

**From: Ronnie Gran <ragran@uci.edu>  
Date: Wed, Mar 30, 2011 at 9:46 AM  
Subject: UC Irvine Graduate Application  
To: leonardo.ecco@gmail.com**

**Dear Leonardo Luiz Ecco,**

**Congratulations! We are pleased to inform you that the members of the Electrical Engineering and Computer Science admissions committee have recommended that you be admitted for graduate degree study at the University of California, Irvine beginning in the Fall 2011 academic term.**

**This is a very competitive application process, with over 1400 applicants this year, and less than 15% will be admitted. You should be proud of your academic accomplishments so far! We are proud of you, and we hope you choose to join us to continue your graduate studies.**

**...**

**Once again, warm congratulations on a promising career! We hope to see you in the fall!**

**Sincerely,  
Admissions Committee  
Electrical Engineering and Computer Science  
University of California, Irvine**

**Luiz C. V. dos Santos, INE/CTC/UFSC**

**INE 5411, A-II, slide 1**

From: Ronnie Gran <ragran@uci.edu>  
Date: Wed, Mar 30, 2011 at 9:46 AM  
Subject: UC Irvine Graduate Application  
To: leonardo.ecco@gmail.com

**Aceito também na École Polytechnique  
Fédérale de Lausanne, Suíça**

**E também na Technische Universität  
Braunschweig, Alemanha**

Dear Leonardo Luiz Ecco,  
**(Aluno desta disciplina em 2005.2 P1=8,0; P2=7,0; P3=8,0; NF = 7,5)**

Congratulations! We are pleased to inform you that the members of the Electrical Engineering and **Computer Science admissions committee have recommended that you be admitted for graduate degree study at the University of California, Irvine** beginning in the Fall 2011 academic term.

This is a very competitive application process, with over 1400 applicants this year, and less than 15% will be admitted. You should be proud of your academic accomplishments so far! We are proud of you, and we hope you choose to join us to continue your graduate studies.

...

Once again, warm congratulations on a promising career! We hope to see you in the fall!

Sincerely,  
Admissions Committee  
Electrical Engineering and Computer Science  
University of California, Irvine

Luiz C. V. dos Santos, INE/CTC/UFSC

INE 5411, A-II, slide 2

# Montador e uso da memória

# Montador: fundamentos

- Processa um arquivo isoladamente por vez
- Desvios e load/store:
  - Podem referenciar endereços simbólicos (labels)
- Labels:
  - Possível uso antes da declaração
- Consequência:
  - Nem todas as instruções podem ser montadas numa única passada

# **Montador: referências a endereços simbólicos**

- **Label local**
  - Só referenciado no arquivo onde é definido
- **Label global**
  - Pode ser referenciado fora do arquivo onde definido
- **Label externo**
  - Referência a endereço simbólico definido (globalmente) em outro arquivo

# Montador: exemplos de referências

```
.text
.align 2
.globl main
main:
    subu    $sp, $sp, 32
    sw      $ra, 20($sp)
    sd      $a0, 32($sp)
    sw      $0, 24($sp)
    sw      $0, 28($sp)
loop:
    lw      $t6, 28($sp)
    mul     $t7, $t6, $t6
    lw      $t8, 24($sp)
    addu    $t9, $t8, $t7
    sw      $t9, 24($sp)
    addu    $t0, $t6, 1
    sw      $t0, 28($sp)
    ble     $t0, 100, loop
    la      $a0, str
    lw      $a1, 24($sp)
    jal     printf
    move    $v0, $0
    lw      $ra, 20($sp)
    addu    $sp, $sp, 32
    jr      $ra

.data
.align 0
str:
.asciiz "The sum from 0 .. 100 is %d\n"
```

# **Montador: princípio de funcionamento**

- **Passo 1: resolução de referências**
  - Extração de componentes
  - Mapeamento de referências para endereços
- **Passo 2: tradução das instruções**
  - Codificação binária dos elementos nos campos do formato da instrução

# Montador:

## passo 1 - resolução de referências

- Lê cada linha do arquivo e extrai seus componentes
- Se componente contém referência
  - É armazenada em uma **tabela de símbolos**
- Ao final
  - Todas as referências locais resolvidas
  - Referências externas não resolvidas



