**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 3
# KIẾN TRÚC TẬP LỆNH MIPS: CÁC LỆNH ĐIỀU KIỂN

## Mục tiêu

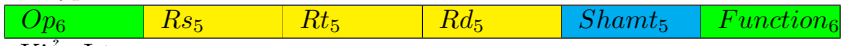
- Chuyển từ ngôn ngữ cấp cao (C) sang hợp ngữ MIPS.
- Sử dụng lệnh điều khiển (nhảy, rẽ nhánh) để điều kiển luồng chương trình.

## Yêu cầu

- Xem cách dùng các lệnh (set, branch, jump, load, store) trong slide và trong file tham khảo.
- Nộp các file code hợp ngữ Bai*.asm (để trong thư mục Lab2_MSSV)

## Kiểu lệnh

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.

**Note:** SV xác định kiểu lệnh của các lệnh MIPS trong bài tập của mình, xác định các trường trong một lệnh kiểu lệnh đã được xác định.
Ví dụ: `addi $t1, $0, 0 #kieu I, rs:$t1, rt:$0, immediate: 10`

## Bài tập và Thực hành

Lập trình có cấu trúc.
**Sinh viên chuyển các cấu trúc sau của ngôn ngữ C qua ngôn ngữ assembly**. Tham khảo hình ảnh về các cấu trúc ở cuối bài thực hành.

**Bài 1**: Phát biểu IF-ELSE (1)

```
if( a >= 0) { Print string: "Computer Architecture is so easy!"}
else        { Print string: "you are right!"}
```

**Bài 2**: Phát biểu IF-ELSE (2)

```
if( a >= -5 && a <= 5) { a = b + c;}
else                   { a = b - c;}
```

**Bài 3**: Phát biểu SWITCH-CASE
Hiện thực phát biểu switch-case bên dưới bằng hợp ngữ. Cho biết b = 10, c = 5. Giá trị input nhập từ người dùng.

```
switch (input)
{
   case 0: a = b + c; break;
   case 1: a = b - c; break;
   case 2: a = b x c; break;
   case 3: a = b / c; break;
   default: NOP; break;
}
```

**Bài 4**: Vòng lặp FOR - xác định chuỗi Fibonacci bằng vòng lặp. Nhập vào n(nguyên dương), xuất ra số Fibonacci Fn.

```
if     (n == 0) {return 0;}
else if(n == 1) {return 1;}
else{
   f0= 0; f1 = 1;
   for ( $t0 = 2; $t0 <= n; $t0++){
      fn = fn-1 + fn-2
   }
}
return fn;
```

**Note:** sinh viên có thể là theo cách riêng để tìm ra số Fibonacci Fn.
Dãy số Fibonacci http://en.wikipedia.org/wiki/Fibonacci_number

| $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 |
| $F_{10}$ | $F_{11}$ | $F_{12}$ | $F_{13}$ | $F_{14}$ | $F_{15}$ | $F_{16}$ | $F_{17}$ | $F_{18}$ | $F_{19}$ |
| 55 | 89 | 144 | 233 | 377 | 610 | 987 | 1597 | 2584 | 4181 |

**Bài 5**: Vòng lặp WHILE
Xác định vị trí chữ 'e' đầu tiên trong chuỗi "Computer Architecture CSE-HCMUT".

```
i = 0;
while( charArray[i] != 'e' && charArray[i] != '\0'){
  i = i + 1;
}
// if not found return -1
```

Xuất ra giá trị index của ký tự 'e'. Nếu không tìm thấy thì xuất ra -1.

## Làm thêm

1. ENDIANESS.
Cho mảng số nguyên bên dưới.

```
.data
intArray: .word 0xCA002017, 0xC0002008
.text
        la  $a0, intArray
        lb  $t0, 0($a0)
        lb  $t1, 1($a0)
        lb  $t2, 2($a0)
        lb  $t3, 3($a0)
        lbu $t4, 0($a0)
```

```
lbu $t5, 1($a0)
lbu $t6, 2($a0)
lbu $t7, 3($a0)
```

(a) Giả sử MIPS được thiết kế theo kiểu BIG ENDIAN, xác định giá trị các ô nhớ (theo byte) của mảng trên.

