ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Computer Architecture (CO2008)

Bài báo cáo Chapter 2.3

Họ và tên Nguyễn Phúc Tiến
MSSV 2014725
Lớp L03

Ngày 26 tháng 4 năm 2022

Mục lục

1 Leaf function (hàm lá)

1.1 Đề bài

Chuyển thủ tục "reverse" (đảo thứ tự chuỗi) từ ngôn ngữ C sang hợp ngữ MIPS. Thủ tục reverse được gọi khi thực thi lệnh jal reverse từ vùng .text. cArray, cArray_size được gắn vào các thanh ghi thanh ghi $a0, $a1. Giá trị trả về (nếu có) chứa vào $v0. Xuất chuỗi ra console.

```
char[] cArray = "Computer Architecture 2020"
int cArray_size = 26;
void reverse(char[] cArray, int cArray_size)
{
    int i;
    char temp;
    for (i = 0 ; i < cArray_size/2; i ++)
    {
        temp = cArray[i];
        cArray[i] = cArray[cArray_size -1 -i];
        cArray[cArray_size -1 - i] = temp;
    }
}
```

Lưu ý: Dùng "jal reverse" để gọi thủ tục "reverse" và dùng "jr $ra" trở về vị trí thanh ghi $ra đánh dấu.

1.2 Hiện thực

- Đầu tiên ta khai báo cArray trong .data và trong phần .text ta gắn cArray vào thanh ghi $a0 cũng như cArray_size vào thanh ghi $a1.

```
.data
cArray: .asciiz "Computer Architecture 2020"
.text
      la      $a0, cArray     #store cArray and
      li      $a1, 26         #cArray_size
```

- Sau đó sử dụng lệnh jal để jump tới "reverse" và lưu địa chỉ tiếp theo vào thanh ghi $ra.

```
      jal   reverse
```

- Sau khi thực thi lệnh jal reverse thì chương trình sẽ jump tới *reverse:*, ta tiến hành thực thi hàm tại đây.
  Ta gán index i vào thanh ghi $t0. Điều kiện khi kết thúc vòng lặp for là cArray_size/2 được gán vào thanh ghi $t1 và thanh ghi $t5 có giá trị (cArray/2 - 1) được dùng khi ta lấy các index tại các kí tự cuối.

```
reverse:
      li  $t0, 0    #index i
      div $t1, $a1, 2 #t1 = size of cArray / 2
      subi  $t5, $a1, 1 #t2 = cArray_size - 1
```

- Ta tạo một vòng lặp là *reverse_ loop:* với điều kiện thoát ra là $t0 = $t1 tức index i = cArray_size/2.

```
reverse_loop:
        beq     $t0, $t1, exit
```

- Tiếp theo ta cần lấy giá trị đầu tiên của chuỗi và giá trị cuối cùng (tương ứng với index i) để hoán vị cho nhau sau mỗi lần lặp.
  Thanh ghi $t2 là input cArray sau khi cộng index i ($t0). Sau đó ta sử dụng lb để lấy giá trị đầu tiên của thanh ghi $t2 (0($t2) và được lưu vào thanh ghi $t4. Giá trị cuối cùng sẽ lấy từ cArray với index là $t5 và được lưu vào thanh ghi $s0.

```
        add     $t2, $a0, $t0       #increase cArray by 1
        lb      $t4, 0($t2)         #t4 = cArray[i] (temp)
        lb      $s0, cArray($t5)    #lay ki tu cuoi cung va luu vao thanh ghi s0
```

– Sau khi lấy được 2 kí tự đầu và cuối, ta tiến hành hoán vị chúng cho nhau bằng cách sử dụng lệnh sb (store byte).

```
    sb      $s0, cArray($t0)    #luu ki tu cuoi vao index dau tien moi vong lap
    sb      $t4, cArray($t5)    #luu ki tu dau vao index cuoi cung moi vong lap
```

– Sau khi đã hoán vị xong, ta cộng index đầu tiên là i lên 1 (i++), giảm index cuối cùng đi 1 (cArray_size - 1)−− và tiến hành jump không điều kiện lại vòng lặp của chúng ta.

```
    addi    $t0, $t0, 1
    subi    $t5, $t5, 1
    j       reverse_loop
```

– Nếu điều kiện thanh ghi $t0 = với $t1 tức là index i = cArray_size/2 thì ta rẽ nhánh tới exit (thoát khỏi vòng lặp) và jump đến địa chỉ $ra đã được đánh dấu

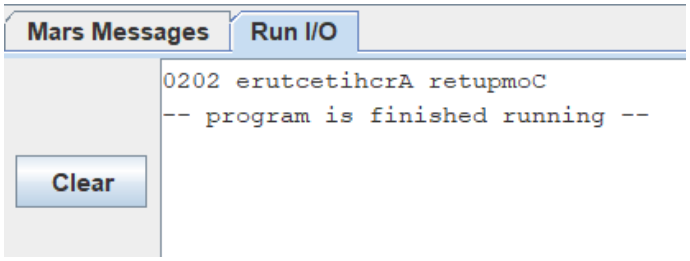
```
exit:
    add     $v0, $a0, $zero
    jr      $ra
```