# Montador: exemplo de montagem

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
      add $t1, $t1, $s6
      lw $t0, 0($t1)
      bne $t0, $s5, Exit
      addi $s3, $s3, 1
      j Loop
Exit:  sub $s3, $s4, $s5
      jal printf
```

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

```
PLC = ?????? → . Text 0x400000
                  add $s5, $s1, $s2
Loop:             sll $t1, $s3, 2
                  add $t1, $t1, $s6
                  lw $t0, 0($t1)
                  bne $t0, $s5, Exit
                  addi $s3, $s3, 1
                  j Loop
Exit:             sub $s3, $s4, $s5
                  jal printf
```

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

**PLC = 400000** → **. Text 0x400000**  
**Loop:** **add \$s5, \$s1, \$s2**  
**sll \$t1, \$s3, 2**  
**add \$t1, \$t1, \$s6**  
**lw \$t0, 0(\$t1)**  
**bne \$t0, \$s5, Exit**  
**addi \$s3, \$s3, 1**  
**j Loop**  
**Exit:** **sub \$s3, \$s4, \$s5**  
**jal printf**

**Tabela de símbolos**

**Código assembly**

# Montador: exemplo de montagem

**PLC = 400004** → **Loop:**

```
. Text 0x400000
    add $s5, $s1, $s2
    sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
     jal printf
```

**Tabela de símbolos**

**Código assembly**

# Montador: exemplo de montagem

**PLC = 400004** → **Loop:**

```
. Text 0x400000
    add $s5, $s1, $s2
    sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
     jal printf
```

Loop	0x400004
------	----------

**Tabela de símbolos**

**Código assembly**

# Montador: exemplo de montagem

PLC = 400008 →

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
      add $t1, $t1, $s6
      lw $t0, 0($t1)
      bne $t0, $s5, Exit
      addi $s3, $s3, 1
      j Loop
Exit: sub $s3, $s4, $s5
      jal printf
```

Loop	0x400004
------	----------

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

PLC = 40000c →

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
    jal printf
```

Loop	0x400004
------	----------

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

**PLC = 400010 →**

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
    jal printf
```

Loop	0x400004
------	----------

**Tabela de símbolos**

**Código assembly**



# Montador: exemplo de montagem

PLC = 400010 →

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
    jal printf
```

Loop	0x400004
Exit	

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

PLC = 400014 →

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
    jal printf
```

Loop	0x400004
Exit	

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
      add $t1, $t1, $s6
      lw $t0, 0($t1)
      bne $t0, $s5, Exit
      addi $s3, $s3, 1
      j Loop
Exit: sub $s3, $s4, $s5
      jal printf
```

PLC = 400018 →

Loop	0x400004
Exit	

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
    jal printf
```

PLC = 40001c →

Exit:

Loop	0x400004
Exit	

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
    jal printf
```

PLC = 40001c →

Exit:

Loop	0x400004
Exit	0x 40001c

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop
Exit: sub $s3, $s4, $s5
    jal printf
```

PLC = 400100 →

Loop	0x400004
Exit	0x 40001c

Tabela de símbolos

Código assembly

# Montador: exemplo de montagem

```
. Text 0x400000
    add $s5, $s1, $s2
Loop: sll $t1, $s3, 2
      add $t1, $t1, $s6
      lw $t0, 0($t1)
      bne $t0, $s5, Exit
      addi $s3, $s3, 1
      j Loop
Exit: sub $s3, $s4, $s5
      jal printf
```

PLC = 400100 →

Loop	0x400004
Exit	0x 40001c
printf	?

Tabela de símbolos

Código assembly

# **Montador:**

## **passo 2 - tradução de instruções**

- **Percorre cada linha do arquivo**
  - Para cada elemento, procura sua codificação binária
    - » Tabela de opcodes
    - » Tabela de registradores
    - » Tabela de símbolos
- **Insere a codificação do elemento**
  - No campo próprio do formato de instrução
- **Referências não resolvidas anotadas**
  - Para serem resolvidas pelo ligador



# Montador: recursos adicionais

- **Diretivas de organização da memória**

**.data**

Itens subsequentes armazenados no segmento de dados

**.text**

Itens subsequentes armazenados no segmento de texto

**. byte b1, ..., bn**

Armazena n valores em bytes sucessivos da memória

**. word w1, ..., wn**

Armazena n valores em palavras sucessivas da memória

**.asciiz str**

Armazena o string str em memória (terminando-o com caracter nulo)

