Bộ chia

1. **Các lệnh chia trong RISC-V (RV32M)**

| **Lệnh** | **Toán hạng** | **Kết quả** | **Ý nghĩa** |
| --- | --- | --- | --- |
| DIV | signed ÷ signed | quotient | Thương có dấu |
| DIVU | unsigned ÷ unsigned | quotient | Thương không dấu |
| REM | signed ÷ signed | remainder | Dư có dấu |
| REMU | unsigned ÷ unsigned | remainder | Dư không dấu |

DIV 7 ÷ 3 = 2

REM 7 ÷ 3 = 1

DIV -7 ÷ 3 = -2

REM -7 ÷ 3 = -1

**Khái niệm cơ bản trong phép chia**

* Dividend (số bị chia)
* Divisor (số chia)
* Quotient (thương)
* Remainder (số dư)

Quan hệ:

*Dividend = Quotient x Diviosr + Remainder*

1. **Ý tưởng của “Thuật toán chia tiểu học”**
   1. **Ý tưởng cơ bản**

Phần cứng bắt chước cách ta chia số “bằng tay”:

1. **So sánh** divisor với phần còn lại của dividend (remainder).
2. Nếu remainder ≥ divisor → trừ divisor, đặt bit thương = 1.
3. Nếu remainder < divisor → giữ nguyên remainder, đặt bit thương = 0.
4. Dịch divisor sang phải (align dần theo dividend).
5. Lặp lại cho đến hết số bit.

Trong nhị phân, mọi quyết định chỉ có 2 khả năng: “chia được 1 lần” hoặc “chia được 0 lần”.

* 1. **Kiến trúc mạch chia cơ bản**

Một mạch chia cơ bản (32-bit dividend ÷ 32-bit divisor) gồm các khối:

* **Remainder register (64-bit)**: ban đầu chứa dividend (ở nửa thấp), nửa cao = 0.
* **Divisor register (32-bit)**: chứa divisor, có thể dịch trái/phải để align.
* **Quotient register (32-bit)**: khởi tạo = 0, cập nhật dần từng bit.
* **ALU (Subtract + Compare)**: dùng để thử trừ divisor khỏi remainder.
* **Control FSM**: điều khiển lặp qua 32 chu kỳ.

**Một vòng lặp:**

1. Dịch **Remainder** sang trái (chuẩn bị tạo bit thương mới).
2. Trừ divisor khỏi phần trên của remainder.
3. Nếu kết quả ≥ 0 → ghi bit thương = 1, giữ remainder mới.  
   Nếu kết quả < 0 → khôi phục remainder cũ (restore), ghi bit thương = 0.
4. Lặp lại cho đến khi đủ số bit quotient.
   * Đây gọi là **Restoring Division Algorithm** vì khi remainder âm ta phải “restore” lại.
   1. **Tối ưu phần cứng**

**Non-Restoring Division**: thay vì phục hồi remainder ngay, ta cho phép remainder âm, và quyết định trừ/cộng divisor ở bước kế tiếp → ít mạch hơn, nhanh hơn.

**Improved datapath (Hennessy & Patterson Figure 3.11)**:

* + - Chỉ cần ALU 32-bit, divisor 32-bit.
    - Remainder giữ 64-bit nhưng dịch trái thay vì dịch divisor sang phải.
    - Quotient gộp vào nửa thấp của Remainder (tiết kiệm thanh ghi).
  1. **Xử lý dấu**

Bước đầu: lấy trị tuyệt đối của dividend và divisor.

Chia như **unsigned** bằng thuật toán trên.

Sau khi có quotient và remainder:

* Quotient nhận dấu dựa vào dividend ⊕ divisor.
* Remainder mang dấu giống dividend.

1. **Ví dụ**

***0. Khởi tạo***

* Dividend (số bị chia) = **1101₂ = 13**
* Divisor (số chia) = **0011₂ = 3**
* **Remainder (dư)**: khởi tạo bằng dividend nhưng đặt trong thanh ghi 2n bit (ở đây 8 bit).
* R = 0000 1101
* **Quotient (thương)**: khởi tạo = 0.
* Q = 0000

***1. Iteration 1***

* **Dịch trái Remainder** (đưa bit cao của dividend lên):
* R = 0001 1010
* **Thử trừ divisor (3 = 0011) khỏi phần trên của R**:
  + Lấy 4 bit cao của R (0001 = 1).
  + 1 – 3 = âm → không được.
  + Restore (giữ nguyên).
* Đặt bit thương (Q) = 0.
* Q = 0000

***2. Iteration 2***

* **Dịch trái R**:
* R = 0011 0100
* Lấy 4 bit cao (0011 = 3).
* Thử trừ 3: 3 – 3 = 0 (≥ 0, hợp lệ).



* Cập nhật R = 0000 0100.



* Đặt bit thương = 1.
* Q = 0001

***3. Iteration 3***

* **Dịch trái R**:
* R = 0000 1000
* 4 bit cao = 0000 = 0.
* 0 – 3 = âm → restore.
* Bit thương = 0.
* Q = 0010

***4. Iteration 4***

* **Dịch trái R**:
* R = 0001 0000
* 4 bit cao = 0001 = 1.
* 1 – 3 = âm → restore.
* Bit thương = 0.
* Q = 0100

***5. Kết quả***

* Sau 4 vòng lặp (tương ứng 4 bit quotient):
* Quotient = 0100₂ = 4
* Remainder = 0001₂ = 1
* Đúng: **13 ÷ 3 = 4 dư 1** ✅