Trường Đại Học Bách Khoa Hà Nội
Khoa Điện tử - Viễn thông
Môn: Kiến trúc máy tính


# Báo cáo bài tập lớn

## Tìm hiểu kiến trúc tập lệnh RISC-V

Sinh viên thực hiện:

| | | |
|---|---|---|
| 1. Phạm Văn Lâm | 20111746 | ĐTTT02-K56 |
| 2. Lê Văn Mạnh | 20111832 | ĐTTT03-K56 |

Hà Nội,12/2014

# Mục Lục

# 1. Giới thiệu

+ RISC-V là một kiến trúc máy tính với kiến trúc tập lệnh mới dựa trên những thiết kế ban đầu là để phục vụ cho học tập và nghiên cứu

+ Tuy nhiên, ngày nay người ta mong muốn phát triển kiến trúc này thành một kiến trúc tiêu chuẩn mở để triển khai trong công nghiệp

+ Bài báo cáo này nhằm mục đích giúp ta thấy rõ được những vấn đề sau:

- RISC-V là một ISA mở hoàn toàn và sẵn dùng cho giáo dục và công nghiệp

- RISC-V là phù hợp với phần cứng máy tính chứ không chỉ là mô phỏng

- RISC-V tránh hiện tượng "over-architecting" phù hợp với những kiểu kiến trúc nhỏ, đặc thù (như microcoded(vi code)...) hoặc triển khai cho các công nghệ ASIC , FPGA...

- RISC-V chia ra làm các mức cơ sở rất nhỏ, do đó nó có thể sử dụng bởi chính nó nhằm phục vụ cho giáo dục, hoặc mở rộng ra để hỗ trợ phát triển về phần mềm

- RISC-V hỗ trợ số thực dấu phẩy động theo chuẩn IEEE-754

- RISC-V hỗ trợ địa chỉ 32 bit và 64 bit để có thể triển khai ứng dụng hay phát triển nhân hệ thống và triển khai phần cứng

- RISC-V hỗ trợ "multicore" và "manycore", bao gồm hệ thống đa xử lý không đồng nhất

- Tuỳ chọn lệnh với độ dài thay đổi để có thể mở rộng ra cho những lệnh yêu cầu độ dài lớn và phù hợp với những lệnh với độ dài nhỏ để có thể cải thiện hiệu suất, kích thước code và năng lượng tiêu thụ

- RISC-V dễ dàng để phát triển cho các mức cao hơn

## 1.1. Tổng quan về RISC-V ISA

+ RISC-V ISA được định nghĩa như là một ISA cơ sở mức số nguyên và có thể triển khai ở bất kỳ hệ thống máy tính nào

+ RISC-V rất giống với bộ xử lý RISC trước đây ngoại trừ nó việc nó không có trễ khi thực hiện lệnh nhảy và có thể tuỳ biến độ dài của chỉ thị lệnh

+ RISC-V cơ sở bị thu hẹp lại nhằm tối thiểu hoá kiến trúc tập lệnh nhằm phù hợp với các mục đích như compilers, assemblers, linhkers và các hoạt động của hệ thống

+ Mỗi tập lệnh số nguyên cơ bản được định rõ đặc điểm bởi độ dài thanh ghi số nguyên và kích thước của các địa chỉ do người dùng sử dụng

+ Có 2 mức số nguyên cơ bản là 32 bit(RV32I) và 64 bit (RV64I), việc triển khai hệ thống phải cung cấp một hoặc cả hai loại trên

+ RISC-V với phần mở rộng được chia làm 2 phần: tiêu chuẩn ("standard") và không tiêu chuẩn ("no standard"). Trong đó,"Standard" là phù hợp cho những yêu cầu chung, nhưng không nên để xung đột với các chuẩn khác. Còn "No Standard" sẽ phù hợp với những yêu cầu riêng biệt hơn, và nó có thể xung đột với các chuẩn khác

## 1.2. Mã hoá độ dài chỉ thị lệnh

+ RISC-V ISA được cố định độ dài chỉ thị lệnh là 32 bit và phải được căn chỉnh lề ở biên

+ Tuy nhiên RISC-V ISA tiêu chuẩn được thiết kế với những chỉ thị lệnh có thể thay đổi được

=> Vì vậy ta phải mã hoá để có thể xác định được độ dài của lệnh đang dùng

+ Những lệnh có độ dài 16 bit kết thúc bởi 2 bit "00" hoặc "01" hoặc "10"

+ Những lệnh có độ dài 32 bit được kết thúc bởi 2 bit "11"

+ Ngoài ra các lệnh có độ dài lớn hơn 32 bit sẽ được cho thêm một số lượng các bit "1" nhất định như hình sau:

