligue a placa DETPIC32 da porta USB do PC, espere uns segundos, volte a ligar e repita o ponto 6.

Parte II

1. Os programas que se apresentam de seguida exercitam a utilização dos *system calls* disponíveis na placa DETPIC32. Verifique os *system calls* disponibilizados, consultando ou a tabela de referência rápida referida nos elementos de apoio no final deste trabalho prático ou dando uma vista de olhos no ficheiro "/opt/pic32mx/include/detpic32.h". Analise a forma como cada um dos *system calls* deve ser invocado (por motivos históricos, os números dos *system calls* são diferentes dos usados em AC1).
2. Identifique a funcionalidade de cada um dos programas que se seguem e traduza-os para *assembly* do MIPS, usando as convenções de passagem de parâmetros e salvaguarda de registos que estudou em AC1. Compile cada um dos programas *assembly*, do mesmo modo que realizou nos pontos 2 a 5 da Parte I. Transfira o resultado da compilação (ficheiros ".hex") para a placa DETPIC32 e verifique o respetivo funcionamento.

   NOTA: O código que escrever vai ser executado numa arquitetura *pipelined* com *delayed branches*. Ou seja, em todas as instruções que alteram o fluxo de execução (**beq**, **bne**, **j**, **jal**, **jr**, **jalr**) a instrução que vem imediatamente a seguir é sempre executada, independentemente do comportamento da instrução de salto. Apesar disso, não necessita de ter em conta este comportamento, uma vez que o *assembler* efetua, de forma automática, a reordenação das instruções de modo a preencher, sempre que possível o *delayed slot*. Nos casos em que o *assembler* deteta que não pode reordenar as instruções devido a dependência(s) de dados, o *delayed slot* é preenchido com a instrução "**nop**". Este comportamento deve ser verificado através da análise do ficheiro produzido pelo programa **hex2asm** (por exemplo "prog1.hex.s")

```
// ************************************************************************
// Programa 2 – teste do system call "inkey()"
// ************************************************************************
int main(void)
{
   char c;

   while (1)
   {
      while ((c = inkey()) == 0);
      if (c == '\n')
         break;
      printStr("Key pressed\n");
   }
   return 0;
}
```

```
// ************************************************************************
// Programa 3 - teste dos system calls "getChar()" e "putChar()"
// ************************************************************************
int main(void)
{
   char c;

   while (1)
   {
      c = getChar();
      if (c == '\n')
         break;
      putChar(c);
   }
   return 1;
}

// ************************************************************************
// Programa 4 - teste dos system calls de leitura e impressão de inteiros
// ************************************************************************
void main(void)
{
   int value;

   while (1)
   {
      printStr("\nIntroduza um numero (sinal e módulo): ");
      value = readInt10();
      printStr("\nValor lido em base 2: ");
      printInt(value, 2);
      printStr("\nValor lido em base 16: ");
      printInt(value, 16);
      printStr("\nValor lido em base 10 (unsigned): ");
      printInt(value, 10);
      printStr("\nValor lido em base 10 (signed): ");
      printInt10(value);
   }
}

// ************************************************************************
// Programa 5 - teste do system call "readStr()" e manipulação de strings
// ************************************************************************
#define STR_MAX_SIZE 20

char *strcat(char *, char *);
char *strcpy(char *, char *);
int strlen(char *);

int main(void)
{
   static char str1[STR_MAX_SIZE + 1];
   static char str2[STR_MAX_SIZE + 1];
   static char str3[2 * STR_MAX_SIZE + 1];

   printStr("Introduza 2 strings: ");
   readStr( str1, STR_MAX_SIZE );
   readStr( str2, STR_MAX_SIZE );
   printStr("Resultados:\n");
   prinInt( strlen(str1), 10 );
   prinInt( strlen(str2), 10 );
   strcpy(str3, str1);
   printStr( strcat(str3, str2) );
   printInt10( strcmp(str1, str2) );
   return 0;
}
```

Note: a versão do *assembler* que está a ser usada nas aulas práticas não interpreta corretamente o caracter de terminação das strings, '\0'; em *assembly* use, em vez deste caracter, o valor 0 (tal como está no código abaixo).

```
// ***************************************************************************
// String length
// ***************************************************************************
//
int strlen(char *s)
{
   int len;
   for(len = 0; *s != 0; len++, s++);
   return len;
}

// ***************************************************************************
// String concatenate
// ***************************************************************************
//
char *strcat(char *dst, char *src)
{
   char *rp = dst;

   for(; *dst != 0; dst++);
   strcpy(dst, src);
   return rp;
}

// ***************************************************************************
// String copy
// ***************************************************************************
//
char *strcpy(char *dst, char *src)
{
   char *rp = dst;

   for(; (*dst = *src) != 0; dst++, src++);
   return rp;
}

// ***************************************************************************
// String compare.
// Returned value is:
//   < 0  string "s1" is less than string "s2"
//   = 0  string "s1" is equal to string "s2"
//   > 0  string "s1" is greater than string "s2"
// ***************************************************************************
//
int strcmp(char *s1, char *s2)
{
   for(; (*s1 == *s2) && (*s1 != 0); s1++, s2++);
   return(*s1 - *s2);
}
```

Elementos de apoio

- Tabela com resumo do conjunto de instruções da arquitectura MIPS, na versão adaptada a Arquitectura de Computadores II (disponível no site da disciplina).
- Slides das aulas teóricas de Arquitectura de Computadores I.
- David A. Patterson, John L. Hennessy, Computer Organization & Design – The Hardware/Software Interface, Morgan Kaufmann Publishers.