Microprocessadores

Hugo Marcondes
hugo.marcondes@ifsc.edu.br

Aula 09

Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

Argumentos e valores de retorno

- Implemente o seguinte pseudo código

```
# Definição do procedimento
# Pseudo código:
#  int sumOfSquare(int a, int b) {
#     return a*a + b*b;
#  }


# Uso do procedimento
# Pseudo código:
# int main() {
#  int c;
#  c = sumOfSquares(3,5)
# }
```


Exemplo

```
# Definição do procedimento
sumOfSquares:
   mul  $t0, $a0, $a0  # tmp1 = a*a
   mul  $t1, $a1, $a1  # tmp2 = b*b
   add  $v0, $t0, $t1  # res = tmp1 + tmp2
   jr   $ra            # return res

# Uso do procedimento
   li   $a0, 3         # (prepara parâmetro)
   li   $a1, 5         #
   jal  sumOfSquares   # (chama procedimento)
   move $t2, $v0       # (pega resultado)
```

$t2 = Res

$a0 à $a3 -> parametros

$v0 e $v1 -> parametro e retorno

Convenção de Chamadas

What's wrong with this code?

```
# Pseudocode:
#   c = sumOfSquares(x,y)
#   c = c - x
# Registers: x => $t0, y => $t1, c => $t2
   move $a0, $t0       # (set up arguments)
   move $a1, $t1
   jal  sumOfSquares   # (call procedure)
   move $t2, $v0       # (get result)
   sub  $t2, $t2, $t0  # c = c - x

# Pseudocode:
#   int sumOfSquares(int a, int b) {
#     return a*a + b*b
#   }
# Registers: a => $a0, b => $a1, res => $v0
sumOfSquares:
   mult $t0, $a0, $a0  # tmp1 = a*a
   mult $t1, $a1, $a1  # tmp2 = b*b
   add  $v0, $t0, $t1  # res = tmp1 + tmp2
   jr   $ra            # return res
```

$ra

sumOfSquares modifica $t0 !

Quem deve preserva-lo ?

Convenção de Chamadas

INSTITUTO FEDERAL SANTA CATARINA

What's wrong with this code?

```
# Pseudocode:
# void question() {
#   print(quest)
#   waitForGiveUp()
#   return
# }
question:
    li $v0, 4          # print(quest)
    la $a0, quest
    syscall

    jal waitForGiveUp  # waitForGiveUp()

    jr $ra             # return

# Pseudocode:
# void waitForGiveUp() { ... }
waitForGiveUp:
    ...
    jr $ra             # return
```

jal modifica $ra !

Quem deve preserva-lo ?

Convenção de Chamadas

INSTITUTO FEDERAL SANTA CATARINA

- Como passar parâmetros para o procedimento ?
  - Solução parcial:
    - armazenar argumento em $a0 – $a3
    - armazenar resultado $v0 e $v1
  - E se houver mais parâmetros ?
- Registradores são variáveis "globais"
  - Os valores que precisamos ainda estão lá após a execução do procedimento ?
  - O $ra permace correto após chamar outro procedimento?
- Os dados no segmento de .data também estão em uma memória "global"
  - E se o procedimento precisar de sua própria área de memória?
    - ex. variáveis locais!
  - Não podemos simplesmente declarar dados globais (recursão)

Convenção de Chamadas

INSTITUTO FEDERAL SANTA CATARINA

- Convenções que o programador segue para
  - garantir o bom funcionamento do programa
  - permitir que ele coopere com código gerado por terceiros
- As convenções de chamada devem responder:
  - Como é realizada a passagem de parâmetros ?
  - Quais são as responsabilidades do caller?
  - Quais são as responsabilidades do callee?
  - Aonde podemos armazenar os dados locais?
- Isso não é implementado pela arquitetura !
- Múltiplas convenções existem !

Convenção de Chamadas

INSTITUTO FEDERAL SANTA CATARINA

Number	Name	Usage	Preserved?
$0	$zero	constant 0x00000000	N/A
$1	$at	assembler temporary	N/A
$2–$3	$v0–$v1	function return values	✗
$4–$7	$a0–$a3	function arguments	✗
$8–$15	$t0–$t7	temporaries	✗
$16–$23	$s0–$s7	saved temporaries	✓
$24–$25	$t8–$t9	more temporaries	✗
$26–$27	$k0–$k1	reserved for OS kernel	N/A
$28	$gp	global pointer	✓
$29	$sp	stack pointer	✓
$30	$fp	frame pointer	✓
$31	$ra	return address	✓

✗= **caller** is responsible ✓= **callee** is responsible

Importância da Pilha

- Onde podemos salvar os registradores quando precisamos, dentro de um procedimento ?
  - No segmento de dados ?
  - Em outro registrador ?
- E se chamarmos um novo procedimento ? E outro ? ...