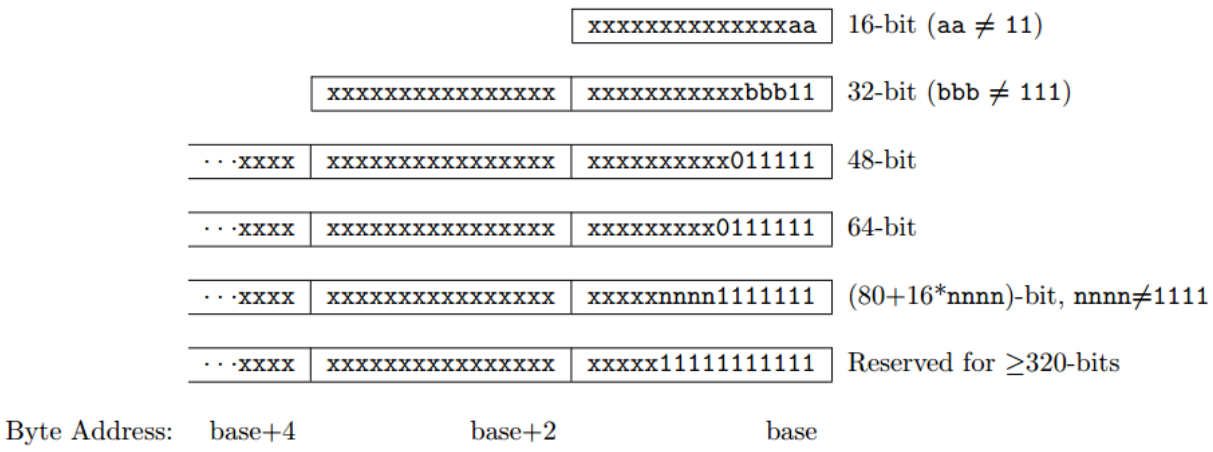



Figure 1.1: RISC-V instruction length encoding.

## 1.3. Exceptions ,Traps and Interrupts

(Ngoại lệ, ngắt chương trình và làm gián đoạn chương trình)

+ "Exceptions" được sử dụng để hướng tới những điều kiện không thường xảy ra trong hệ thống khi chạy chương trình

+ "Traps" được sử dụng để hướng tới việc chuyển quyền điều khiển một cách đồng bộ tới môi trường người giám sát với những ngoại lệ bị gây ra ở bên trong hệ thống

+ "Interrupts" được sử dụng để hướng tới việc chuyển quyền điều khiển một cách không đồng bộ tới môi trường người giám sát với những ngoại lệ bị gây ra ở bên ngoài hệ thống

# 2. RV32I

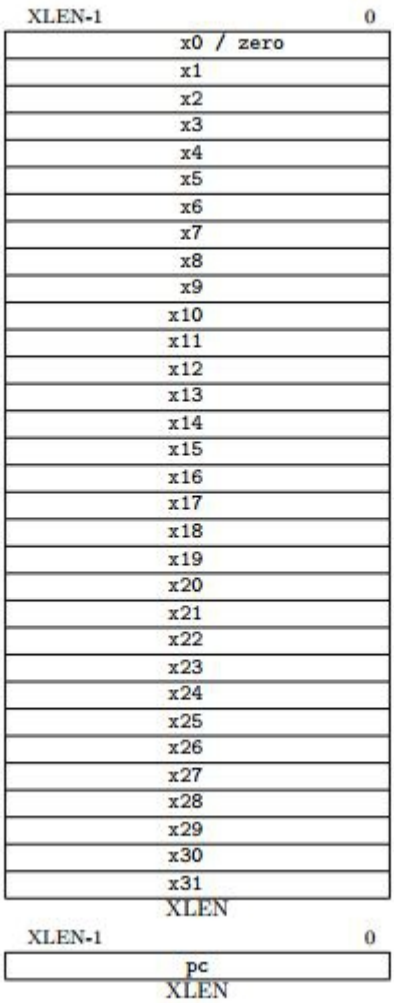
## 2.1. Cấu trúc

+ Có 32 thanh ghi với mục đính chung đó là các thanh ghi từ x1 tới x31 (giữ những giá trị nguyên). Riêng thanh ghi x0 được cố định sẵn là luôn mang giá trị 0 (x0-zero)

+ Với R32I, thanh ghi x bất kỳ có độ rộng 32 bit (với R64I là 64 bit)

