

Aula prática 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 criado um diretório por aula prática. Para este exemplo o ficheiro poder-se-á chamar "**prog1.s**" (usa-se a extensão ".s" para ficheiros *assembly*) e deve ser colocado no diretório "aula1".

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

.equ      PRINT_STR, 8

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

3. Compile o programa *assembly* anterior. Para isso abra um terminal¹ e, no diretório onde guardou o ficheiro com o código fonte (**prog1.s**), 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**"; 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**"

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/oxuRxtR02Ag>.

- c) Identifique no ficheiro "**prog1.hex.s**" os endereços correspondentes aos *labels* **msg** e **main** do programa que editou.
5. Transfira o programa "**prog1.hex**" para a memória FLASH do microcontrolador da placa DETPIC32, realizando os seguintes passos:
- ligue a placa à porta USB do PC
 - execute o seguinte comando:
- ```
ldpic32 prog1.hex (a extensão .hex pode ser omitida)
```
- prima o botão de *reset* da placa DETPIC32 e aguarde que a transferência se processe
6. Execute o programa transferido e observe o resultado; para isso:
- execute, em linha de comando, o programa **pterm**<sup>2</sup>
  - prima novamente o botão de *reset*.

## Parte II

Os programas que se apresentam de seguida exercitam a utilização dos *system calls* disponíveis na placa DETPIC32. Verifique os *system calls* disponibilizados, consultando 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.

1. 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();
 putChar(c);
 } while(c != '\n');
 return 0;
}
```

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

---

<sup>2</sup> 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 && pterm
```

b) 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 (0x00).

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

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

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

```
int main(void)
{
 int value;
 while(1)
 {
 printStr("\nIntroduza um inteiro (sinal e módulo): ");
 value = readInt10();
 printStr("\nValor em base 10 (signed): ");
 printInt10(value);
 printStr("\nValor em base 2: ");
 printInt(value, 2);
 printStr("\nValor em base 16: ");
 printInt(value, 16);
 printStr("\nValor em base 10 (unsigned): ");
 printInt(value, 10);
 printStr("\nValor em base 10 (unsigned), formatado: ");
 printInt(value, 10 | 5 << 16); // ver nota de rodapé 3
 }
 return 0;
}
```

## Notas

1. 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 resultado (*taken/not taken*) da instrução de salto. Apesar disso, não é necessário ter em conta esse comportamento, uma vez que o *assembler* efetua, de forma automática, a reordenação das instruções de modo a preencher, sempre que possível, a *delayed slot*. Nos casos em que o *assembler* deteta que não pode reordenar as instruções devido a dependência(s) de dados, a *delayed slot* é preenchida com a instrução `nop`. Este comportamento pode ser observado através da análise do ficheiro produzido pelo programa `hex2asm` (por exemplo `prog1.hex.s`).
2. 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()`.

<sup>3</sup> 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 especificar 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 )`.

## Exercícios adicionais

1. Utilização do system call `inkey()` na implementação de um contador up/down de 8 bits, módulo 256. 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()`). O estado "up" ou "down" do contador é assegurado por uma máquina de estados simples, com 2 estados, controlada pelas teclas '+' e '-'.

```

#define UP 1
#define DOWN 0

void wait(int);

int main(void)
{
 int state = 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); // wait 0.5s

 c = inkey();
 if(c == '+')
 state = UP;
 if(c == '-')
 state = DOWN;

 if(state == UP)
 cnt = (cnt + 1) & 0xFF; // Up counter MOD 256
 else
 cnt = (cnt - 1) & 0xFF; // Down counter MOD 256
 } while(c != 'q');
 return 0;
}

void wait(int ts)
{
 int i;
 for(i=0; i < 515000 * ts; i++); // wait approximately ts/10 seconds
}

```

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

2. Manipulação de *strings*<sup>4</sup> e teste do *system call* `readStr()`.

```

#define SIZE 20

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

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

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

// Returns the length of string "str" (excluding the null character)
int strlen(char *str)
{
 int len;
 for(len = 0; *str != '\0'; len++, str++);
 return len;
}

// Copy the string pointed by "src" (including the null character) to
// destination (pointed by "dst")
char *strcpy(char *dst, char *src)
{
 char *p = dst;

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

// Concatenates "dst" and "src" strings
// The result is stored in the "dst" string
char *strcat(char *dst, char *src)
{
 char *p = dst;

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

```

---

<sup>4</sup> A versão do *assembler* que está a ser usada nas aulas práticas não interpreta corretamente o carácter de terminação das *strings*, `'\0'`; em *assembly* use, em vez desse carácter, o valor 0 (que é o código ASCII correspondente a `'\0'`).

```
// Compares two strings character by character
// Returned value is:
// < 0 string "str1" is "less than" string "str2" (first
// non-matching character in str1 is lower, in ASCII, than
// that of str2)
// = 0 string "str1" is equal to string "str2"
// > 0 string "str1" is "greater than" string "str2" (first
// non-matching character in str1 is greater, in ASCII, than
// that of str2)

int strcmp(char *str1, char *str2)
{
 for(; (*str1 == *str2) && (*str1 != '\0'); str1++, str2++);
 return(*str1 - *str2);
}
```

### Elementos de apoio

- Tabela com o 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
```

(continua na página seguinte)

## Configuração do computador para comunicar com a placa DETPIC32

- 1) Remover pacote **brltty** (em Ubuntu):

```
sudo apt remove brltty
```

Se der erro a desinstalar, tem que desativar o serviço e para isso deve executar a seguinte sequência de comandos:

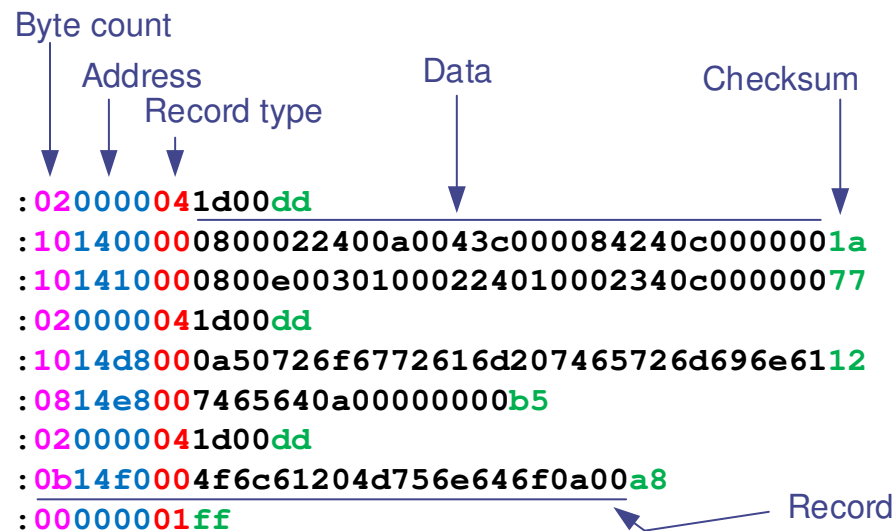
```
systemctl stop brltty-udev.service
sudo systemctl mask brltty-udev.service
systemctl stop brltty.service
systemctl disable brltty.service
```

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

```
sudo adduser $USER dialout
```

- 3) 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

### Decodificação do exemplo:

:020000041d00dd

- 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:  $0x100 - \text{trunc8}(02+00+00+04+1D+00) = 0xDD$ , ou seja, complemento para dois da soma, truncada a 8 bits, de todos os bytes do *record*

:1014000008000224...0c0000001a

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