«««< HEAD

Aula 7 - Linguagem de montagem - 04/07/22

Procedimentos

> Função é um tipo de procedimento que possui retorno. Procedimentos não possuem retorno.

Para compilar procedimentos em Assembly é necessário entender alguns conceitos:

- caller: procedimento que faz a chamada
- callee: procedimento que é chamado

  Em procedimentos recursivos, o próprio procedimento será caller e callee, diferenciando-se as instâncias em execução.

- Program Counter (PC)
  - O programa é um vetor de instruções na memória, ou seja, cada instrução possui um endereço de memória. O PC é um registrador que armazena o endereço de memória da próxima instrução a ser executada pelo programa.

Obs: As instruções de desvio (beq, bne e j) operam sobre o PC.

- Procedimento folha: é um procedimento que não faz chamada a nenhum outro procedimento. O nome folha faz referência à estrutura de árvore, estes procedimentos estão nas extremidades, ou seja, não fazem chamadas à nenhum outro procedimento.

Exemplo:

```
main() { // Nesse instante o caller é o S.O. e o callee é a main
    // ...
    media(x, y); // Nesse instante o caller é a main e o callee é a função média.
}

media(int x, int y){ # Procedimento folha, pois não faz chamada à nenhum outro procedimento.
    return (x+y)/2;
}
```

- No Assembly MIPS as variáveis (registradores) possuem "escopo" global, então cabe ao programador salvar os valores utilizados nos registradores salvos (ex: $s0) ao iniciar um procedimento, para não sobrescrever informações utilizadas em outros procedimentos.

Etapas para a chamada de um procedimento

1. Armazenar os parâmetros nos registradores ($a0-a3$), se couber, e caso não caiba a passagem de argumentos deve ser feita pela memória (utilizando

as instruções sw e lw).

2. Desviar o fluxo do programa para o procedimento (jal label).
3. Ajustar o armazenamento no procedimento:
   - Há argumentos na memória que precisam ser carregados?
   - É necessário fazer backup de registradores salvos $s?
4. Executar as instruções do procedimento.
5. Salve o retorno do procedimento ($v0-$v1).
6. Restaure os backups.
7. Retorne ao caller (jr $ra).

Para fazer o desvio do programa para um procedimento utilizamos a instrução jump and link (jal).

- `jal label`: desvia para um procedimento rotulado por label, e salva o conteúdo do PC no registrador $ra.

Para retornar ao caller utilizamos a instrução jump register (jr).

- `jr $ra`: desvia para a instrução cujo endereço está em $ra.

Como fazer e restaurar os backups?

Por convenção, é necessário salvar o conteúdos dos registradores $s. Para isso, utilizamos a pilha.

A memória do computador é segmentada em 4 partes, do endereço 0 ao endereço máximo, temos:

- Os endereços mais baixos são reservados para o sistema operacional
- Acima dele vem o segmento de texto, que é a parte da memória reservada para armazenar as instruções de programa. O PC sempre contém um endereço desse segmento da memória.
- Dados estáticos: dado cujo o endereço de memória permanece alocado para o programa mesmo que o escopo da variável já tenha acabado.
- Dados dinâmicos:
  - heap: parte da memória reservada para as alocações dinâmicas, cresce de baixo para cima.
  - pilha: cresce de cima para baixo.

O registrador global stack pointer ($sp) aponta para a última posição de memória com dados na pilha, ou seja, o topo da pilha. Para armazenar dados na pilha é necessário seguir o seguinte passo a passo:

1. Abrir espaço na pilha: decrementar a quantidade necessária de bytes de $sp.
2. Armazenar os dados utilizando a expressão sw a partir de $sp.

Para restaurar os dados da pilha:

1. Restaurar os dados usando lw.
2. Restaurar o espaço da pilha: incrementar a quantidade de bytes que foram utilizadas em $sp novamente.

Exemplo: armazenar e restaurar o conteúdo dos registradores $s0 e $s1 na pilha.

```
subi $sp, $sp, 8 # Abrindo 2 espaços de memória
sw $s0, 0($sp) # Armazena $s0 no primeiro espaço, apontado por $sp
sw $s1, 4($sp) # Armazena $s1 no segundo espaço, apontado por $sp + 4
# ...

lw $s1, 4($sp)
lw $s0, 0($sp)
addi $sp, $sp, 8
```

Exemplo 2: fatorial

```
int fat(int n){
    if (n<1)
        return 1;
    else
        return n*fat(n-1);

}
```

```
    fat: # Argumento em $a0

    addi $sp, $sp, -8 # Abre espaço para salvar $ra e $a0
    sw $a0, 0($sp) # Salva a0 na pilha
    sw $ra, 4($sp) # Salva $ra na pilha

    addi $t1, $zero, 1
    slt $t0, $a0, $t1 # t0 = n < 1
    beq $t0, $zero, L1 # if (n<1) goto L1

    addi $v0, $zero, 1
    jr $ra


L1: addi $a0, $a0, -1 #n--
```

```
jal fat

lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8

mul $v0, $a0, $v0
jr $ra
```