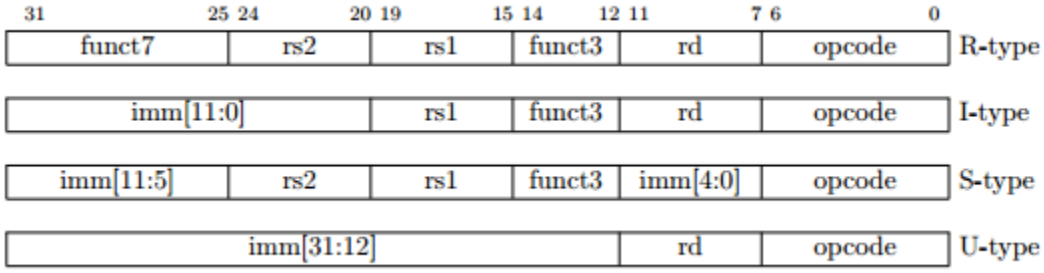
+ Người ta dùng XLEN để chỉ độ dài của thanh ghi x (có thể là 32 hay 64)

+ Ngoài ra còn có một thanh ghi nữa là thanh ghi PC (program counter) dùng để lưu địa chỉ của chỉ thị lệnh hiện tại



## 2.2. Những định dạng lệnh cơ sở

+ Có 4 lệnh cơ sở là lệnh R, I, S, U như hình sau:



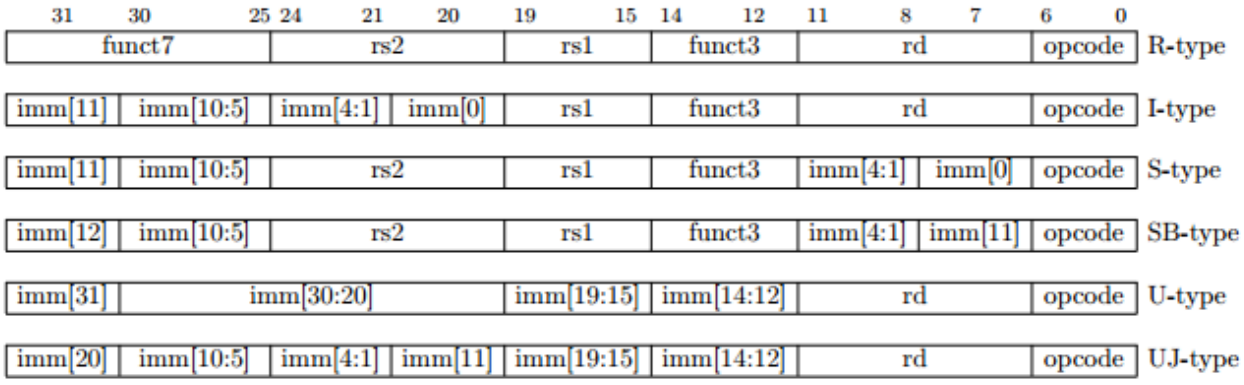

+ Trong đó ta thấy vị trí của nguồn (rs1 và rs2) và thanh ghi đích (rd) là giống nhau trong cả 4 loại lệnh

+ Thành phần "immediate" nếu có thì luôn luôn ở phía trước bên trái,và được sắp xếp vị trí sao cho giảm thiểu được độ phức tạp của phần cứng

+ Bít dấu luôn luôn là bit 31, nhằm tăng tốc độ xử lý trong mạch

## 2.3. Những biến thể của lệnh S,U

+ Có thêm 2 dạng biến thể của lệnh S và U là SB và UJ



## 2.4. Những chỉ thị lệnh tính toán số nguyên

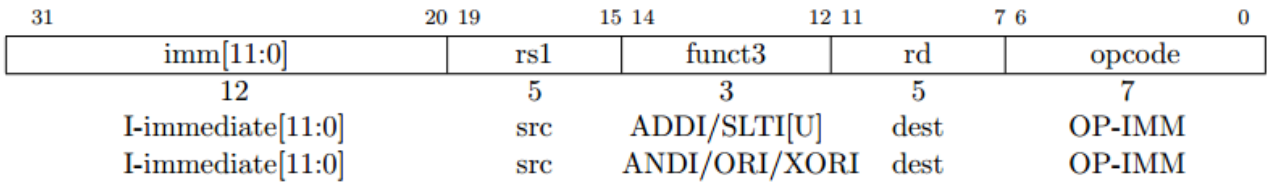
+ Có 2 loại lệnh có thể được sử dụng để tính toán số nguyên đó là lệnh I và lệnh R

+ Đối với cả hai loại thì kết quả đều có thể lưu vào thanh ghi rd

### 2.4.1. Những chỉ thị lệnh giữa thanh ghi và immediate

#### 2.4.1.1. Lệnh loại I

* Cấu trúc câu lệnh là:



| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| imm[11:0] | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |
| I-immediate[11:0] | src | ADDI/SLTI[U] | dest | OP-IMM |
| I-immediate[11:0] | src | ANDI/ORI/XORI | dest | OP-IMM |

