

AULA PRÁTICA N.º 7

Objectivos:

- Implementação de subrotinas.
- Utilização da convenção do MIPS para passagem de parâmetros e uso dos registos.
- Implementação e utilização da *stack* no MIPS.

Guião:

Neste guião vão ser implementadas algumas funções para manipulação de *strings*, habitualmente disponíveis nas bibliotecas de suporte à linguagem C.

1. A função **strlen()** determina e devolve a dimensão de uma *string* (como já visto anteriormente, em linguagem C uma *string* é terminada com o carácter '**\0**'). O parâmetro de entrada dessa função é um ponteiro para o início da *string* (i.e., o seu endereço inicial) e o resultado é o número de caracteres dessa *string* (excluindo o terminador).

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

- a) Traduza a função anterior para *assembly*, aplicando as regras de utilização de registos e de passagem e devolução de valores do MIPS (veja o resumo das regras no final do guião).
- b) Para teste da função **strlen()** o programa seguinte imprime o número de caracteres de uma *string* definida de forma estática no programa. Traduza esse programa para *assembly* aplicando as regras de utilização de registos e de passagem e devolução de valores. Não se esqueça que a função **main()** é, em termos de implementação, tratada como qualquer outra função.

```
int main(void)
{
    static char str[]="String de teste";

    print_int10(strlen(str));
    return 0;
}
```

- c) O programa seguinte usa a função **strlen()** para determinar e afixar no ecrã o número de caracteres de cada uma das *strings* passadas como argumentos na linha de comando. Traduza para *assembly* e teste o seu funcionamento.

```
int main(int argc, char *argv[])
{
    int i;
    for(i=0; i < argc; i++)
    {
        print_char('\n');
        print_str(argv[i]);
        print_str(" - ");
        print_int10(strlen(argv[i]));
    }
    return 0;
}
```

2. A função **strrev()** (*string reverse*) inverte o conteúdo de uma *string*. Tal como no caso da função anterior, o parâmetro de entrada dessa função é um ponteiro para o início da *string*, i.e., o seu endereço inicial. A função retorna o ponteiro com o mesmo valor que foi passado como argumento.

```
char *strrev(char *str)
{
    char *p1 = str;
    char *p2 = str;

    while(*p2 != '\0')
        p2++;
    p2--;
    while( p1 < p2 )
    {
        exchange(p1, p2);
        p1++;
        p2--;
    }
    return str;
}

void exchange(char *c1, char *c2)
{
    char aux = *c1;

    *c1 = *c2;
    *c2 = aux;
}
```

- a) Traduza a função **strrev()** para *assembly*.
- b) Escreva, em linguagem C, a função **main()** para chamada e teste desta função (use como base o programa apresentado na alínea b) do exercício 1. Traduza esse programa para *assembly* e teste a função **strrev()**.
3. A função **strcpy()** (*string copy*) copia uma *string* residente numa zona de memória para outra zona de memória. A função aceita como argumentos um ponteiro para a *string* de origem (**src**) e um ponteiro para a zona de memória destino (**dst**). A função devolve ainda o ponteiro "**dst**" com o mesmo valor que foi passado como argumento.

```
char *strcpy(char *dst, char *src)
{
    int i=0;
    do
    {
        dst[i] = src[i];
    } while(src[i++] != '\0');
    return dst;
}
```

- a) Traduza a função **strcpy()** para *assembly*.
- b) Traduza para *assembly* a função **main()**, apresentada de seguida, para chamada e teste da função **strcpy()**, que usa também a função **strlen()** implementada no exercício 1. Teste o conjunto.
- c) Reescreva, em C, a função **strcpy()** usando acesso por ponteiro em vez de acesso indexado. Faça as correspondentes alterações no programa *assembly* e teste o resultado.

```

#define STR_MAX_SIZE 10

int main(int argc, char *argv[])
{
    static char buf[STR_MAX_SIZE + 1];

    if(argc == 1)
    {
        if(strlen(argv[0]) <= STR_MAX_SIZE)
        {
            strcpy(buf, argv[0]);
            print_str(buf);
        }
        else
        {
            print_str("String too long. Nothing done!\n");
            return 1;
        }
    }
    return 0;
}

```

4. A função **strcat()** (*string concatenate*) permite concatenar duas *strings* – a *string* origem é concatenada no fim (isto é, na posição do terminador) da *string* destino. Tal como na função **strcpy()**, os argumentos de entrada são os ponteiros para a *string* origem (**src**) e para a *string* destino (**dst**). A função devolve ainda o ponteiro "**dst**" com o mesmo valor que foi passado como argumento. Compete ao programa chamador reservar espaço em memória com dimensão suficiente para armazenar a *string* resultante.

```

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

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

```

- a) Traduza a função **strcat()** para *assembly*.
b) Traduza para *assembly* a função **main()**, apresentada de seguida, para chamada e teste da função **strcat()**.

```

int main(void)
{
    static char str1[]="Arquitectura de ";
    static char str2[50];

    strcpy(str2, str1);
    print_str( strcat(str2, "Computadores") );
    return 0;
}

```

Anexo:

Regras para a implementação de subrotinas no MIPS:

1. A subrotina chamadora, antes de chamar:
 - Passa os parâmetros; os 4 primeiros são passados nos registos **\$a0..\$a3** e os restantes na *stack*.
 - Executa a instrução **"jal"**.
2. A subrotina chamada, no início:
 - Salva na *stack* os registos **\$s0 a \$s7** que pretende utilizar.
 - Salva o registo **\$ra** no caso de a rotina também ser chamadora.
3. A subrotina chamada, no fim:
 - Coloca o valor de retorno em **\$v0** (excepto se for tipo **void**).
 - Restaura os registos **\$s0 a \$s7** que salvaguardou no início.
 - Restaura o registo **\$ra** (no caso de ter sido salvaguardado no início).
 - Retorna, executando a instrução **"jr \$ra"**.
4. A subrotina chamadora, após regresso:
 - Usa o valor de retorno que está em **\$v0**.
5. A subrotina chamadora não pode assumir em caso algum que qualquer dos registos **\$a0..\$a3**, **\$t0..\$t9**, **\$v0** e **\$v1** são preservados pela rotina chamada.
6. A codificação da subrotina **"main"** está sujeita às mesmas regras que se aplicam às restantes subrotinas.

Considerações práticas sobre a convenção de utilização de registos

1. Subrotinas terminais (que não chamam qualquer subrotina):
 - Só devem utilizar (preferencialmente) registos que não necessitam de ser salvaguardados (**\$t0..\$t9**, **\$v0..\$v1** e **\$a0..\$a3**).
2. Subrotinas intermédias (que chamam outras subrotinas):
 - Devem utilizar os registos **\$s0..\$s7** para o armazenamento de valores que se pretenda preservar. A utilização destes registos implica a sua prévia salvaguarda na *stack* logo no início da subrotina e a respectiva reposição no final.
 - Devem utilizar os registos **\$t0..\$t9**, **\$v0..\$v1** e **\$a0..\$a3** para os restantes valores.