Hợp ngữ MIPS

Hợp ngữ và kiến trúc máy tính được chi làm 2 loại:

- ✓ **CICS** (**Complex insrtuction set computer**): Đại diện tiêu biểu là x86 - được sử dụng trên các máy tính cá nhân và server.
  - ⇨ Hợp ngữ của **CISC** rất phức tạp
- ✓ **RISC** (**Reduced instruction set computer**): Đại diện cho **RISC** là **ARM** và **MIPS**. **ARM** được sử dụng trong các thiết bị di động và **MIPS** được sử dụng trong một số siêu máy tính, và các thiết bị như router, Nintendo 64, Sony Playstation 2.
  - ⇨ Hợp ngữ của **RISC** thì đơn giản hơn

Tên	Thanh ghi	Ý nghĩa
$zero	**0**	Thanh ghi này luôn chứa giá trị 0
$at	**1**	Assembler Temporary - Được dành riêng cho các mục đích khác, khi viết hạn chế dùng thanh ghi này
$v0, $v1	**2, 3**	Lưu giá trị trả về của hàm
$a0- $a3	**4-7**	Lưu tham số truyền vào của hàm
$t0 - $t7	**8-15**	Lưu biến tạm
$s0 - $s7	**16-23**	Lưu biến
$t8, $t9	**24, 25**	Như các $t ở trên
$k0, $k1	**26, 27**	Được dùng cho nhân HĐH sử dụng
$gp	**28**	Pointer to global area
$sp	**29**	Stack pointer
$fp	**30**	Frame pointer
$ra	**31**	Return address, sử dụng cho việc gọi hàm

R-format

R-format có 6 tham số:

Tên tham số	op	rs	rt	rd	shamt	funct
Độ dài (bit)	6	5	5	5	5	6

I-format

Lệnh I-format dùng cho thao tác giữa thanh ghi và một hằng số được lưu sẵn trong lệnh. Cấu trúc như sau:

Tên tham số	op	rs	rt	immediate
Độ dài (bit)	6	5	5	16

J-format

J-format dành cho các lệnh nhảy (`goto` trong C), có cấu trúc:

Tên tham số	op	target address
Độ dài (bit)	6	26

- 4 lệnh **add**, **sub**, **addu**, **subu** dùng để cộng/trừ giá trị của 2 thanh ghi, và lưu kết quả vào thanh ghi đích. Cú pháp:

```
<tên lệnh> <thanh ghi đích>, <thanh ghi 1>, <thanh ghi 2>
```

- 2 lệnh **addi**, **addiu** dùng để cộng một thanh ghi với 1 hằng số, rồi lưu vào thanh ghi đích. Cú pháp:

```
<tên lệnh> <thanh ghi đích>, <thanh ghi>, <hằng số>
```

➔ Khác biệt giữa **addu** và **add**: **add** sẽ báo lỗi khi có tràn số, còn **addu** thì không. Tương tự với các lệnh có **u** và không có **u** khác.

-
- Các lệnh tính toán logic:
  - Có 3 lệnh: **and**, **or**, **nor**. NOR là thao tác "NOT OR": A nor B = not (A or B). Cú pháp của 3 lệnh này tương tự như lệnh **add** ở trên.
  - Lệnh **andi** và **ori** để tính **AND/OR** của một thanh ghi với một hằng số.

- **Tính toán với các hằng số 32 bit**

- Lệnh **lui (load upper immediate)** với chức năng ghi một hằng số 16-bit vào 2 byte cao của thanh ghi, 2 byte thấp sẽ được gán bằng 0.

- Lệnh dịch

- 2 lệnh **sll** và **srl** dùng để dịch trái và dịch phải. Đây là dịch logic, các giá trị trống sau khi dịch luôn là 0.
- Cú pháp tương tự như **addi** ở trên, tuy nhiên số bit cần dịch luôn là một số không âm từ 0 đến 31.

- Mô hình bộ nhớ của MIPS

- **Quy tắc Alignment Restriction**: "Địa chỉ vùng nhớ cần truy cập phải chia hết cho kích thước cần truy cập".
- MIPS lưu trữ dữ liệu theo dạng **Big Endian**, tức là byte cao sẽ được lưu ở địa chỉ thấp. Ví dụ, số 12345678h (thập lục phân) khi được lưu trong bộ nhớ thì byte đầu tiên sẽ là 12h, byte tiếp theo là 34,…

- Lệnh load/store

Cú pháp:

```
tên lệnh r1, offset(r2)
```

Trong đó:

- ✓ **r1**: thanh ghi cần nạp dữ liệu vào / lấy dữ liệu ra.
- ✓ **r2**: thanh ghi lưu địa chỉ gốc.
- ✓ **offset**: hằng số nguyên (16 bit), giá trị này sẽ được cộng với giá trị của r2 để được địa chỉ cần nạp vào / lấy ra.