+ Lệnh ADDI : dùng để cộng số ở imm[11:0] (đã được mở rộng thành 32 bit có dấu) và số trong rs1. Việc xác định tràn bị từ chối, và kết quả đơn giản chỉ có 32 bit lưu ở thanh ghi rd. Đặc biệt, lệnh ADDI rd,rs1,0 dùng để gán rd = rs1

*addi rd,rs1,imm[11:0]*

+ Lệnh SLTI: dùng để so sánh kém với số có dấu. Tức là rd = 1 khi rs1 < imm[11:0] (imm[11:0] đã mở rộng) ngược lại rd = 0 . SLTIU tương tự ,nhưng chỉ áp dụng cho số không dấu

*slti rd,rs1,imm[11:0]*

*sltiu rd,rs1,imm[11:0]*

*Đặc biệt: sltiu rd,rs1,1 => cho kết quả là rd = 1 khi rs1 =0 và rd = 0 khi rs1 > 0*

+ Lệnh ANDI, ORI, XORI: là những toán tử logic thực hiện các phép toán AND, OR, XOR theo từng bit đối với rs1 và imm[11:0] và kết quả sẽ lưu vào rd.

Đặc biệt XORI rd,rs1,-1 sẽ thực hiện việc đảo tất cả các bit của rs1

* Ngoài ra còn một số lệnh dịch có cấu trúc như sau:

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|
| imm[11:5] | imm[4:0] | rs1 | funct3 | rd | opcode |
| 7 | 5 | 5 | 3 | 5 | 7 |
| 0000000 | shamt[4:0] | src | SLLI | dest | OP-IMM |
| 0000000 | shamt[4:0] | src | SRLI | dest | OP-IMM |
| 0100000 | shamt[4:0] | src | SRAI | dest | OP-IMM |

+ Những lệnh dịch bởi hằng số là một dạng đặc biệt của lệnh I. Với toán hạng được dịch lưu tại rs1, số lượng dịch lưu tại 5 bit thấp của trường immediate.

+ Lệnh SLLI: là lệnh dịch trái logic, bit 0 được dịch vào trường bit thấp từ bên phải sang bên trái

+ Lệnh SRLI: là lệnh dịch phải logic, bit 0 được dịch vào trường bit cao, từ bên trái sang bên phải

+ Lệnh SRAI: là lệnh dịch phải số học, các bit được dịch từ trái sang phải, riêng bit MSB được giữ lại

* Đặc biệt: Lệnh NOP:

+ Cấu trúc là:

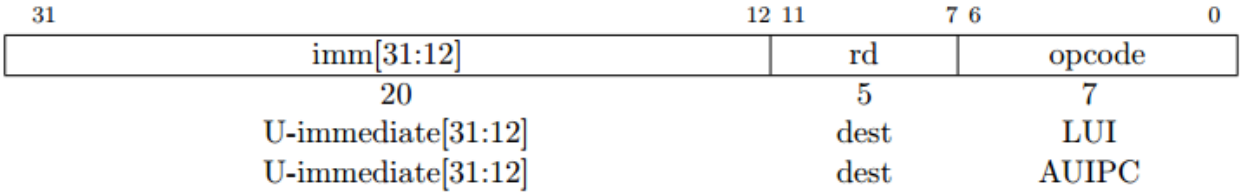
| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| imm[11:0] | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |
| 0 | 0 | ADDI | 0 | OP-IMM |

+ Lệnh NOP không làm thay đổi trạng thái hiện tại vì nó thực hiện:

*ADDI x0,x0,0*

### 2.4.1.2. Lệnh loại U

+ Cấu trúc lệnh là:

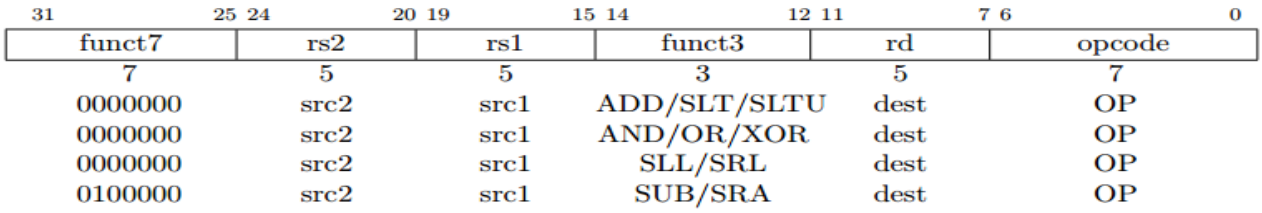



+ Lệnh LUI (load upper immediate): Dùng để tạo hằng số 32 bit, lệnh này sẽ copy giá trị imm[31:12] vào 20 bit đầu tiên của thanh ghi rd, và điền giá trị 0 vào 12 bit còn lại của rd

