Cheatsheet: Kiến trúc tập lệnh (ISA) - MIPS

Kiến trúc Máy tính - Giữa kỳ 241

1. Tổng quan về ISA

- **ISA (Instruction Set Architecture)**:
  - Giao diện giữa phần cứng và phần mềm
  - Mô tả các lệnh CPU có thể thực thi và cách mã hóa
- **Thành phần chính của ISA**:
  - *Tập lệnh* (Instruction Set)
  - *Định dạng lệnh* (Instruction Format)
  - *Chế độ địa chỉ* (Addressing Modes)
  - *Thanh ghi và bộ nhớ*
- **Tầm quan trọng**: Ảnh hưởng đến hiệu năng và tính tương thích của phần mềm với phần cứng

2. Phân loại lệnh

- **Lệnh số học** (Arithmetic Instructions):
  - Ví dụ: `add`, `sub`, `mul`, `div`
  - Chức năng: Thực hiện các phép tính cơ bản
- **Lệnh logic** (Logical Instructions):
  - Ví dụ: `and`, `or`, `xor`, `not`
  - Chức năng: Thực hiện các phép toán bitwise
- **Lệnh điều khiển** (Control Instructions):
  - Ví dụ: `beq`, `bne`, `j`, `jal`, `jr`
  - Chức năng: Điều khiển luồng thực thi chương trình
- **Lệnh bộ nhớ** (Memory Instructions):
  - Ví dụ: `lw`, `sw`, `lb`, `sb`
  - Chức năng: Truyền dữ liệu giữa bộ nhớ và thanh ghi
- **Lệnh dịch chuyển và xoay** (Shift and Rotate Instructions):
  - Ví dụ: `sll`, `srl`, `sra`
  - Chức năng: Dịch bit sang trái hoặc phải
  - Dịch trái (`sll`): Dịch 1, 2, 3,... bit sang trái
  - Dịch phải (`srl`): Dịch 1, 2, 3,... bit sang phải
- **Lệnh với chữ "i"**:
  - Các lệnh như `addi`, `andi` sử dụng chế độ địa chỉ immediate
  - Ví dụ: `addi $t0, $t1, 10` (Thêm 10 vào $t1, lưu vào $t0)

3. Tập lệnh MIPS

- **Kiến trúc**: RISC (Reduced Instruction Set Computing)
- **Thanh ghi**: 32 thanh ghi 32-bit
  - `$zero`: Thanh ghi hằng số 0
  - `$at`: Thanh ghi dành riêng cho assembler
  - `$v0 - $v1`: Giá trị trả về từ hàm
  - `$a0 - $a3`: Tham số truyền vào hàm
  - `$t0 - $t9`: Thanh ghi tạm thời
  - `$s0 - $s7`: Thanh ghi lưu trữ
  - `$k0 - $k1`: Dành cho kernel
  - `$gp`: Global pointer
  - `$sp`: Stack pointer
  - `$fp`: Frame pointer
  - `$ra`: Return address

4. Định dạng lệnh trong MIPS

- **R-format** (Register format):
  - Cấu trúc: `op (6 bits) | rs (5 bits) | rt (5 bits) | rd (5 bits) | shamt (5 bits) | funct (6 bits)`
  - `op`: Mã lệnh (opcode)
  - `rs, rt`: Thanh ghi nguồn
  - `rd`: Thanh ghi đích
  - `shamt`: Số bit dịch (shift amount)
  - `funct`: Mã chức năng
  - Ví dụ: `add $t0, $s1, $s2`
  - Mã máy: `000000 10001 10010 01000 00000 100000`
- **I-format** (Immediate format):
  - Cấu trúc: `op (6 bits) | rs (5 bits) | rt (5 bits) | immediate (16 bits)`
  - `immediate`: Hằng số 16-bit hoặc địa chỉ dịch
  - Ví dụ: `lw $t0, 32($s3)`
  - Mã máy: `100011 10011 01000 0000000000100000`
- **J-format** (Jump format):
  - Cấu trúc: `op (6 bits) | address (26 bits)`
  - Ví dụ: `j 40000`
  - Mã máy: `000010 00000000000001111000100000`

