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. Se ainda não o fez, instale as ferramentas de desenvolvimento (veja as instruções no anexo deste guião).
2. Utilizando um editor de texto, 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 criada uma diretoria por trabalho prático, estando o nome do ficheiro relacionado com a alínea a que diz respeito. Neste caso o ficheiro poder-se-á chamar "**`prog1.s`**" (usa-se a extensão "**`.s`**" para ficheiros *assembly*) a colocar na diretoria "**`tp01`**" [1].

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

      .equ     PRINT_STR,8

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

3. Compile o programa *assembly* anterior. Para isso abra um terminal e execute o comando:

   pcompile prog1.s

4. 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.
   a) Observe o conteúdo do ficheiro "prog1.hex"[2]; para isso abra-o com um editor de texto (gedit, gvim, geany, …).
   b) Execute, em linha de comando, o programa **`hex2asm`** (é um *disassembler* que converte o código binário das instruções para mnemónicas *assembly* do MIPS):

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

      De seguida abra, com um editor de texto, o ficheiro "**`prog1.hex.s`**"

   c) Identifique no ficheiro "**`prog1.hex.s`**" os endereços correspondentes aos *labels* **`msg`** e **`main`** do programa que editou.

[1] Pode encontrar no *youtube* numerosos vídeos sobre a utilização do terminal em Linux. A título de exemplo: "Beginner's Guide to the Bash Terminal" - **`https://youtu.be/oxuRxtrO2Ag`**.

[2] **`https://en.wikipedia.org/wiki/Intel_HEX`**

5. Transfira o programa "**prog1.hex**" para a memória FLASH do microcontrolador da placa DETPIC32, realizando os seguintes passos[3]:
   - ligue a placa à porta USB do PC
   - execute o seguinte comando:

     ldpic32 prog1.hex
   - prima o botão de *reset* da placa DETPIC32 e aguarde que a transferência se processe
6. Execute o programa transferido e observe o resultado:
   - execute, em linha de comando, o programa **pterm** [4]
   - prima novamente o botão de *reset*.

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 analisando o ficheiro "**/opt/pic32mx/include/detpic32.h**". Analise a forma como cada um dos *system calls* deve ser invocado.
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*, usando o **pcompile**. Transfira o resultado da compilação (ficheiros "**.hex**") para a placa DETPIC32 (usando o **ldpic32**) e verifique o respetivo funcionamento.

   a) Teste dos system calls "getChar()" e "putChar()".

   O *system call* "**getChar()**" é bloqueante, ou seja, só regressa ao programa chamador quando for premida uma tecla, retornando o respetivo código ASCII.

```
int main(void)
{
  char c;

  do
  {
    c = getChar();
    if( c != '\n' )
      putChar( c );
  } while( c != '\n' );
  return 0;
}
```

   b) Substitua a linha "**putChar(c)**" por "**putChar(c+1)**" e volte a testar o programa.

   Nota: O código *assembly* que escrever vai ser executado numa arquitetura *pipelined* de 5 fases 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

[3] Na fase de aulas em regime remoto e enquanto os grupos não tiverem acesso às placas de trabalho, esta tarefa deverá ser executada pelo docente. Para o efeito, o aluno deverá enviar o ficheiro "*.hex" ao docente que demonstrará os resultados.

[4] Os 3 comandos normalmente usados (**pcompile**, **ldpic32** e **pterm**) podem ser encadeados numa única linha de comando, do seguinte modo (usando como exemplo o ficheiro "prog1.s"):

```
pcompile prog1.s && ldpic32 prog1.hex && pterm
```

as instruções devido a dependência(s) de dados, o *delayed slot* é preenchido com a instrução "**nop**". Este comportamento pode ser observado através da análise do ficheiro produzido pelo programa **hex2asm** (por exemplo "**prog1.hex.s**").

c) Teste do *system call* "**inkey()**".