Esses locais não são extensíveis !

Lembrando que quando voltar a função dar Move do $s0 para outro reg pois $s0 é usando para chamar procedimentos e ele acaba mudando o valor de $v0.

Visão geral da Pilha

- Pilha
  - Uma região da memória
  - Composta por quadros da pilha (Stack Frame)
  - Cada quadro armazena dados específicos da execução de uma chamada de procedimento
  - Cada chamada a um procedimento gera um novo stack frame
    - A pilha é extensível !

Visão geral da Pilha

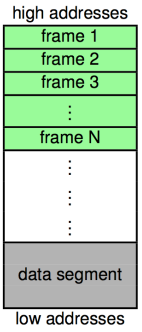
- O que podemos armazenar no quadro da pilha ?
  - Parâmetros adicionais do procedimento
  - Valores dos registradores salvos
  - Endereço de retorno do procedimento ($ra)
  - Variáveis locais
- As convenções de chamada a procedimentos impõe:
  - Como gerenciar a pilha
  - Como organizar e estruturar seus elementos

Como a pilha funciona

- LIFO - Last In, First Out
  - No início de um procedimento, um novo quadro da pilha é colocado (push)
  - No final de um procedimento, o quadro de pilha é retirado (pop)
- O quadro do procedimento atualmente em execução está sempre no TOPO da pilha.

A Pilha na memória

- A pilha é apenas uma região de memória
  - Segmento de Texto: Instruções do programa
  - Segmento de Dados: Constantes e variáveis globais
  - Pilha: Suporte a chamada de procedimentos
    - variáveis locais, passagem de parâmetros e backup de registradores
- Layout da Memória
  - Dados e Pilha: mesmo espaço de endereçamento
  - Pilha inicia nos endereços mais altos
  - Dados inicia nos endereços mais baixo
  - A Pilha cresce para baixo
    - O topo da pilha está no endereço de memória mais baixo



Como utilizar a Pilha em assembly

- Stack Pointer - registrador $sp
  - Aponta para o topo da pilha
  - SO inicializa $sp quando o programa é carregado
  - Após isso, o $sp é manipulado apenas pelo próprio programa

Push stack frame

```
myProcedure:              # start of procedure
    addiu $sp, $sp, -24   # allocate 6 words on the stack

    ...                   # procedure body
```

Pop stack frame

```
    addiu $sp, $sp, 24    # deallocate 6 words on the stack
    jr    $ra             # return
```


Como acessar os dados na pilha

- Leitura e escrita na pilha
  - Da mesma forma como é feito no segmento de dados
    - Usar sw para escrever, lw para ler

Example stack usage

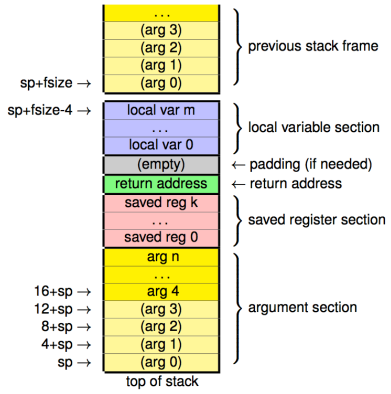
```
# Registers: myVar => $t0
myProcedure:              # start of procedure
    addiu $sp, $sp, -24   # push a new stack frame (6 words)
    sw    $ra, 20($sp)    # save return address
    ...
    sw    $t0, 16($sp)    # save myVar
    jal   subProcedure    # call sub-procedure
    lw    $t0, 16($sp)    # restore myVar
    ...
    lw    $ra, 20($sp)    # restore return address
    addiu $sp, $sp, 24    # pop stack frame
    jr    $ra             # return
```


Quadros de Pilha

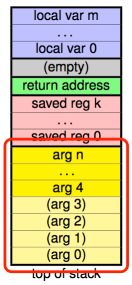
- No exemplo anterior:
  - $t0 é armazenado em 16($sp)
  - $ra é armazenado em 20($sp)
- Como determinamos estes offsets ?
- Por que não utilizar offsets 0, 4, 8 ou 12 ??