# Montador: recursos adicionais

- **Diretivas de organização da memória**
  - Torna mais amigável a inspeção de código
  - Exemplo:  
    .asciiz "The quick brown fox jumps over the lazy dog"
  - Contra-exemplo:  
    .byte 84, 104, 101, 32, 113, 117, 105, 99  
    .byte 107, 32, 98, 114, 111, 119, 110, 32  
    .byte 102, 111, 120, 32, 106, 117, 109, 112  
    .byte 115, 32, 111, 118, 101, 114, 32, 116  
    .byte 104, 101, 32, 108, 97, 122, 121, 32  
    .byte 100, 111, 103, 0

# Montador: recursos adicionais

- **Macros**
  - **Utilitário para encapsular**
    - » Uma sequência frequente de instruções
  - **Princípio:**
    - » Detecção de padrão e substituição
  - **Requisito:**
    - » Montador precisa embutir processador de macros
  - **Exemplo: Imprimir vários números inteiros**
    - » Armazenados em registradores
    - » Usando uma rotina da biblioteca: printf
      - String formatador e valor(es) a imprimir

# Montador: recursos adicionais

```
.data
int_str:.asciiz "%d"
.text
la $a0, int_str
mov $a1, $7
jal printf
```

```
.data
int_str:.asciiz "%d"
.text
.macro print_int($arg)
la $a0, int_str
mov $a1, $arg
jal printf
.end_macro

print_int($7)
print_int($t0)
print_int($a0)
```

# Montador: recursos adicionais

```
la $a0, int_str  
mov $a1, $7  
jal printf
```

```
.data  
int_str:.asciiz "%d"  
.text  
.macro print_int($arg)  
la $a0, int_str  
mov $a1, $arg  
jal printf  
.end_macro  
print_int($7)  
print_int($t0)  
print_int($a0)
```

# Montador: recursos adicionais

```
la $a0, int_str  
mov $a1, $7  
jal printf
```

```
la $a0, int_str  
mov $a1, $t0  
jal printf
```

```
.data  
int_str:.asciiz "%d"  
.text  
.macro print_int($arg)  
la $a0, int_str  
mov $a1, $arg  
jal printf  
.end_macro  
print_int($7)  
print_int($t0)  
print_int($a0)
```

# Montador: recursos adicionais

```
la $a0, int_str  
mov $a1, $7  
jal printf
```

```
la $a0, int_str  
mov $a1, $t0  
jal printf
```

```
la $a0, int_str  
mov $a1, $a0  
jal printf
```

??

```
.data  
int_str:.asciiz "%d"  
.text  
.macro print_int($arg)  
la $a0, int_str  
mov $a1, $arg  
jal printf  
.end_macro  
print_int($7)  
print_int($t0)  
print_int($a0)
```

# Montador: recursos adicionais

Limitação: Não pode ser usada para imprimir um número armazenado num registrador arbitrário  
(pois a macro usa um dos registradores)

```
la $a0, int_str  
mov $a1, $a0  
jal printf
```

??

