

AULA PRÁTICA N.º 8

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:

1. A função seguinte converte para um inteiro de 32 bits a quantidade representada por uma *string* numérica em que cada caracter representa o código ASCII de um dígito decimal (i.e., 0 - 9). A conversão termina quando é encontrado um caracter não numérico.

```
unsigned int atoi(char *s)
{
    unsigned int digit, res = 0;

    while( (*s >= '0') && (*s <= '9') )
    {
        digit = *s++ - '0';
        res = 10 * res + digit;
    }
    return res;
}
```

- a) Traduza para *assembly* a função `atoi()` (não se esqueça da aplicação das regras de utilização dos registos do MIPS).
- b) O programa seguinte permite fazer o teste da função `atoi()`. Traduza para *assembly* e verifique o correcto funcionamento da função com outras *strings*.

```
int main(void)
{
    static char str[]="2040, o ano do fim das PPP";

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

- c) Altere a função `atoi()` de modo a processar uma *string* binária. Traduza as alterações para *assembly* e teste-as.
2. A função `itoa()`, que se apresenta de seguida, determina a representação do inteiro "n" na base "b" (b pode variar entre 2 e 16), colocando o resultado no *array* de caracteres "s", em ASCII. Esta função utiliza o método das divisões sucessivas para efectuar a conversão entre a base original (hexadecimal) e a base destino "b": por cada nova divisão é encontrado um novo dígito da conversão (o resto da divisão inteira), esse dígito é convertido para ASCII e o resultado é colocado no *array* de caracteres.
Como é conhecido, neste método de conversão o primeiro dígito a ser encontrado é o menos significativo do resultado. Assim, a última tarefa da função `itoa()` é a chamada à função `strrev()` (implementada na aula anterior) para efectuar a inversão da *string* resultado.

```

char toAscii( char );
char *strRev( char *);

char *itoa(unsigned int n, unsigned int b, char *s)
{
    char *p = s;
    char digit;

    do
    {
        digit = n % b;
        n = n / b;
        *p = toascii( digit );
        p++;
    } while( n > 0 );
    *p = '\0';
    strrev( s );
    return s;
}

// Converte o digito "v" para o respectivo código ASCII
char toascii(char v)
{
    v += '0';
    if( v > '9' )
        v += 7; // 'A' - '9' - 1
    return v;
}

```

- Traduza a função `itoa()` para *assembly*¹.
- Escreva em linguagem C a função `main()` para teste da função `itoa()` e traduza-a para *assembly*. Teste a função `itoa()` com diferentes valores e bases.
- A função seguinte apresenta a implementação de uma função para impressão de um inteiro através da utilização da *system call* `print_str()` e da função `itoa()`. Traduza para *assembly* esta função e teste-a, escrevendo a respectiva função `main()`.

```

void print_int_acl(unsigned int val, unsigned int base)
{
    static char buf[33];

    print_str( itoa(val, base, buf) );
}

```

¹ A função `strrev()` foi já implementada no guião anterior. De modo a simplificar a gestão do código desenvolvido, pode usar várias janelas do editor do MARS (a que correspondem outros tantos ficheiros): por exemplo, uma janela para o código a escrever da função `itoa()` e respectivo `main()` e outra janela com a função `strrev()`. Nesse caso, deverá ter em atenção o seguinte:

- No menu *settings* a opção "Assemble all files in directory" tem que ser activada.
- Os nomes das funções que sejam declaradas no(s) ficheiro(s) secundário(s) (o ficheiro principal é o que tem definido o label "`main`") têm que ser declarados como globais. Por exemplo, se o ficheiro que contém a declaração dos labels "`strrev:`" e "`strcpy:`" é um ficheiro secundário, no topo desse ficheiro deve aparecer a seguinte directiva:

```
.globl strrev, strcpy
```

- Apenas um ficheiro pode conter a declaração do label "`main:`".

3. O programa seguinte lê da linha de comando 3 argumentos (sob a forma de *strings*): dois valores inteiros e uma operação (e.g. 27 x 12). Com base nesses 3 argumentos o programa calcula o resultado e imprime-o, usando a função implementada no exercício anterior.

```
int main(int argc, char *argv[])
{
    int val1, val2, res, exit_code;
    char op;

    exit_code = 0;
    if(argc == 3)
    {
        val1 = atoi(argv[0]);
        val2 = atoi(argv[2]);
        op = argv[1][0];
        if(op == 'x')
            res = val1 * val2;
        else if(op == '/')
            res = val1 / val2;
        else if(op == '%')
            res = val1 % val2;
        else
        {
            print_str("\nOperacao desconhecida");
            exit_code = 1;
        }
    }
    else
    {
        print_str("\nNumero de argumentos errado");
        exit_code = 2;
    }
    if(exit_code == 0)
    {
        print_int_acl(val1, 10);
        print_char(op);
        print_int_acl(val2, 10);
        print_char('=');
        print_int_acl(res, 10);
    }
    return exit_code;
}
```

- a) Traduza para *assembly* o programa anterior e teste-o no MARS passando diferentes argumentos na linha de comando.

4. A função seguinte implementa o algoritmo de divisão de inteiros apresentado nas aulas teóricas (versão otimizada), para operandos de 16 bits.

```

unsigned int div(unsigned int dividendo, unsigned int divisor)
{
    int i, bit, quociente, resto;

    divisor = divisor << 16;
    dividendo = (dividendo & 0xFFFF) << 1;

    for(i=0; i < 16; i++)
    {
        bit = 0;
        if(dividendo >= divisor)
        {
            dividendo = dividendo - divisor;
            bit = 1;
        }
        dividendo = (dividendo << 1) | bit;
    }
    resto = (dividendo >> 1) & 0xFFFF0000;
    quociente = dividendo & 0xFFFF;

    return (resto | quociente);
}

```

- a) Traduza esta função para *assembly* e teste-a com diferentes valores de entrada, tendo em atenção que os operandos têm uma dimensão máxima de 16 bits.
- b) Rescreva, em linguagem C, o programa do exercício 3 substituindo as operações de divisão e de obtenção do resto pela chamada à função `div()`. Traduza as alterações para *assembly* e teste de novo o programa com diferentes argumentos de entrada.
- c) O programa anterior apresenta uma deficiência de funcionamento em situações em que o dividendo é igual ou superior a `0x8000` e o divisor é superior ao dividendo. Verifique, com um exemplo, essa situação, identifique a origem do problema e proponha uma solução, em linguagem C, para o resolver.