Sau khi jump về địa chỉ $ra (sau lệnh jal reverse) ta tiến hành in chuỗi ra console và kết thúc chương trình.

```
    #print cArray
    li      $v0, 4
    la      $a0, cArray
    syscall

    li      $v0, 10 #exit()
    syscall
```

Kết quả:



2 Non-leaf function (là hàm/thủ tục gọi một hàm/thủ tục bên trong)

2.1 Đề bài

Chuyển thủ tục range từ C sang hợp ngữ MIPS tương đương.

```
int iArray[10];
int iArray_size = 10;
int range(iArray, iArray_size)
{
    int temp1 = max(iArray, iArray_size);
    int temp2 = min(iArray, iArray_size);
    int range = temp1 - temp2;
    return range;
}
```

Chương trình bắt đầu từ vùng .text, sau đó nó gọi hàm range. Trong hàm range lại gọi 2 hàm con là max và min. Giả sử địa chỉ và kích thước iarray được gắn lần lượt vào các thanh ghi $a0, $a1. Xuất giá trị range ra ngoài console.
Lưu ý: Khi gọi các hàm/thủ tục thanh ghi $ra sẽ tự đánh dấu lệnh tiếp theo như là vị trí trở về. Do đó trước khi gọi hàm con trong hàm range thì sinh viên cần lưu lại giá trị thanh ghi $ra trong stack. Sau khi thực thi xong, sinh viên cần phục hồi lại giá trị cho thanh ghi $ra từ stack. Dùng "jal range", "jal max", "jal min" để gọi thủ tục range, max, min. Dùng "jr $ra" để trở về vị trí lệnh mà thanh ghi $ra đã đánh dấu.

2.2 Hiện thực

– Đầu tiên trong phần .data ta khai báo các string để giao tiếp với người dùng và mảng iArray có độ rộng là 40 bytes tương ứng với 10 giá trị số nguyên.

```
.data
    message: .asciiz "Nhap vao mang co 10 phan tu: \n"
    range_message: .asciiz "\nRange giua max va min la: "
    iArray: .space 40
```

– Tiếp đến ta tiến hành tạo một loop để người dùng nhập mảng iArray vào.

```
.text
      li      $v0, 4
      la      $a0, message
      syscall
      #a loop to enter user numbers into iArray
      enter_Array:
            beq     $t0, 40, exit_enter
            li      $v0, 5
            syscall
            sb      $v0, iArray($t0)
            addi    $t0, $t0, 4
            j       enter_Array
      exit_enter:
```

– Sau khi nhập, ta cũng sử dụng lệnh jal range để gọi thủ tục range.

```
      la      $a0, iArray   #load address of iArray
      li      $a1, 10       #size of iArray

      jal range
```

– Chương trình sẽ jump đến label *range*. Trong đó có gọi hàm khác, nên ta sẽ lưu địa chỉ trở về $ra vào trong stack và sử dụng lệnh jal max để gọi thủ tục max.

```
range:
      addi    $sp, $sp, -4    #adjust stack for 1 item
      sw      $ra, 0($sp)     #store range return address
      jal     max
```

– Tìm **max**, đầu tiên ta gán giá trị max là iArray[0] vào thanh ghi $t1, $t0 là index của ta. Vì đây là mảng cho nên 4 bytes sẽ tương ứng với 1 ô chứa giá trị số nguyên. index bắt đầu từ 4 vì max đã được gán vào index 0.

```
#find max of iArray
max:
      li      $t0, 4
      lb      $t1, iArray($zero)  #t1 = max
```

Sau đó sẽ sử dụng 1 vòng lặp để tìm ra max, vòng lặp kết thúc khi index i = 10 tương ứng với $t0 = 40 (10x4).

```
      find_max:
            beq $t0, 40, exit_find_max
```

Ta load giá trị iArray[i] và gán vào thanh ghi $s0 để so sánh với max. Nếu iArray[i] < max thì ta sẽ cho $t0 += 4, ngược lại max = iArray[i]. Khi kết thúc so sánh, ta cho index++ và jump không điều kiện lại vòng lặp của ta.

```
    lb      $s0, iArray($t0)
    ble     $s0, $t1, find_max  #if iArray[i] < max, index++
    add     $t1, $s0, $zero     #else max = iArray[i]
    addi    $t0, $t0, 4         #index += 4
    j       find_max
```

Khi điều kiện thoát vòng lặp thỏa, thì chương trình sẽ rẽ nhánh đến exit_find_max và ta sẽ sử dụng jr $ra để trở lại địa chỉ mà thanh ghi $ra đã đánh dấu (sau lệnh jal max). Vì địa chỉ trở về của range đã được lưu vào stack nên ta không lo ngại việc ghi đè giá trị của thanh ghi $ra.

```
    exit_find_max:
    jr      $ra
```

– Sau khi trở lại sau câu lệnh jal max ta tiếp tục dùng jal min để gọi thủ tục min.

```
    jal     min
```