A convenção de chamada de procedimentos define qual é a estrutura do quadro de pilha

Estrutura dos quadros de pilha

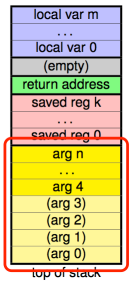


Seção de Parâmetros



- Utilizada para a passagem de parâmetros para sub-rotinas
  - procedimentos chamados por ESTE procedimento
- Primeiras 4 palavras (arg0 - arg3)
  - 0($sp), 4($sp), 8($sp), 12($sp)
  - Nunca utilizado por este procedimento
  - Lugar para as sub-rotinas armazenarem $a0 - $a3

Seção de Parâmetros



- Palavras restantes (arg4 - argN)
  - Utilizado para a passagem de mais parâmetros para sub-rotinas
  - escritos por este procedimento
  - lidos pela sub-rotina
- E os parâmetros deste procedimento ?
  - No topo do quadro de pilha ANTERIOR
  - Lidos antes de colocar esse quadro na pilha

Utilizando a seção de parâmetros

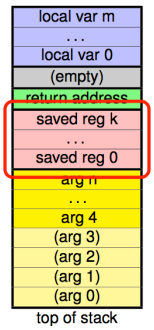
```
# Pseudocode: myProcedure(a,b,c,d,e)
# Registers: a,b,c,d => $a0-$a3, e => $s0
myProcedure:
    lw    $s0, 16($sp)  # retrieve e from prev stack frame
    addiu $sp, $sp, -32 # push new stack frame
    ...
    ...                 # put first four args in $a0--$a3
    sw    $s1, 16($sp)  # store j for subroutine
    sw    $s2, 20($sp)  # store k for subroutine
    jal   subRoutine    # call subRoutine
    ...
```

```
# Pseudocode: subRoutine(f,g,h,i,j,k) { ... }
# Registers: f,g,h,i => $a0-$a3, j => $t0, k => $t1
subRoutine:
    lw    $t0, 16($sp)  # retrieve j from prev stack frame
    lw    $t1, 20($sp)  # retrieve k from prev stack frame
    ...
```

Tamanho da seção de parâmetros

- Quanto espaço é necessário para essa seção ?
  - Verificar todas as sub-rotinas que serão chamadas
  - Seja n o maior número de parâmetros para qualquer uma destas sub-rotinas
  - Será necessário N palavras.
    - Lembre-se - convenção!

Seção para salvamento de registradores



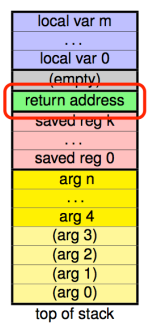
- Todos os registradores salvos que são utilizados pelo procedimento devem ser salvos
  - Para que possamos restaurá-los no final do procedimento
- Como usar
  - No início do procedimento, salve cada registrados $s utilizado
  - No final do procedimento, restaure os valores

Sempre devemos salvá-los !

Utilizando a seção de salvamento de registradores

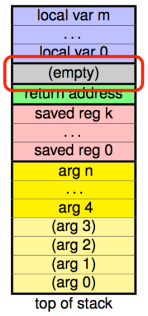
```
# Registers: a => $s0, b => $s1
myProcedure:
    ...                 # maybe retrieve args
    addiu $sp, $sp, -32 # push new stack frame
    sw    $s0, 16($sp)  # save $s0
    sw    $s1, 20($sp)  # save $s1
    ...
    (body of procedure) # (uses $s0 and $s1)
    ...
    lw    $s0, 16($sp)  # restore $s0
    lw    $s1, 20($sp)  # restore $s1
    addiu $sp, $sp, 32  # pop stack frame
    jr    $ra           # return
```


Endereço de retorno



- Precisamos salvar o $ra, para que possamos restaurá-lo depois
  - Necessário se iremos chamar um outro procedimento
- Como usar
  - No início do procedimento, salve o $ra
  - No final do procedimento, restaure o $ra

Padding

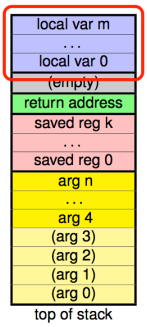


- $sp sempre deve ser múltiplo de 8!
  - Caso parâmetros de palavra dupla!
- Se o tamanho do quadro é múltiplo de 4, uma palavra de padding é adicionada.

sempre que tiver palavras impares eu vou ter não multiplos de 8 e o OLDstack tem que ser multiplo de 8

Então usamos o empty para cobrir esse espaço.