+ Lệnh AUIPC (add upper immediate to pc): Dùng để xây dựng địa chỉ tương đối cho pc, lệnh này sẽ tạo ra độ lệch 32 bit từ 20 bit của imm[31:12] và điền giá trị 0 vào 12 bit thấp còn lại, sau đó sẽ cộng với giá trị pc và kết quả đạt được sẽ lưu vào rd

### 2.4.2. Những chỉ thị lệnh giữa thanh ghi với thanh ghi

+ Đó là những lệnh loại R




+ Tất cả các lệnh đều lấy toán hạng ở rs1, rs2 và kết quả lưu vào rd

+ funct7 và funct3 dùng để lựa chọn kiểu toán tử sẽ được sử dụng

+ Lệnh ADD và SUB thực hiện lệnh cộng hoặc trừ(rs1 với rs2), và không xác định trạng thái tràn bit. Kết quả lưu vào rd

+ Lệnh SLT và SLTU thực hiện việc so sánh có dấu hay không có dấu.

Nếu rs1 < rs2 thì rd = 1 ngược lại rd = 0; Đặc biệt lệnh SLTU rd,x0,rs2 sẽ cho kết quả rd = 1 nếu rs2 không bằng 0, ngược lại sẽ cho giá trị rd = 0;

+ Lệnh AND, OR, XOR là những toán tử logic sẽ thực hiện AND, OR, XOR với rs1 và rs2 sau đó kết quả lưu vào rd

+ Lệnh SLL, SLR, SRA thực hiện phép dịch trái logic, dịch phải logic và dịch phải số học đối với giá trị trên thanh ghi rs1 và số lượng dịch là 5 bit thấp của thanh ghi rs2

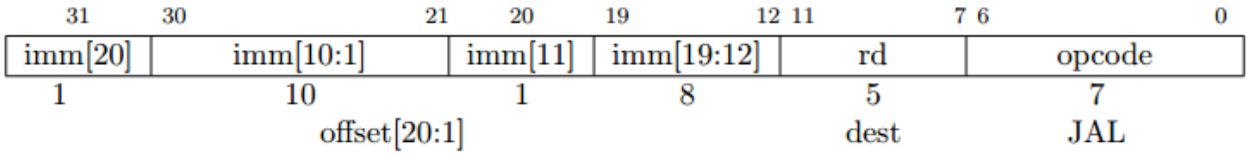
## 2.5. Chỉ thị lệnh thực hiện việc nhảy

+ RV32I cung cấp 2 chỉ thị cho lệnh nhảy là : nhảy không điều kiện và lệnh nhảy có điều kiện.

+ Các lệnh nhảy trong RV32I là hoàn toàn không có khoảng trễ.

### 2.5.1. Lệnh nhảy không có điều kiện

* Chỉ thị lệnh nhảy và liên kết (JAL: jump and link): sử dụng lệnh kiểu UJ.

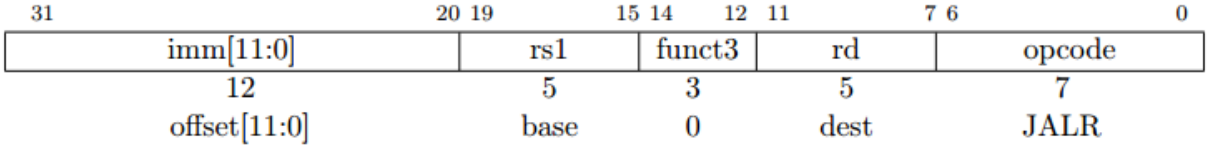



+ Thành phần offset được mở rộng bit và cộng vào PC để xác định địa chỉ đích nhảy tới

+ JAL sẽ lưu địa chỉ của chỉ thị lệnh ngay sau lệnh nhảy (pc+4) vào thanh ghi rd (thường sử dụng rd = x1)

+ Trường hợp nhảy mà không liên kết thì nó giống với JAL với rd = x0

* Lệnh nhảy gián tiếp (JALR: jump and link register): sử dụng định dạng lệnh I



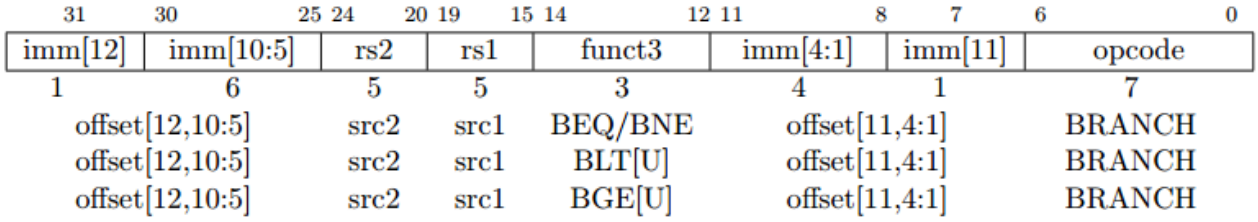
| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| imm[11:0] | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |
| offset[11:0] | base | 0 | dest | JALR |

+ Địa chỉ đích thu được bởi việc cộng 12 bit ở I-immediate với thanh ghi rs1 , sau đó, những bit ít ý nghĩa sẽ cho bằng 0.