(b) Giả sử MIPS được thiết kế theo kiểu LITTLE ENDIAN, xác định giá trị các ô nhớ (theo byte) của mảng trên.

(c) Xác định giá trị các thanh ghi $t của đoạn code bên dưới, giả sử MIPS được thiết kế theo kiểu BIG ENDIAN.

(d) Xác định giá trị các thanh ghi $t của đoạn code bên dưới, giả sử MIPS được thiết kế theo kiểu LITTLE ENDIAN.

2. MEMORY ALIGN.
Cho đoạn code mips bên dưới

```
.data
int_1:  .word 0xCA002017
char_1: .byte 0xFF
int_2:  .word 2017
char_2: .byte 0xCA 0xFE 0xED
.text
        la  $a0, int_1
        lw  $t0, 0($a0)
        lw  $t1, 1($a0)
        lh  $t2, 2($a0)
        lh  $t3, 3($a0)
        lb  $t4, 0($a0)
        lb  $t5, 1($a0)
```

(a) Xác định nội dung của vùng nhớ dữ liệu và xác định các lệnh sẽ gây ra lỗi khi thực thi, giải thích. Biết MIPS chuẩn được thiết kế theo kiểu BIG ENDIAN.

(b) Xếp lại dữ liệu sao cho bộ nhớ tối ưu hơn (trong kiến trúc 32 bit).

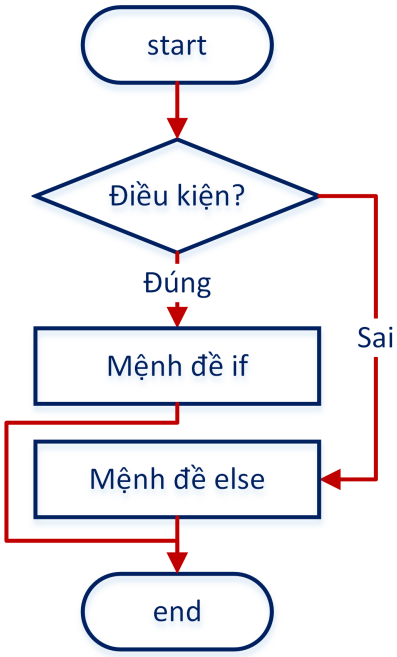
## Tập lệnh [tham khảo nhanh]

| Cú pháp | Ảnh hưởng | Mô tả |
|---|---|---|
| slt Rd, Rs, Rt | Rd = (Rs < Rt) ? 1 : 0 | [Có dấu]Rd = 1 khi Rs < Rt, ngược lại Rd = 0 |
| sltu Rd, Rs, Rt | Rd = (Rs < Rt) ? 1 : 0 | [Không dấu] Rd = 1 khi Rs < Rt, ngược lại Rd = 0 |
| Lệnh nhảy, rẽ nhánh | | |
| beq Rs, Rt, label | if (Rs == Rt) PC ← label | Rẽ nhánh đến label nếu Rs == Rt |
| bne Rs, Rt, label | if (Rs != Rt) PC ← label | Rẽ nhánh đến label nếu Rs != Rt |
| bltz Rs, label | if (Rs < 0) PC ← label | Rẽ nhánh đến label nếu Rs < 0 |
| blez Rs, label | if (Rs <= 0) PC ← label | Rẽ nhánh đến label nếu Rs <= 0 |
| bgtz Rs, label | if (Rs > 0) PC ← label | Rẽ nhánh đến label nếu Rs > 0 |
| bgez Rs, label | if (Rs >= 0) PC ← label | Rẽ nhánh đến label nếu Rs >= 0 |
| b label<br>j label | PC ← label<br>PC ← label | Rẻ nhánh không điều kiện đến label<br>Nhảy không điều kiện đến label |
| Gọi hàm | | |
| jr Rs | PC ← Rs | Trở về vị trí thanh ghi Rs trỏ đến |
| jal label | $ra ← PC+4, PC ← label | Gọi hàm label, khi đó $ra nắm vị trí lệnh tiếp theo |
| jalr Rs | $ra ← PC+4, PC ← Rs | Gọi hàm Rs đang trỏ đến, khi đó $ra nắm vị trí lệnh tiếp theo |