5. Các lệnh cơ bản trong MIPS

- **Lệnh số học và logic**:
  - `add $rd, $rs, $rt`:
    - Cộng nội dung của thanh ghi `$rs` và `$rt`, rồi lưu kết quả vào `$rd`.
    - **Ví dụ**: `add $t0, $t1, $t2` – Cộng giá trị của `$t1` và `$t2`, lưu vào `$t0`.
    - **Lưu ý**: Kết quả có thể tràn nếu vượt quá giới hạn của số nguyên 32-bit.
  - `sub $rd, $rs, $rt`:
    - Trừ nội dung của `$rt` từ `$rs`, lưu kết quả vào `$rd`.
    - **Ví dụ**: `sub $t0, $t1, $t2` – Trừ giá trị trong `$t2` từ giá trị trong `$t1` và lưu vào `$t0`.
    - **Lưu ý**: Kết quả có thể tràn nếu giá trị vượt quá phạm vi của số nguyên có dấu.
  - `lui $rt, imm`:
    - Tải một giá trị hằng số 16-bit vào các bit cao hơn của thanh ghi `$rt`. Các bit thấp hơn của thanh ghi sẽ được thiết lập thành 0.
    - **Ví dụ**: `lui $t0, 0x1234` – Tải giá trị 0x1234 vào các bit cao của `$t0`, kết quả sẽ là:

      ```
      $t0 = 0x12340000
      ```

    - Lệnh này thường được sử dụng để thiết lập địa chỉ trong thanh ghi khi kết hợp với các lệnh khác.
  - `mul $rd, $rs, $rt`:
    - Nhân nội dung của `$rs` và `$rt`, lưu kết quả vào `$rd`.
    - **Ví dụ**: `mul $t0, $t1, $t2` – Nhân giá trị trong `$t1` và `$t2`, lưu vào `$t0`.
  - `div $rs, $rt`:
    - Chia nội dung của `$rs` cho `$rt`. Kết quả được lưu trong thanh ghi `$lo` (thương) và `$hi` (dư).
    - **Ví dụ**: `div $t1, $t2` – Chia giá trị trong `$t1` cho `$t2`.
  - `and $rd, $rs, $rt`:
    - Thực hiện phép AND bitwise giữa `$rs` và `$rt`, lưu vào `$rd`.
    - **Ví dụ**: `and $t0, $t1, $t2` – Thực hiện AND giữa `$t1` và `$t2`, lưu vào `$t0`.
  - `or $rd, $rs, $rt`:
    - Thực hiện phép OR bitwise giữa `$rs` và `$rt`, lưu vào `$rd`.
    - **Ví dụ**: `or $t0, $t1, $t2` – Thực hiện OR giữa `$t1` và `$t2`, lưu vào `$t0`.
  - `xori $rt, $rs, imm`:
    - Thực hiện phép XOR bitwise giữa `$rs` và hằng số `imm`, lưu vào `$rt`.
    - **Ví dụ**: `xori $t0, $t1, 5` – Thực hiện XOR giữa giá trị trong `$t1` và 5, lưu vào `$t0`.
  - `slt $rd, $rs, $rt`:
    - So sánh `$rs` và `$rt`. Nếu `$rs` < `$rt`, lưu 1 vào `$rd`, ngược lại lưu 0.
    - **Ví dụ**: `slt $t0, $t1, $t2` – Nếu `$t1` < `$t2`, thì `$t0` sẽ bằng 1, nếu không `$t0` sẽ bằng 0.
- **Lệnh bộ nhớ**:
  - `lw $rt, offset($rs)`:
    - Tải một từ (word) từ địa chỉ bộ nhớ vào thanh ghi `$rt`.
    - **Ví dụ**: `lw $t0, 32($t1)` – Tải dữ liệu từ bộ nhớ tại địa chỉ = giá trị trong `$t1` + 32 vào `$t0`.
  - `sw $rt, offset($rs)`:
    - Lưu một từ (word) từ thanh ghi `$rt` vào địa chỉ bộ nhớ.
    - **Ví dụ**: `sw $t0, 32($t1)` – Lưu giá trị trong `$t0` vào bộ nhớ tại địa chỉ = giá trị trong `$t1` + 32.
  - `lb $rt, offset($rs)`:
    - Tải một byte từ địa chỉ bộ nhớ vào thanh ghi `$rt`.
    - **Ví dụ**: `lb $t0, 1($t1)` – Tải byte tại địa chỉ = giá trị trong `$t1` + 1 vào `$t0`.
  - `sb $rt, offset($rs)`:
    - Lưu một byte từ thanh ghi `$rt` vào địa chỉ bộ nhớ.
    - **Ví dụ**: `sb $t0, 1($t1)` – Lưu byte trong `$t0` vào bộ nhớ tại địa chỉ = giá trị trong `$t1` + 1.
- **Lệnh điều khiển**:
  - `beq $rs, $rt, Label`:
    - Nhảy đến Label nếu `$rs` bằng `$rt`.
    - **Ví dụ**: `beq $t0, $t1, loop` – Nhảy đến `loop` nếu `$t0` = `$t1`.
  - `bne $rs, $rt, Label`:
    - Nhảy đến Label nếu `$rs` không bằng `$rt`.
    - **Ví dụ**: `bne $t0, $t1, loop` – Nhảy đến `loop` nếu `$t0` `$t1`.
  - `j Label`:
    - Nhảy không điều kiện đến Label.
    - **Ví dụ**: `j loop` – Nhảy đến `loop` mà không cần điều kiện.
  - `jal Label`:
    - Nhảy đến Label và lưu địa chỉ quay lại trong `$ra`.
    - **Ví dụ**: `jal function` – Nhảy đến hàm `function` và lưu địa chỉ trở lại vào `$ra`.
  - `jr $ra`:
    - Quay lại địa chỉ đã lưu trong `$ra`.
    - **Ví dụ**: `jr $ra` – Trở về từ hàm được gọi trước đó.