+ Địa chỉ của lệnh sau lệnh nhảy (pc+4) sẽ được lưu vào thanh ghi rd, thanh ghi x0 có thể sử dụng làm thanh ghi đích nếu kết quả không yêu cầu.

### 2.5.2. Lệnh nhảy có điều kiện

+ Tất cả lệnh nhảy có điều kiện đều sử dụng lệnh dạng SB và 12 bit B-immediate sẽ được cộng với PC để xác định địa chỉ đích sẽ được nhảy tới



| 31 | 30 25 | 24 20 | 19 15 | 14 12 | 11 8 | 7 | 6 0 |
|---|---|---|---|---|---|---|---|
| imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode |
| 1 | 6 | 5 | 5 | 3 | 4 | 1 | 7 |
| offset[12,10:5] | | src2 | src1 | BEQ/BNE | offset[11,4:1] | | BRANCH |
| offset[12,10:5] | | src2 | src1 | BLT[U] | offset[11,4:1] | | BRANCH |
| offset[12,10:5] | | src2 | src1 | BGE[U] | offset[11,4:1] | | BRANCH |

+ Các lệnh nhảy này sẽ thực hiện việc so sánh giá trị ở 2 thanh ghi rs1 và rs2

+ Lệnh BEQ sẽ thực hiện lệnh nhảy nếu rs1 = rs2

+ Lệnh BNE sẽ thực hiện lệnh nhảy nếu rs1 != rs2

+ BLT và BLTU sẽ nhảy nếu rs1 < rs2 ,lần lượt áp dụng với số có dấu và không dấu

+ BGE và BGEU sẽ nhảy nếu rs1>=rs2, áp dụng với số có dấu và không dấu

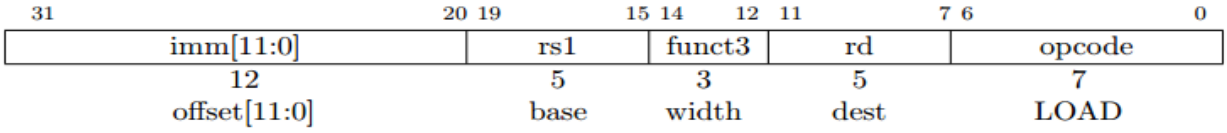
+ Ngoài ra còn có các loại BGT, BGTU, GLE, BLEU cũng tương tự

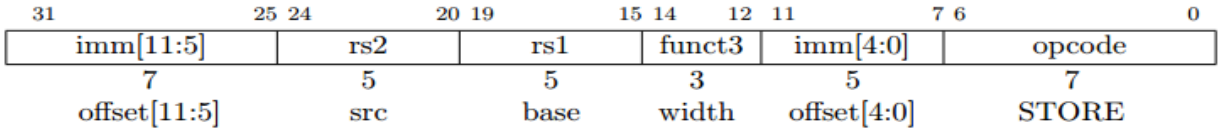
## 2.6. Lệnh tải và lệnh lưu trữ

+ Lệnh tải (load) và lệnh lưu (store) là những lệnh có thể truy cập vào bộ nhớ

+ RV32I cung cấp cho người dùng địa chỉ 32 bit, để định nghĩa địa chỉ ô nhớ sẽ được truy cập , đó là địa chỉ byte theo kiểu little-endian

+ Cấu trúc lệnh:




+ Lệnh load và store sẽ dịch chuyển dữ liệu giữa thanh ghi và bộ nhớ

+ Lệnh load có định dạng I, còn lệnh store có định dạng S

+ Địa chỉ byte trong bộ nhớ mà được sử dụng sẽ thu được bằng cách cộng giá trị trong thanh ghi rs1 với offset

+ Lệnh Load sẽ tải giá trị trong bộ nhớ vào thanh ghi rd

+ Lệnh Store sẽ sẽ copy giá trị trong thanh ghi rs2 vào bộ nhớ

* Cụ thể:

+ Lệnh LW sẽ tải giá trị 32 bit từ bộ nhớ vào rd

+ Lệnh LH sẽ tải giá trị 16 bit từ bộ nhớ , sau khi mở rộng dấu thành 32 bit sẽ lưu vào rd