– Tìm **min**, tương tự như việc tìm **max** ta chỉ cần câu lệnh so sánh bé hơn thành so sánh lớn hơn và lưu các biến tạm vào trong thanh ghi khác. Ở đây ta sẽ cho thanh ghi $t2 chứa giá trị min và câu lệnh ble thành bge, còn lại thì hầu như không thay đổi gì.

```
  li      $t0, 4              #reset value of index
  lb      $t2, iArray($zero)  #t2 = min
  find_min:
        beq     $t0, 40, exit_find_max
        lb      $s0, iArray($t0)
        addi    $t0, $t0, 4
        bge     $s0, $t2, find_min  #if iArray[i] > min, index++
        add     $t2, $s0, $zero     #else min = iArray[i]
        j     find_min
  exit_find_min:
  jr      $ra
```

– Sau khi trở lại sau câu lệnh jal min cũng như đã tìm được max và min. Ta load lại giá trị thanh ghi $ra chứa địa chỉ trở về của range và pop giá trị đó khỏi stack.

```
  lw      $ra, 0($sp) #load range return address
  addi    $sp, $sp, 4 #pop 1 item from stack
  jr      $ra
```

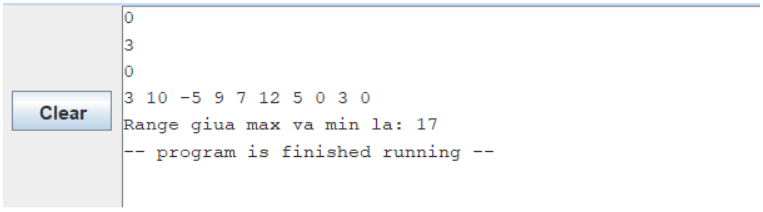
– Cuối cùng, chương trình của ta jump về range, ta tiến hành tính range bằng max - min và lưu vào thanh ghi $t3. Sau đó in kết quả ra console và kết thúc chương trình.

```
  li      $v0, 4
  la      $a0, range_message
  syscall

  sub     $t3, $t1, $t2     #range = max - min
  li      $v0, 1            #print range
```

```
7    move    $a0, $t3
8    syscall
9
10   li      $v0, 10    #exit()
11   syscall
```

Kết quả: Khi ta nhập các giá trị vào mảng là: 3, 10, -5, 9, 7, 12, 5, 0, 3, 0 thì range = 12 - (-5) = 17.



3 Giải quyết bài toán

Cho đoạn code hợp ngữ MIPS bên dưới

```
1          addi    $a0, $zero, 100 // upper threshold
2          addi    $a1, $zero, 0 // count variable
3          add     $a2, $zero, $zero // sum initialization
4  loop:
5          beq     $a0, $a1, exit
6          add     $a2, $a2, $a1
7          addi    $a1, $a1, 1
8          j       loop
9  exit:
```

a) Xác định giá trị của thanh ghi $a2 sau khi thực thi đoạn code trên.

– $a0 = 100, $a1 = $a2 = 0. Khi $a0 = $a1 thì kết thúc. Ngược lại $a2 += $a1 và $a1 += 1. Giá trị của thanh ghi $a2 là tổng của giá trị thanh ghi $a1 sau mỗi lần lặp. Vòng lặp của ta sẽ lặp từ 0->99 cho nên tổng sẽ là: $0 + 1 + 2 +..+ 99 = \frac{n*(n+1)}{2} = \frac{99*100}{2} = 4950.$

b) Xác định tổng số chu kỳ thực thi khi thực thi đoạn chương trình trên. Giả sử CPI của các lệnh là 1.

– Đầu tiên có 3 câu lệnh được thực hiện trước khi bắt đầu vòng lặp.

– Trong vòng lặp loop có 4 câu lệnh với số lần lặp là 100 (khi $a0 = [0, 99]), khi $a0 = 100 thì chỉ có câu lệnh beq được thực hiện. Vậy tổng số câu lệnh mà vòng lặp thực hiện = $100*4+1$.
⇒ Tổng số chu kì = (3 + 100 * 4 + 1) * 1 = 404 chu kì.

c) Giả sử vùng .text (text segment - vùng để chứa các lệnh thực thi) bắt đầu từ địa chỉ 0x10080000. Xác định mã máy của lệnh "j loop" ở dạng HEX.

– Địa chỉ mã hex của loop là: 0x1008000C. Ta chuyển sang mã nhị phân và thực hiện các bước.

+ Drop 4 bit cao: 0000 0000 1000 0000 0000 0000 1100

+ Drop tiếp 2 bit thấp: 0000 0000 1000 0000 0000 0000 11 còn 26 bit.

6 bit Opcode của lệnh *jump* là 2 tương ứng với 000010 với 26bit address sau khi drop 4 bit cao và 2 bit thấp. Ta ghép lại và chuyển sang mã hex.

Opcode	Address
000010	00000000100000000000000011

jump 0x1008000C
0000 1000 0000 0010 0000 0000 0000 0011
0 8 0 2 0 0 0 3

⇒ Mã máy của lệnh "j loop" ở dạng HEX là: 0x08020003