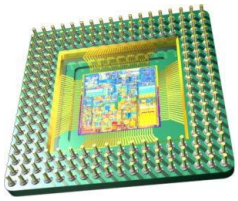

Lab 4 – Variáveis, Operações Lógicas e Chamada a Funções

Os exercícios que se seguem devem ser executados através do MARS, um simulador da arquitetura MIPS disponível em http://courses.missouristate.edu/KenVollmar/MARS/.

1. Variáveis

Num programa em C, as variáveis têm de ser declaradas, tipicamente no início do programa, para posteriormente poderem ser utilizadas nas funções definidas no código. Em *assembly* o programador tem ele próprio de reservar o espaço em memória para as variáveis. Isso é feito através da alocação de memória na secção ***.data***, através de diretivas tais como .space, .byte, .word e .asciiz. Por exemplo, uma tabela com 32 caracteres chamada *buf* pode ser declarada como:

```
          .data
buf:      .space 32 # char buf [32];
```

Um programa em *assembly* pode referenciar esta tabela da seguinte forma:

```
          .text
          ...
          la $t0,buf       # t0=&(buf[0]);
          lbu $t5,0($t0)   # t5=*t0;
          ...
```

Outros tipos de dados podem ser declarados como variáveis, como por exemplo:

```
          .data
str:      .asciiz "Hello World\n" # Null terminated string
num1:     .word 42               # As variáveis inteiras podem ser
                                 # declaradas como words de 32 bits
tab1:     .word 1, 2, 3, 4, 5    # Tabelas de Inteiros
tab2:     .byte 'a', 'b'         # Tabelas de Caracteres (1 byte)
buffer:   .space 60              # Espaço reservado para 60 bytes
```

2. Chamada a Funções

O MIPS tem algumas instruções que permitem implementar de forma simples o mecanismo de chamada a funções. Essas instruções são as seguintes:

Instrução JAL (*Jump And Link*) - é uma instrução de salto utilizada unicamente em funções. Essa instrução altera o fluxo de execução do programa, saltando para um novo endereço na memória (*jump*) guardando no registo ***$ra*** o endereço da próxima instrução a ser executada após o regresso da função (*link*). A sintaxe da instrução é a seguinte:

```
jal label_funcao
```

Por exemplo:

```
jal Soma
```

onde *Soma* é o *label* da primeira instrução da função. Recorde que um *label* é uma forma elegante de representar um endereço de memória.

Instrução JR (*Jump to Register*): é uma instrução de salto incondicional para o endereço contido no registo passado como parâmetro. Esta instrução é útil para permitir o retorno de uma função através de um salto para o endereço armazenado no registo ***$ra***. A sintaxe da instrução é a seguinte:

```
jr $ra
```

Na utilização de funções são utilizados alguns registos que desempenham um papel específico:

`$a0` até `$a3`: são registos onde são passados os parâmetros da função;

`$v0` e `$v1`: são os registos utilizados para a devolução do resultado de uma função, caso a função retorne alguma coisa;

`$ra`: é o registo onde é armazenado o endereço do retorno da função. Permite voltar ao ponto de origem da chamada da função.

Mais tarde, noutra ficha prática revisitaremos este assunto, discutindo com maior detalhe as questões envolvidas nas chamadas a funções no *assembly* do MIPS.

3. Exercícios

Com os exercícios a seguir pretende-se que o aluno pratique a programação em *assembly* utilizando variáveis, funções e operações lógicas:

1. Implemente uma função que permita trocar a ordem dos *bytes* de uma *word*. Por exemplo, se o valor original da *word* for igual a `0x10203040`, então o resultado da função deverá ser `0x40302010`. Para testar a função utilize uma variável do tipo *.word* para inicializar o valor a trocar. Escreva no ecrã o resultado utilizando a função 34 do *syscall* (*print integer in hexadecimal*).

2. Implemente uma função que permita calcular o valor absoluto da diferença de dois números inteiros passados como parâmetros: |A - B|. Para testar a função, defina no segmento de dados três variáveis inteiras, duas para os parâmetros e uma terceira para armazenar o resultado. Utilizando o syscall apropriado, escreva também no ecrã o resultado.

3. Implemente uma função que permita calcular a paridade de um número inteiro de 32 bits. Considere que a paridade é 1 se o número de bits a 1 for ímpar e 0 caso contrário. Mais uma vez utilize variáveis para inicializar o valor a testar e para armazenar o resultado da função.

4. Implemente uma função que receba o endereço de uma *string* e retorne o seu comprimento. Assuma que a *string* termina com o código zero. Para testar esta função defina uma variável do tipo *.asciiz* no segmento de dados. Imprima no ecrã o resultado devolvido pela função, utilizando uma `syscall` e guarde numa variável o comprimento da *string*.

```
            .data
str:        .asciiz "Uma frase!"
comp:       .word     0
```

5. Pretende-se implementar em *Assembly* uma função denominada troca(str,c1,c2), que dada uma *string* e dois caracteres, c1 e c2, substitui nessa *string* todas as ocorrências do caracter c1 pelo caracter c2. A função terá como argumentos o endereço da *string*, o caracter a procurar e o caracter para a substituição. Assuma que a *string* termina com o código zero. Por exemplo, para a *string* "socorro", se o caracter a procurar for o caracter 'o', e o caracter a substituir for o caracter 'u', a *string* deve ser alterada para "sucurru". A função deverá retornar o número de caracteres que efectivamente foram alterados. Armazene esse valor numa variável.