6. Kiểu dữ liệu trong MIPS

- **Số nguyên** (Integer):
  - 32-bit (4 bytes)
  - Phạm vi có dấu: -2,147,483,648 đến 2,147,483,647
  - Phạm vi không dấu: 0 đến 4,294,967,295
- **Byte**: 8-bit
- **Halfword**: 16-bit (2 bytes)
- **Word**: 32-bit (4 bytes)
- **Float**: 32-bit, IEEE 754 single precision

- **Double**: 64-bit, IEEE 754 double precision

7. Chế độ địa chỉ (Addressing Modes)

- **Immediate Addressing**:
  - Giá trị hằng số trong lệnh
  - Ví dụ: `addi $t0, $t1, 10`
- **Register Addressing**:
  - Dữ liệu trong thanh ghi
  - Ví dụ: `add $t0, $t1, $t2`
- **Base Addressing**:
  - Địa chỉ = Thanh ghi cơ sở + Offset
  - Ví dụ: `lw $t0, 4($t1)`
- **PC-Relative Addressing**:
  - Địa chỉ tương đối từ PC
  - Ví dụ: `beq $t0, $t1, Label`
- **Pseudodirect Addressing**:
  - Cho lệnh nhảy xa
  - Ví dụ: `j 40000`

8. Địa chỉ hóa bộ nhớ

- MIPS sử dụng địa chỉ hóa byte
- Dữ liệu truy cập theo từ (4 byte)
- **Big Endian**:
  - Byte có giá trị lớn nhất ở địa chỉ thấp nhất
  - Ví dụ: 0x12345678 lưu trữ như 12 34 56 78
- **Little Endian**:
  - Byte có giá trị nhỏ nhất ở địa chỉ thấp nhất
  - Ví dụ: 0x12345678 lưu trữ như 78 56 34 12
- MIPS thường sử dụng Big Endian

. Tính toán địa chỉ nhánh

- Trong MIPS, địa chỉ nhánh được tính toán dựa trên giá trị hiện tại của thanh ghi PC (Program Counter).
- Các lệnh nhánh được chia thành hai loại: có điều kiện và không có điều kiện.
- **Nhánh có điều kiện**:
  - Lệnh nhảy có điều kiện sẽ kiểm tra một điều kiện và chỉ nhảy nếu điều kiện đó đúng.
  - Công thức tính địa chỉ nhánh:

$$\text{Địa chỉ nhảy} = \text{PC} + 4 + (\text{offset} \times 4)$$

  - **Offset** là số lượng lệnh giữa lệnh nhảy và lệnh mục tiêu.
  - **Ví dụ**:
    - Lệnh:

```
beq $t0, $t1, target
...
target: ...
```

    - Nếu `$t0` bằng `$t1`, nhảy đến địa chỉ `target`.
    - Nếu lệnh nhảy là thứ 5 trong mã, `offset` sẽ là 2 (nhảy đến lệnh mục tiêu cách 2 lệnh).
    - Tính địa chỉ:

$$\text{Địa chỉ nhảy} = \text{PC} + 4 + (2 \times 4)$$

- **Nhánh không điều kiện**:
  - Lệnh nhảy không điều kiện sẽ nhảy đến một địa chỉ mà không cần kiểm tra điều kiện.
  - Công thức tính địa chỉ nhảy cũng tương tự như trên:

$$\text{Địa chỉ nhảy} = \text{PC} + 4 + (\text{offset} \times 4)$$

  - **Ví dụ**:
    - Lệnh:

```
j target
```

    - Nhảy đến địa chỉ `target` mà không kiểm tra điều kiện.
    - Nếu `target` là lệnh thứ 10 trong mã, `offset` sẽ là 5.
    - Tính địa chỉ:

$$\text{Địa chỉ nhảy} = \text{PC} + 4 + (5 \times 4)$$

- **Điều cần lưu ý**:
  - Trong MIPS, `PC` sẽ luôn trỏ đến lệnh tiếp theo, vì vậy giá trị `PC` cần được điều chỉnh khi tính địa chỉ nhảy.
  - Các lệnh nhánh sử dụng địa chỉ tương đối, do đó, việc tính toán chính xác `offset` là rất quan trọng để đảm bảo nhảy đến đúng lệnh mục tiêu.
  - Các nhánh có điều kiện sẽ không nhảy nếu điều kiện không đúng, trong khi các nhánh không điều kiện sẽ luôn nhảy đến địa chỉ được chỉ định.

