**CO2008 - KIẾN TRÚC MÁY TÍNH**

**Khoa Khoa học và Kỹ thuật Máy tính**
**Đại học Bách Khoa – Tp.HCM**

9/2017

Bài thực hành số 4

# KIẾN TRÚC TẬP LỆNH MIPS: GỌI HÀM(LẬP TRÌNH CẤU TRÚC), THỜI GIAN THỰC THI

## Mục tiêu

- Chuyển từ ngôn ngữ cấp cao (C) sang hợp ngữ MIPS.
- Sử dụng lệnh điều khiển (nhảy, rẽ nhánh) để lập trình cấu trúc.
- Biết nguyên lý gọi hàm. Sử dụng các lệnh gọi hàm jal, jr.
- Tính toán thời gian thực thi của chương trình.

## Yêu cầu

- Xem cách dùng các lệnh (set, branch, jump, load, store, **jal, jr**) trong slide và trong file tham khảo.
- Nộp các file code hợp ngữ Bai*.asm (để trong thư mục Lab4_MSSV)

## Review: MIPS instruction types

R-type


Kiểu I-type


Kiểu J-type


- Op (operation code) Mã lệnh, dùng để xác định kiểu lệnh, và lệnh thực thi (Kiểu R thì Op = 0).
- Rs, Rt, Rd (register): Trường xác định thanh ghi (trường thanh ghi 5 bit tương ứng với 32 thanh ghi).
- Shamt (shift amount): Xác định số bits dịch trong các lệnh dịch bit.
- Function: Xác định toán tử(operator hay còn gọi là lệnh) trong kiểu lệnh R.
- Immediate: Số trực tiếp, địa chỉ.

## Bài tập và Thực hành

Sinh viên chuyển chương trình C bên dưới qua hợp ngữ MIPS tương ứng.

1. Leaf function (hàm lá)
   Chuyển thủ tục "swap" từ ngôn ngữ C sang hợp ngữ MIPS. Thủ tục swap được gọi khi thực thi lệnh **`jal`** `swap` từ vùng .text. cArray, first_idx, swapped_idx được gắn vào các thanh ghi thanh ghi $a$_0$, $a$_1$, và $a$_2$. Giá trị trả về chứa vào $v$_0$. Xuất chuỗi ra console.

```
char cArray = "Computer Architecture 2017"
int swap(cArray, int first_idx, int swapped_idx)
{
   char temp;
   if (first_idx < 0 || swapped_idx < 0){
      return -1;
   }
   else{
      temp = cArray[first_idx];
      cArray[first_idx] = cArray[swapped_idx];
      cArray[swapped_idx] = temp;
   }
   return 0;
}
```

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

2. Non-leaf function (là hàm/thủ tục gọi một hàm/thủ tục bên trong).
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 $\$a_0$, $\$a_1$. 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.
**Để lưu(push) giá trị $r$_a$ vào stack, sinh viên có thể dùng các lệnh sau:**

```
addi $sp, $sp, -4 # adjust stack for 1 item
sw $ra, 0($sp)    # save return address
```

**Để phục hồi(pop) $r$_a$ từ stack, sinh viên có thể dùng các lệnh sau:**

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

3. Cho đoạn code hợp ngữ MIPS bên dưới

```
        addi $a0, $zero, 10     // upper threshold
        addi $a1, $zero, 0      // count variable
        add  $a2, $zero, $zero  // sum initialization
loop:
        beq  $a0, $a1, exit
        add  $a2, $a2, $a1
        addi $a1, $a1, 1
        j loop
exit:
```

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

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

(c) Giả sử vùng .text (text segment - vùng để chứa các lệnh thực thi) bắt đầu từ địa chỉ 0x10040000. Xác định mã máy của lệnh `"j loop"` ở dạng HEX.

## Instructions [Quick reference]

| **Syntax** | **Purpose** | **Description** |
|---|---|---|
| Set instructions | | |
| slt Rd, Rs, Rt | Rd = (Rs < Rt) ? 1 : 0 | [Signed]Rd = 1 when Rs < Rt, otherwise Rd = 0 |
| sltu Rd, Rs, Rt | Rd = (Rs < Rt) ? 1 : 0 | [Unsigned] Rd = 1 when Rs < Rt, otherwise Rd = 0 |
| Branch, Jump instructions | | |
| beq Rs, Rt, label | if (Rs == Rt) PC ← Pc + label x 4 | Branch to label if Rs == Rt |
| bne Rs, Rt, label | if (Rs != Rt) PC ← PC + label x 4 | Branch to label if Rs != Rt |
| bltz Rs, label | if (Rs < 0) PC ← PC + label x 4 | Branch to label if Rs < 0 |
| blez Rs, label | if (Rs <= 0) PC ← PC + label x 4 | Branch to label if Rs <= 0 |
| bgtz Rs, label | if (Rs > 0) PC ← PC + label x 4 | Branch to label if Rs > 0 |
| bgez Rs, label | if (Rs >= 0) PC ← PC + label x 4 | Branch to label if Rs >= 0 |
| j label | PC ← label | Unconditional Jump to label |
| Calling function | | |
| jr Rs | PC ← Rs | Return to address where the register Rs point to |
| jal label | $ra ← PC+4, PC ← label | Calling function, PC goto label, $r$_a$ hold the next instruction |
| jalr Rs | $ra ← PC+4, PC ← Rs | Calling function, PC goto address which the content is store in Rs, $r$_a$ holds the next instruction |