SSC0511 - Organização de Computadores Digitais 1

Prof. Fernando Santos Osório
Monitor: André Bannwart Perina

Lista 2 - Processador MIPS32

Links úteis (`http://spimsimulator.sourceforge.net/HP_AppA.pdf`):

- **Lista de registradores:** Página 23
- **Lista de syscalls:** Página 43
- **Lista de diretivas:** Página 46
- **Instruction set:** Página 50 em diante

1. Faça um programa para leitura de strings da entrada padrão (até 32 bytes) e a imprima na saída padrão.
2. Faça um programa não-recursivo para cálculo de fatorial. Lembre-se que $0! = 1! = 1$.
3. Faça uma função para calcular a^b (onde $a \in \mathbb{Z}$ e $b \in \mathbb{N} \cup \{0\}$). Tal função deve receber a base e o expoente nos registradores de argumentos `$a0` e `$a1`, respectivamente, e retornar o resultado no registrador `$v0`. Faça um caso de teste.
4. Faça uma função para calcular fatorial recursivamente. Neste caso, será necessário utilizar a pilha. Use o registrador `$a0` para passar a entrada e retorne o resultado pelo registrador `$v0`. O tratamento da pilha é feito totalmente pelo programador. Veja a seção de dicas abaixo para auxiliar em como utilizar a pilha corretamente.
5. **DESAFIO (+0.5):** Refaça a função para cálculo de potência, desta vez considerando que a é um ponto flutuante. Veja a seção de dicas abaixo para mais informações.

Dicas:

1. Para realizar um load imediato de um ponto flutuante, utilize a seguinte técnica:

```
    li $s0, 0x3f800000 # Carrega 1.0 (0x3f800000 em hex) imediato no registrador $s0
        do processador
    mtc1 $s0, $f0 # Move 1.0 para o registrador $f0 do coprocessador de ponto
        flutuante
```

2. Use `http://www.binaryconvert.com/convert_float.html` para converter valores reais em representação hexadecimal.

3. A montagem da pilha é de responsabilidade do programador. Por convenção, quem faz push e pop na pilha é a função que foi chamada, e não quem chamou. Para uso correto da pilha, deve-se entender primeiro como se monta o quadro (frame) a ser armazenado:

   - Os primeiros 16 bytes são reservados para cópia dos registradores de argumentos `$a0` a `$a3`. Em geral, este espaço reservado não é utilizado;
   - Guardar os argumentos adicionais, caso haja;
   - Guardar variáveis que devem se manter após uma chamada de função (`$s0` a `$s7`, caso um destes seja utilizado);
   - Registrador do endereço de retorno da chamada de função (`$ra`). Deve obrigatoriamente ser guardado caso uma função chame outra função dentro de si;
   - Espaço em branco de 4 bytes, necessário SOMENTE para deixar o frame com tamanho em bytes múltiplo de 8. Caso o frame já seja múltiplo sem este espaço, desconsiderá-lo (Tal restrição permite que doubles sejam guardados na pilha);
   - Variáveis temporárias que o usuário queira manter após uma chamada à função.

   Abaixo segue um exemplo onde o usuário vai utilizar todos os registradores (`$s0` a `$s7`), (`$t0` a `$t9`) e fará chamadas à função dentro desta função `example`:

```
example:addiu $sp, $sp, -96 # Allocate (push) space for the stack frame (96 bytes)
    sw $s0, 16($sp) # Save $s0
    sw $s1, 20($sp) # Save $s1
    sw $s2, 24($sp) # Save $s2
    sw $s3, 28($sp) # Save $s3
    sw $s4, 32($sp) # Save $s4
    sw $s5, 36($sp) # Save $s5
    sw $s6, 40($sp) # Save $s6
    sw $s7, 44($sp) # Save $s7
    sw $ra, 48($sp) # Save return address register
    # Position 52($sp) jumped to align double words
    sw $t0, 56($sp) # Save $t0
    sw $t1, 60($sp) # Save $t1
    sw $t2, 64($sp) # Save $t2
    sw $t3, 68($sp) # Save $t3
    sw $t4, 72($sp) # Save $t4
    sw $t5, 76($sp) # Save $t5
    sw $t6, 80($sp) # Save $t6
    sw $t7, 84($sp) # Save $t7
    sw $t8, 88($sp) # Save $t8
    sw $t9, 92($sp) # Save $t9

    # Do stuff, including call to other subroutines
    # Example:
    jal example2
    # Do more stuff

    lw $t9, 92($sp) # Restore $t9
```

```
    lw $t8, 88($sp) # Restore $t8
    lw $t7, 84($sp) # Restore $t7
    lw $t6, 80($sp) # Restore $t6
    lw $t5, 76($sp) # Restore $t5
    lw $t4, 72($sp) # Restore $t4
    lw $t3, 68($sp) # Restore $t3
    lw $t2, 64($sp) # Restore $t2
    lw $t1, 60($sp) # Restore $t1
    lw $t0, 56($sp) # Restore $t0
    lw $ra, 48($sp) # Restore return address register
    lw $s7, 44($sp) # Restore $s7
    lw $s6, 40($sp) # Restore $s6
    lw $s5, 36($sp) # Restore $s5
    lw $s4, 32($sp) # Restore $s4
    lw $s3, 28($sp) # Restore $s3
    lw $s2, 24($sp) # Restore $s2
    lw $s1, 20($sp) # Restore $s1
    lw $s0, 16($sp) # Restore $s0
    addiu $sp, $sp, 96 # Deallocate (pop) space for the stack frame

    jr $ra # Return to above function
```

Em um exemplo mais básico (como do exercício 4), caso o usuário utilize apenas `$s0`:

```
example:addiu $sp, $sp, -24 # Allocate (push) space for the stack frame (24 bytes)
    sw $s0, 16($sp) # Save $s0
    sw $ra, 20($sp) # Save return address register

    # Do stuff, including call to functions

    lw $ra, 20($sp) # Restore return address register
    lw $s0, 16($sp) # Restore $s0
    addiu $sp, $sp, 24 # Deallocate (pop) space for the stack frame
    jr $ra # Return
```

Não se esqueça que o registrador `$ra` deve sempre ser salvo na pilha caso uma função chame outra função!