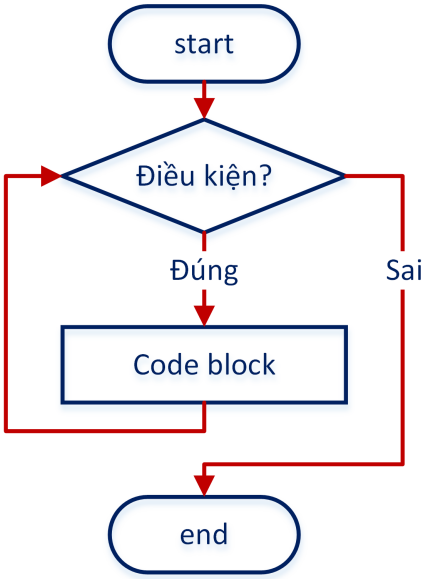
Bảng. 1: Danh sách 32 thanh ghi

| **REGISTERS** | | |
|---|---|---|
| Theo số | Theo tên | Miêu tả |
| 0 | zero | Always equal to zero |
| 1 | at | Assembler temporary; used by the assembler |
| 2-3 | v0-v1 | Return value from a function call |
| 4-7 | a0-a3 | First four parameters for a function call |
| 8-15 | t0-t7 | Temporary variables; need not be preserved |
| 16-23 | s0-s7 | Function variables; must be preserved |
| 24-25 | t8-t9 | Two more temporary variables |
| 26-27 | k0-k1 | Kernel use registers; may change unexpectedly |
| 28 | gp | Global pointer |
| 29 | sp | Stack pointer |
| 30 | fp/s8 | Stack frame pointer or subroutine variable |
| 31 | ra | Return address of the last subroutine call |

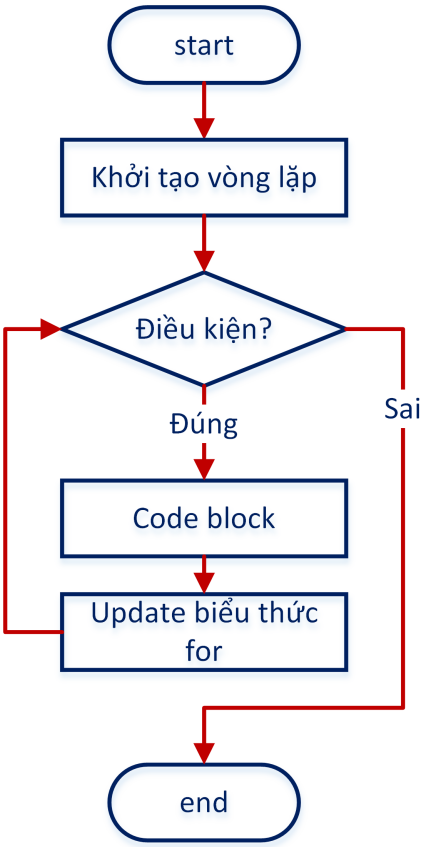
**Sơ đồ cấu trúc của phát biểu (if-else, for, while, do-while)**



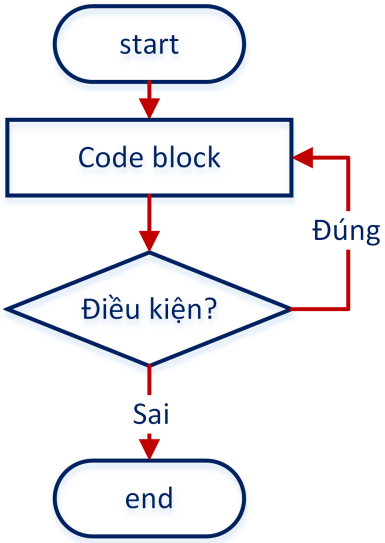

Hình. 1: Cấu trúc IF ELSE




Hình. 2: Cấu trúc WHILE




Hình. 3: Cấu trúc FOR




Hình. 4: Cấu trúc DO WHILE