Tên các lệnh:

- ✓ **lw** (**load word**), **lh** (**load halfword**), **lb** (**load byte**): Đọc 4/2/1 byte. Đối với **lh** và **lb**, vì thanh ghi có độ dài 4 byte, nhiều hơn lượng dữ liệu đọc được nên các bit trống sẽ được gán bằng bit dấu của số đọc được.
- ✓ **lhu** (**load halfword unsigned**), **lbu** (**load byte unsigned**): tương tự như trên, tuy nhiên các bit trống được gán bằng 0.
- ✓ **sw** (**store word**), **sh** (**store halfword**), **sb** (**store byte**): lưu 4/2/1 byte dữ liệu trong thanh ghi vào bộ nhớ.
- ✓ Đối với **sh** và **sb** sẽ lưu các byte thấp trong thanh ghi vào bộ nhớ.

- Lệnh nhảy

- Có 2 lệnh nhảy là **j** và **jr**:
  - ✓ Cú pháp lệnh **j**:

```
j <địa chỉ cần nhảy tới hoặc nhãn>
```

  - ✓ **jr** cũng tương tự như **j**, tuy nhiên ta đọc địa chỉ lệnh cần nhảy đến trong một thanh ghi.
  - ✓ Lệnh **jr** sẽ gán **PC** bằng với thanh ghi được chỉ định

- ✓ Ở lệnh **j** vì, tham số truyền vào chỉ có 26 bit, mà **PC** lại có đến 32 bit nên ta tính lại **PC** như sau: **PC** = (**PC** & **0xf0000000**) | (**imm** << 2), với **imm** là tham số truyền vào.

- Lệnh rẽ nhánh

- Có 2 lệnh rẽ nhánh là **beq** (**branch if equal**) và **bne** (**branch if not equal**).
- Cú pháp:

```
<Tên lệnh> <thanh ghi 1>, <thanh ghi 2>, <địa chỉ hoặc nhãn>
```

  - ✓ Lệnh **beq** sẽ so sánh giá trị trong 2 thanh ghi, nếu bằng nhau thì nhảy đến nhãn chỉ định.
  - ✓ Lệnh **bne** thì ngược lại, nhảy khi 2 giá trị khác nhau.
  - ⇨ Khi không nhảy, chương trình sẽ thực hiện lệnh tiếp theo.

Địa chỉ truyền vào là địa chỉ tương đối và có dấu, PC sẽ được tính lại như sau: `PC = PC + 4 + imm`, với `imm` là địa chỉ truyền vào.

- Lệnh **slt** (**set on less than**): so sánh lớn hơn / bé hơn
  Cú pháp:

```
slt rt, rs, rd
```

  - ⇨ gán **rt** bằng 1 khi **rs** < **rd**, bằng 0 trong trường hợp ngược lại.
- Để so sánh không dấu, **MIPS** hỗ trợ lệnh **stlu**.

- Vị trí quay về

- lệnh **jal** (**jump and link**): **jal** sẽ gán giá trị thanh ghi **$ra** bằng với địa chỉ của lệnh tiếp theo trước khi thực hiện nhảy.

```
die:
    nor $s0, $0, $0
    jr $ra

man:
    jal die
    or $s0, $0, $0

woman:
    jal die
    lui $t0, 0x0000
    ori $s0, $t0, 0xffff
```

- $sp - Bộ nhớ stack

- **MIPS** đưa ra một số thỏa hiệp giữa hàm gọi (**caller - R**) và hàm được gọi (**callee - E**):
  - ✓ Đối với các thanh ghi **$s0 - $s7** và **$sp**, **E** phải khôi phục lại đúng giá trị ban đầu sau khi thực thi xong.
  - ✓ Đối với các thanh ghi khác: **$t**, **$v**, **$a**, **$ra**, **E** có quyền thay đổi giá trị các thanh ghi này, vì vậy **R** có trách nhiệm sao lưu và khôi phục lại các thanh ghi này trước và sau khi gọi **E** (nếu cần sử dụng).
- Để quản lý sao lưu / khôi phục các thanh ghi như yêu cầu ở trên, ta dùng bộ nhớ **stack**
- Trong **MIPS**, thanh ghi **$sp** có giá trị trỏ tới đỉnh **stack**. Ở đầu hàm, ta lưu các biến cần sao lưu vào **stack**, sau đó ở cuối hàm, ta khôi phục lại các biến đó.

- Truyền tham số - giá trị trả về

- ✓ 4 thanh ghi **$a0** đến **$a3** được quy ước dùng riêng cho các tham số truyền vào.
- ✓ 2 thanh ghi **$v0**, **$v1** được dùng cho giá trị trả về.