O *system call* "**inkey()**" não é bloqueante, ou seja, se foi premida uma tecla devolve o respetivo código ASCII, mas se não foi premida qualquer tecla devolve o valor 0 (zero).

```
int main(void)
{
  char c;

  do
  {
    while( (c = inkey()) == 0 );
    if( c != '\n' )
      putChar( c );
  } while( c != '\n' );
  return 0;
}
```

d) Teste dos *system calls* de leitura e impressão de inteiros.

```
int 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);
  }
  return 0;
}
```

Nota: Devido a limitações do compilador usado, nos programas escritos em *assembly* o *label* "**main**" deve ser o primeiro *label* do segmento de código, ou seja, o código das sub-rotinas deve vir a seguir ao código da função "**main()**".

Exercícios adicionais

1. Utilização do system call "**inkey()**" na implementação de um contador up/down de 8 bits. O valor do contador é atualizado a cada 0.5s (aproximadamente) e é mostrado no ecrã, em decimal e em binário (com o system call "**printInt()**"[5]). O estado "up" ou "down" do contador é assegurado por uma máquina de estados simples, com 2 estados, controlada pelas teclas '+' e '-'.

   a) Traduza para *assembly* do MIPS o programa seguinte.

```
void wait(int);

int main(void)
{
  int s = 0;
  int cnt = 0;
  char c;
  do
  {
    putChar('\r');     // Carriage return character
    printInt( cnt, 10 | 3 << 16 );// 0x0003000A: decimal w/ 3 digits
    putChar('\t');     // Tab character
    printInt( cnt, 2 | 8 << 16 ); // 0x00080002: binary w/ 8 bits
    wait(5);
    c = inkey();
    if( c == '+' )
      s = 0;
    if( c == '-' )
      s = 1;
    if( s == 0 )
      cnt = (cnt + 1) & 0xFF;
    else
      cnt = (cnt - 1) & 0xFF;
  } while( c != 'q' );
  return 0;
}

void wait(int ts)
{
  int i;
  for( i=0; i < 515000 * ts; i++ );
}
```

   b) Altere o código C do programa anterior de modo a adicionar a possibilidade de parar o contador (tecla '**S**') ou de reiniciar o seu valor (tecla '**R**'). Reflita essas alterações no programa *assembly* que escreveu na alínea anterior e teste o resultado na placa.

[5] O *system call* **printInt** permite especificar o número mínimo de dígitos com que o valor é impresso. Essa configuração é feita nos 16 bits mais significativos do registo usado para determinar a base da representação (e.g., para a impressão em binário com 4 bits, o valor a colocar no registo **$a1** é **0x00040002**); em linguagem C: **printInt(val, 2 | 4 << 16)**.

2. Manipulação de *strings* e teste do *system call* "**`readStr()`**" [6].

```
#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;
}
```

Cálculo da dimensão de uma *string.*

```
int strlen(char *s)
{
  int len;
  for( len = 0; *s != '\0'; len++, s++ );
  return len;
}
```

Cópia de uma *string*.

```
char *strcpy(char *dst, char *src)
{
  char *p = dst;

  for( ; ( *dst = *src ) != '\0'; dst++, src++ );
  return p;
}
```

Concatenação de 2 *strings.*

```
char *strcat(char *dst, char *src)
{
  char *p = dst;

  for( ; *dst != '\0'; dst++ );
  strcpy( dst, src );
  return p;
}
```

[6] 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 desse carater, o valor 0.