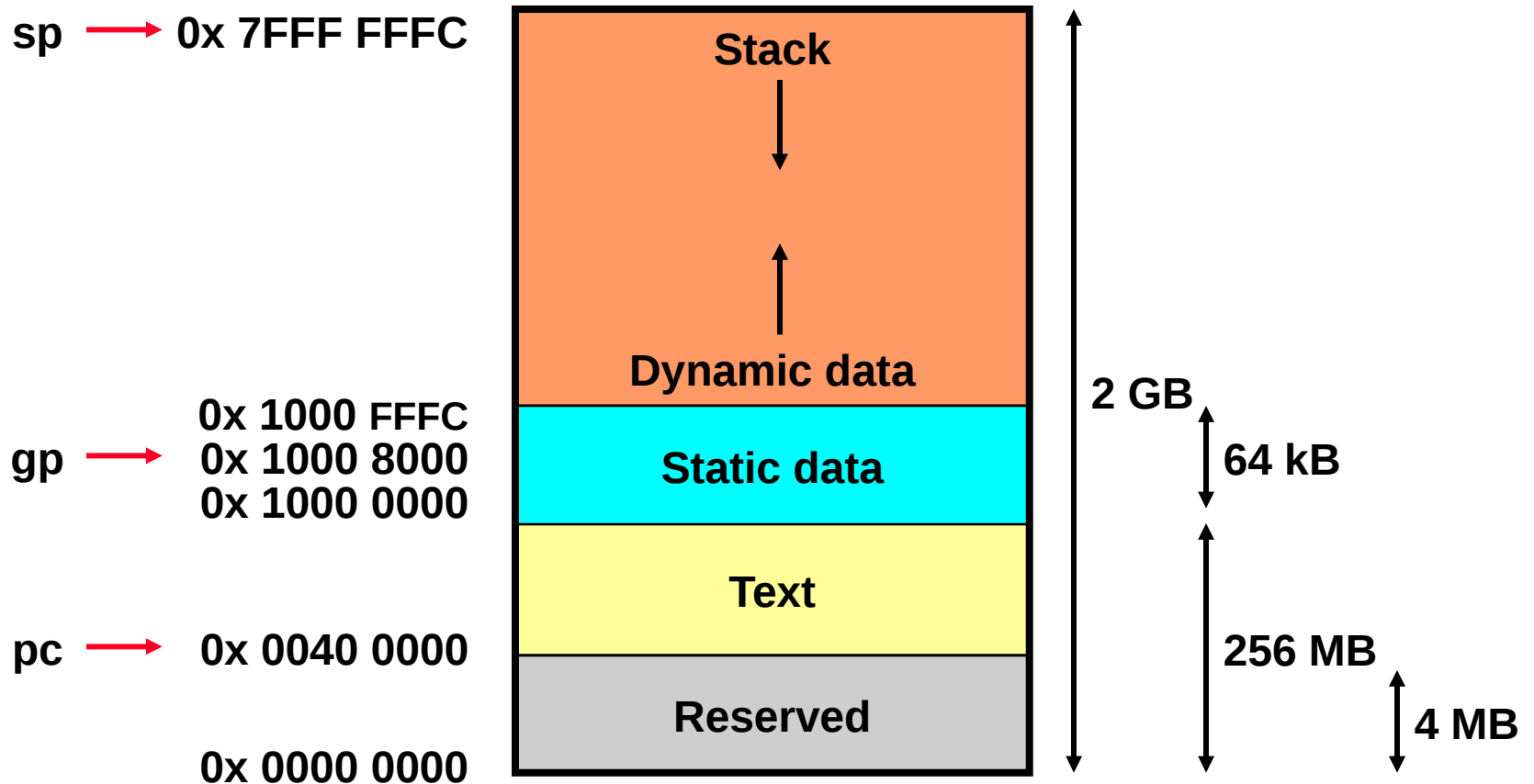
```
.data  
int_str:.asciiz "%d"  
.text  
.macro print_int($arg)  
la $a0, int_str  
mov $a1, $arg  
jal printf  
.end_macro  
print_int($7)  
print_int($t0)  
print_int($a0)
```



# Montador: recursos adicionais

- **Pseudo-instruções**
  - Abstração de detalhes de instruções nativas
  - Tabela de pseudo-instruções
  - Exemplos:
    - » `la $t0, label`
    - » `bge $t0, $t1, exit`
  - Mais seguras e poderosas que macros
    - » Registrador reservado (`$at` no MIPS)