+ Lệnh LHU sẽ tải giá trị 16 bit từ bộ nhớ, sau đó mở rộng thêm bit 0 thành 32 bit sẽ lưu vào rd

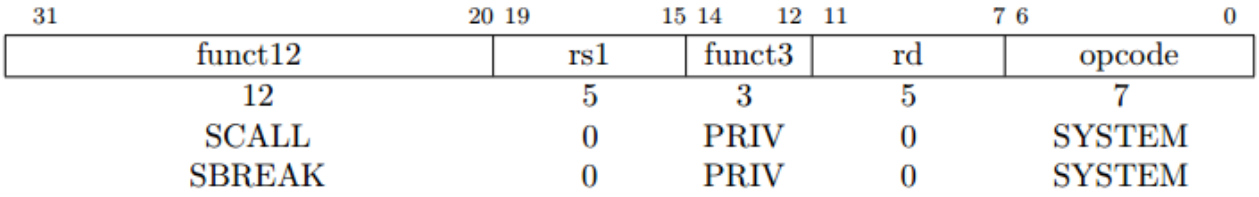
+ Lệnh LB và LBU tương tự như LH và LHU nhưng áp dụng với 8 bit

+ Lệnh SW, SH, SB sẽ lưu giá trị 32 bit, 16 bit và 8 bit thấp từ thanh ghi rs2 vào bộ nhớ

## 2.7. Những chỉ thị lệnh với hệ thống

+ Là những chỉ thị lệnh được sử dụng để truy cập vào hệ thống được sử dụng lệnh dạng I

* Scall và Sbreak:



| 31 — 20 | 19 — 15 | 14 — 12 | 11 — 7 | 6 — 0 |
|---|---|---|---|---|
| funct12 | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |
| SCALL | 0 | PRIV | 0 | SYSTEM |
| SBREAK | 0 | PRIV | 0 | SYSTEM |

+ Lệnh SCALL: được sử dụng để tạo ra một yêu cầu với môi trường hoạt động của hệ thống, khi đó ta sẽ định nghĩa những tham số nhất định cho hệ thống, thông thường đó sẽ là vị trí định nghĩa trong tệp thanh ghi

+ Lệnh SBREAK: được sử dụng bởi người gỡ lỗi, khi đó sẽ có yêu cầu chuyển dữ liệu ngược lại tới môi trường gỡ lỗi

* Timers và counters:

| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| csr | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |
| RDCYCLE[H] | 0 | CSRRS | dest | SYSTEM |
| RDTIME[H] | 0 | CSRRS | dest | SYSTEM |
| RDINSTRET[H] | 0 | CSRRS | dest | SYSTEM |

+ Lệnh RDCYCLE: ghi vào thanh ghi rd số lượng của chu kỳ đồng hồ được thực thi bởi bộ xử lý tại một luồng khác được bắt đầu trước đó

+ RDTIME là lệnh điều khiển việc đếm thời gian trong hệ thống và lưu vào thanh ghi rd giá trị là một số 32 bit

+ RDINSTRET lệnh điều khiển việc ghi số lần nghỉ của hệ thống vào thanh ghi rd, giá trị là một số 32 bit

# 3. RV64I

+ RV64I được phát triển dựa trên RV32I. Vì vậy ta chủ yếu nói về những điểm khác của RV64I với RV32I

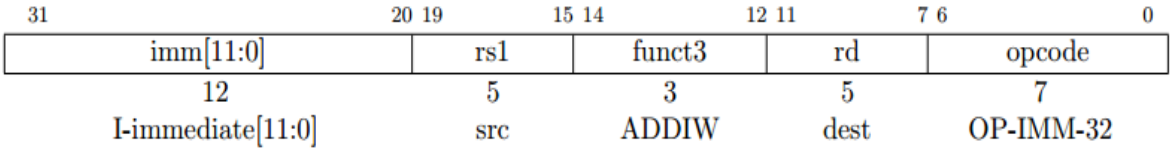
## 3.1. Trạng thái của thanh ghi

+ RV64I mở rộng các thanh ghi số nguyên và hỗ trợ người sử dụng không gian địa chỉ 64 bit ( XLEN = 64)

## 3.2. Những chỉ thị lệnh tính toán số nguyên

### 3.2.1. Những chỉ thị lệnh giữa thanh ghi và immediate

* Lệnh cộng:



| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| imm[11:0] | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |
| I-immediate[11:0] | src | ADDIW | dest | OP-IMM-32 |

+ ADDIW là lệnh chỉ có trong RV64I, lệnh này cộng phần mở rộng dấu 12 bit ở phần immediate vào thanh ghi rs1 và kết quả trả về được lưu vào thanh ghi rd. Overflows được bỏ qua và kết quả được lưu vào 32 bit thấp và được mở rộng dấu 64 bit.