Comparação alfabética de 2 *strings.*

```
// 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 arquitetura MIPS, na versão adaptada a Arquitetura de Computadores II (disponível no *moodle* de AC2).
- Slides das aulas teóricas de Arquitetura de Computadores I.
- David A. Patterson, John L. Hennessy, Computer Organization & Design – The Hardware/Software Interface, Morgan Kaufmann Publishers.

Anexo

Instalação das ferramentas pic32

1) Descarregue do *moodle* de AC2 o *tarball*:
   - **`pic32-32.tgz`** para sistemas de 32 bits (ou sistemas de 64 bits com bibliotecas de 32 e 64 bits)
   - **`pic32-64.tgz`** para sistemas de 64 bits
2) Abra um terminal e execute o comando:

   ```
   sudo tar xzvf TARBALL -C /opt
   ```

   onde **`TARBALL`** é o *path* completo do *tarball*; por exemplo, se descarregou o *tarball* **`pic32-64.tgz`** para o diretório **`Downloads`** deve fazer:

   ```
   sudo tar xzvf ~/Downloads/pic32-64.tgz -C /opt
   ```
3) Com um editor de texto abra o ficheiro **`.bashrc`** (disponível na sua *home directory*) e adicione, no final, as seguintes linhas:

   ```
   if [ -d /opt/pic32mx/bin ] ; then
       export PATH=$PATH:/opt/pic32mx/bin
   fi
   ```
4) Se pretender desinstalar as ferramentas pic32 abra um terminal e execute o comando:

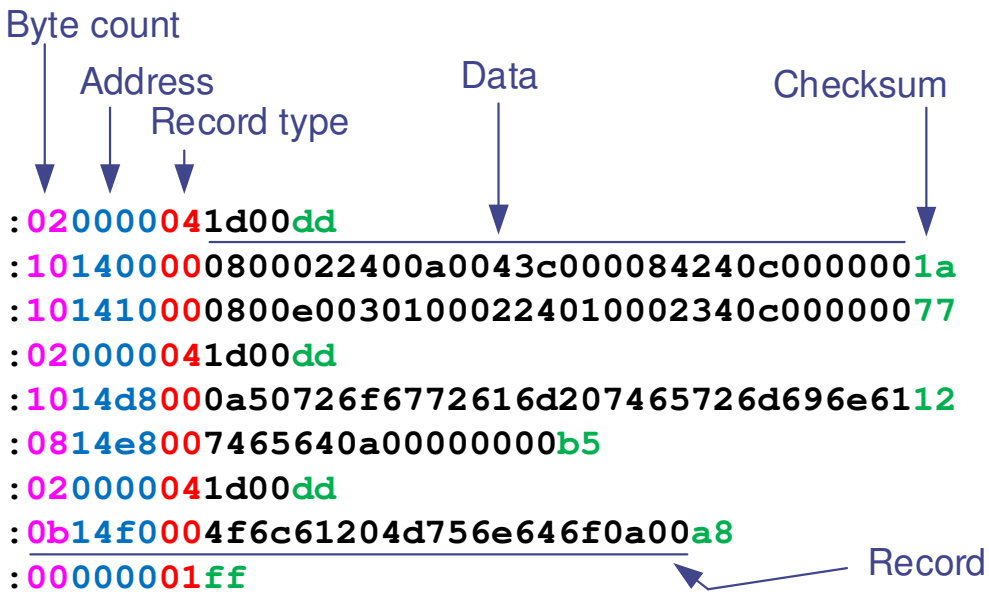
   ```
   sudo rm -rf /opt/pic32mx
   ```

Configuração do computador para comunicar com a placa DETPIC32

1) Adicione o utilizador ao grupo **`dialout`**; para isso abra um terminal e execute o comando:

   ```
   sudo adduser $USER dialout
   ```
2) Para que o comando anterior se torne efetivo faça *reboot* ao PC.

Formato Intel HEX



- **Byte count**: número de bytes do campo de dados
- **Address**: endereço de memória onde é armazenado o primeiro byte do campo de dados; o endereço efetivo é obtido em conjunto com um endereço base especificado anteriormente
- **Record type**:
  - **00** - data record
  - **01** - end-of-file record
  - **04** - extended linear address record (especifica os 16 bits mais significativos do campo de endereço das linhas seguintes)
- **Checksum**: complemento para dois dos 8 bits menos significativos resultantes da soma dos bytes do record; o *checksum* é usado para a deteção de possíveis erros na transmissão dos dados

Descodificação do exemplo:

`:02000004 1d00 dd`

- 2 bytes no campo de dados
- record do tipo 4: o campo de dados contém os 16 bits mais significativos do endereço dos *records* seguintes, ou seja, **`0x1D00`**
- checksum: **`0x100-trunc8(02+00+00+04+1D+00)=0xDD`**, ou seja, complemento para dois da soma, truncada a 8 bits, de todos os bytes do *record*

`:10140000 08000224...0c0000001a`

- 16 bytes no campo de dados
- record do tipo 0: data record
- endereço do primeiro byte: **`0x1D001400`**
- checksum: **`0x100-trunc8(10+14+00+00+08+00+02+...+0C+00+00+00)=0x1A`**