10. So sánh RISC và CISC

- **RISC (Reduced Instruction Set Computer)**:
  - Sử dụng một tập lệnh nhỏ hơn
  - Lệnh có kích thước cố định
  - Thực hiện lệnh trong một chu kỳ đồng hồ
  - Tối ưu hóa cho phần cứng đơn giản
  - Ví dụ: MIPS, ARM
- **CISC (Complex Instruction Set Computer)**:
  - Sử dụng một tập lệnh lớn hơn
  - Lệnh có kích thước biến đổi
  - Thực hiện lệnh trong nhiều chu kỳ đồng hồ
  - Tối ưu hóa cho các lệnh phức tạp
  - Ví dụ: x86, Intel
- **Điểm khác biệt chính**:
  - RISC: Ít lệnh hơn nhưng dễ tối ưu hóa, thường hiệu quả hơn trong thực thi.
  - CISC: Nhiều lệnh phức tạp hơn nhưng có thể giảm mã chương trình.

11. Mở rộng dấu (Sign Extension)

- **Mở rộng dấu** là quá trình mở rộng một số nguyên có dấu từ kích thước nhỏ hơn lên kích thước lớn hơn mà không làm mất thông tin.
- MIPS sử dụng **bù 2 (Two's Complement)** để biểu diễn các số nguyên có dấu. Khi chuyển đổi một số nguyên từ 16-bit lên 32-bit, cần đảm bảo rằng giá trị số nguyên không thay đổi.
- **Quy tắc mở rộng dấu**:
  - Nếu bit dấu (bit cao nhất) của số nguyên có dấu là 0 (số dương), thêm các bit 0 vào phía trước để mở rộng lên kích thước lớn hơn.
  - Nếu bit dấu là 1 (số âm), thêm các bit 1 vào phía trước.
- **Ví dụ mở rộng dấu**:
  - Số nguyên 16-bit: `1111111111111010` (biểu diễn số -6 trong bù 2)
  - Mở rộng lên 32-bit:

    `11111111111111111111111111111010`

  - Số này trong 32-bit vẫn là -6.
- **Sử dụng trong lệnh MIPS**:
  - Khi sử dụng lệnh với hằng số 16-bit (ví dụ: `addi`), MIPS tự động thực hiện mở rộng dấu cho số hằng 16-bit thành 32-bit.
  - Ví dụ:
    - Lệnh: `addi $t0, $t1, -10`
    - Giá trị -10 có dạng 16-bit: `1111111111110110`
    - Sau khi mở rộng dấu, nó trở thành: `11111111111111111111111111110110` (32-bit)
- **Mở rộng dấu trong lệnh chuyển đổi**:
  - MIPS không có lệnh riêng cho mở rộng dấu; thay vào đó, điều này được thực hiện trong quá trình lấy lệnh.
  - Khi một lệnh cần tham chiếu đến một số nguyên có dấu, như trong lệnh tải với offset hoặc trong các phép toán, phần mở rộng dấu là cần thiết để đảm bảo tính chính xác của phép toán.
- **Lệnh có chữ "u" ở cuối**:
  - Các lệnh có chữ "u" ở cuối thường được sử dụng để chỉ rằng các phép toán này hoạt động trên các số nguyên không dấu và có thể liên quan đến việc mở rộng dấu bằng cách thêm bit 0 vào phía trước.
  - **Ví dụ về các lệnh có chữ "u"**:
    - `addiu`:
      - Cộng giá trị trong thanh ghi với một hằng số (có thể âm), nhưng không làm tràn.
      - Ví dụ: `addiu $t0, $t1, 10` – Cộng 10 vào giá trị trong `$t1` và lưu vào `$t0`.
    - `lbu`:
      - Tải một byte từ bộ nhớ và mở rộng dấu 0 vào thanh ghi.
      - Ví dụ: `lbu $t0, 0($t1)` – Tải byte tại địa chỉ trong `$t1` vào `$t0` và mở rộng dấu 0 cho các bit cao hơn.
    - `andi`:
      - Thực hiện phép AND bitwise giữa thanh ghi và một hằng số, mở rộng dấu 0 cho hằng số.
      - Ví dụ: `andi $t0, $t1, 0xFF` – AND giá trị trong `$t1` với 255, lưu vào `$t0`.