Seção de variáveis locais



- Local para salvar:
  - Valores de registradores temporários ($t0 - $t9)
  - Variáveis locais na memória
- Como usar:
  - Salvar temporários antes de chamar outro procedimento
  - Restaurar temporários após retornar dos procedimentos
  - Variáveis locais - como qualquer outro dado na memória
    - Lembrar: Não são inicializados!

Amarelo -> Registradores de chamada de procedimento $a0/$a1 --> offset 0

Vermelho -> Registradores salvos $s0/$s1 --> offset 4 bytes

Return adress -> $ra -> offset +4bytes

Pilha tem o tamanho que eu quiser

Caso eu nao use algum dos parametros ele só tem offset igual a 0 isso significa que nao é usado

Ex: $a0 <-- 0($sp)
$ra 4offset

old_stack 8offset (Quando de pilha que chamou (main)).

quando eu chamar fatorial 5 os parametros estão destinados e ficam no OLD_Stack e nao na STACK atual que eu chamei.

Quanto espaço preciso para o quadros e pilha ?

- Três tipos de procedimentos:
  - Final simples
    - Sem chamada a sub-rotinas
    - Sem quadro de pilha !
  - Final com dados
    - Sem chamada de sub-rotinas, dados locais
    - Quanto for necessário!
  - Não final
    - Executa chamadas a subrotinas
    - A maioria dos dados do quadro de pilha

Minimum size: 6 words (24 bytes)
- arg 0 – arg 3 (4)
- return address (1)
- padding (1)

OLD_STACK SEMPRE MULTIPLO DE 8

Calculando o tamanho do quadro de pilha

- Para determinar o número de palavras:
  1. Tamanho da seção de parâmetros
     - Verifique todas as sub-rotinas
     - Seja N o maior número de parâmetros
     - N palavras
  2. + tamanho dos registradora salvos
     - Número de registradores $s que o procedimento usa
  3. + 1 para o endereço de retorno
  4. + 1 para padding, se necessário
  5. + tamanho da seção de dados locais
     - Número de registradores $t
     - + espaço alocado para variáveis locais
- Basta multiplicar por 4 para ter o tamanho em bytes !

$ra (Retorn adress) --> sempre salvar ele para nao entrar no loop infinito

$v0 --> Retorno do parametro para a função

QUando chamo função, ele faz 2 coisas:

Armazena o endereço daquele local para voltar depois de fazer a função, no reg $ra

e pulo para a função.

Chamando os procedimentos

- Ao chamar procedimentos, precisamos sastifazer:
  - As convenções de chamada de procedimentos
  - Gerenciar a pilha

- Caller
  1. Preparação da chamada
  2. Chama o procedimento
  3. Limpeza da chamada

- Callee
  5. Prólogo do procedimento
  6. Execução
  7. Epílogo do procedimento

Responsabilidades do Caller

- Preparação da chamada
  1. Salvar registradores $t necessário após a chamada
  2. Configurar os argumentos que devem ser passados para o procedimento

- Chama procedimento

- Limpeza da chamada
  1. Recupera o resultado do procedimento ($v0 - $v1)
  2. Restaura o valor dos registradores $t

Responsabilidades do Callee

- Prólogo do procedimento
  1. Busca parâmetros da pilha (quadro anterior)
  2. Cria seu próprio quadro de pilha
  3. Salva todos os registradores $s utilizados
  4. Armazena o $ra

- Executa o procedimento

- Epílogo do procedimento
  1. Restaura todos os registradores $s salvos no prólogo
  2. Restaura o valor de $ra
  3. Remove o quadro da pilha

Responsabilidades na chamada de procedimento

```
myProcedure:
  # (procedure prologue, as callee)
  ...
  # (caller startup)
  jal subRoutine1
  # (caller cleanup)
  ...
  # (caller startup)
  jal subRoutine2
  # (caller cleanup)
  ...
  # (procedure epilogue, as callee)
  jr  $ra
```

Funções - Ponto Flutuante

- Os registradores do coprocessador de ponto flutuante devem ser tratados da mesma forma que os registradores salvos do banco geral
  - Devem ser salvo os pares (double word)
  - Alinhado em 8 bytes
- Passagem de parâmetros pelos registradores $f12 e $f14
  - Se double - $f12/13, $f14/15
- Retorno de valor - $f0 (ou $f0/1 se double)