# Uso da memória: layout

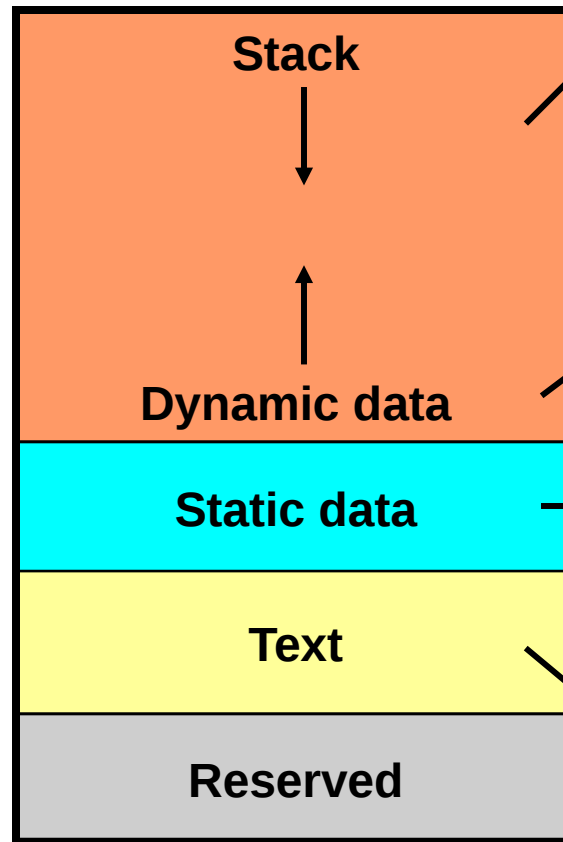


# Uso da memória: alocação

sp → 0x 7FFF FFFC

gp → 0x 1000 FFFC  
0x 1000 8000  
0x 1000 0000

pc → 0x 0040 0000  
  
0x 0000 0000



## Pilha:

dados locais não  
alocáveis em  
registrador na  
rotina chamada e  
contexto da rotina  
chamadora

## Heap:

alocação dinâmica  
(exemplo: listas  
encadeadas)

## Static data:

constantes e  
variáveis  
estáticas  
declaradas

## Text:

código do  
programa


# Uso da memória: interface HW/SW

- Segmento de dados começa em 0x10000000
- Exemplo: lw \$v0, 0x10008020
- Alternativa 1
  - Degenerando modo de endereçamento:
    - » lw \$v0, 0x10008020(\$zero)
  - Somente com o deslocamento de 16 bits
    - » Impossível

# Uso da memória: interface HW/SW

- Segmento de dados começa em 0x10000000
- Exemplo: lw \$v0, 0x10008020
- Alternativa 2
  - Harmonizando restrição de formato e layout

```
lui $at, 0x1000
ori $at, $at, 0x8020
lw $v0, 0($at)
```


  - Acesso com três instruções
    - » Ineficiente

# Uso da memória: interface HW/SW

- Segmento de dados começa em 0x10000000
- Exemplo: lw \$v0, 0x10008020
- Alternativa 3
  - Acelerando o acesso a dados estáticos
  - \$gp apontando para o meio da faixa (0x10008000)  
lw \$v0, 0x0020(\$gp)
  - Acesso em uma única instrução
    - » Eficiente

# Uso da memória: exemplos

```
main() {  
    static int counter = 1;  
    void F ( int i ) {  
        int x;  
        int A1[ 10 ];  
        int *A2 = new int [ 10 ];  
        ...  
        ...  
        delete [ ] A2;  
    }  
    ...  
    F(4)  
    ...  
    return(0);  
}
```

Dados estáticos

Pilha

Heap

# Uso da memória: exemplos

```
main() {  
    static int counter = 1;  
    void F ( int i ) {  
        int x;  
        int A1[ 10 ];  
        int *A2 = new int [ 10 ];  
        ...  
        delete [ ] A2;  
    }  
    ...  
    F(4)  
    ...  
    return(0);  
}
```

Memory leak



# Uso da memória: exemplos

```
class CLASS_A {
public: int  CLASS_A() {
            number_of_objects++;
            instance_number = number_of_objects;
            int *A2 = new int [ 10 ];
        };
    int ~CLASS_A() {
            number_of_objects --;
            delete [ ] A2;
        };
    ...
private: int static number_of_objects = 0;
        int instance_number;
        ...
}
CLASS_A first_instance_of_A;
CLASS_A second_instance_of_A;
main() {
    ...
}
```

Dados estáticos

Pilha

Heap

# **Juntando tudo: Formato de arquivo objeto (unix)**

- **Object file header**
  - » Tamanho e posição das outras seções do arquivo
- **Text segment**
  - » Código em linguagem de máquina
- **Static data segment**
  - » Dados alocados durante todo o programa
- **Relocation information**
  - » Instruções e dados que dependem de endereços absolutos
- **Symbol table**
  - » Lista de referências externas não resolvidas
- **Debugging information**
  - » Sumário de como módulos foram compilados