* Lệnh dịch:

| 31 26 | 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|---|
| imm[11:6] | imm[5] | imm[4:0] | rs1 | funct3 | rd | opcode |
| 6 | 1 | 5 | 5 | 3 | 5 | 7 |
| 000000 | shamt[5] | shamt[4:0] | src | SLLI | dest | OP-IMM |
| 000000 | shamt[5] | shamt[4:0] | src | SRLI | dest | OP-IMM |
| 010000 | shamt[5] | shamt[4:0] | src | SRAI | dest | OP-IMM |
| 000000 | 0 | shamt[4:0] | src | SLLIW | dest | OP-IMM-32 |
| 000000 | 0 | shamt[4:0] | src | SRLIW | dest | OP-IMM-32 |
| 010000 | 0 | shamt[4:0] | src | SRAIW | dest | OP-IMM-32 |

+ Lệnh dịch là lệnh loại I, và có cấu trúc giống với RV32I, toán hạng được dịch lưu ở rs1, số lượng dịch lưu ở 6 bit thấp của I-immediate và kết quả lưu vào rd

+ Lệnh SLLI là lệnh dịch trái logic, số 0 được điền thêm vào các bit thấp

+ Lệnh SRLI là lệnh dịch phải logic, số 0 được điền vào các bít cao

+ Lệnh SRAI là lệnh dịch phải số học , bít đầu tiên được sao chép vào các bit còn trống

+ Lệnh RV32I, SLLI, SRLI, và SRAI không hợp lệ khi imm[5] khác 0

+ Lệnh SLLIW, SRLIW và SRAIW chỉ có trong RV64I được định nghĩa tương tự nhưng thực hiện trên giá trị 32 bit và kết quả trả về cũng là 32 bit

* Lệnh loại U:

+ Cấu trúc:

| 31 — 12 | 11 — 7 | 6 — 0 |
|---|---|---|
| imm[31:12] | rd | opcode |
| 20 | 5 | 7 |
| U-immediate[31:12] | dest | LUI |
| U-immediate[31:12] | dest | AUIPC |

+ LUI tương tự như với RV32I , nó đặt 20 bit U-immediate vào các bit 31-12 của thanh ghi rd và đặt các bit 0 vào các vị trí còn lại. 32 bit kết quả sẽ được mở rộng thành 64 bit

+ AUIPC (add upper immediate to pc) sử dụng opcode giống RV32I, đươc sử dụng để xây dựng địa chỉ PC . AUIPC mở rộng 20 bit U-immediate thành 64 bit rồi cộng nó với PC và kết quả được lưu trong thanh ghi rd

### 3.3.2. Những chỉ thị lệnh giữa thanh ghi với thanh ghi

+ Cấu trúc:

| 31 — 25 | 24 — 20 | 19 — 15 | 14 — 12 | 11 — 7 | 6 — 0 |
|---|---|---|---|---|---|
| funct7 | rs2 | rs1 | funct3 | rd | opcode |
| 7 | 5 | 5 | 3 | 5 | 7 |
| 0000000 | src2 | src1 | SLL/SRL | dest | OP |
| 0100000 | src2 | src1 | SRA | dest | OP |
| 0000000 | src2 | src1 | ADDW | dest | OP-32 |
| 0000000 | src2 | src1 | SLLW/SRLW | dest | OP-32 |
| 0100000 | src2 | src1 | SUBW/SRAW | dest | OP-32 |

+ ADDW và SUBW là lệnh trong RV64I được định nghĩa tương tự như ADD và SUB nhưng thực hiện trên các giá trị 32 bit và tạo ra kết quả 32 bit. Bít tràn được bỏ qua và 32 bít được mở rộng dấu sau đó cho vào thanh ghi 64 bit

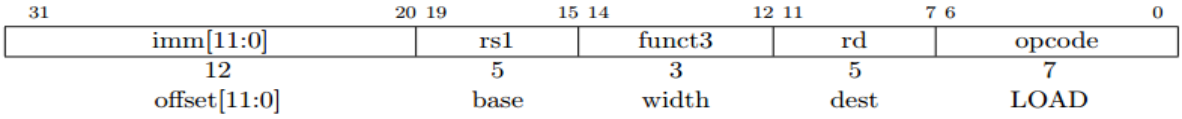
+ SLL, SRL và SRA  là các lệnh dịch trái logic, dịch phải logic và dịch phải số học với thanh ghi rs1 với số lần dịch được lưu trong thanh ghi rs2. Trong RV64I chỉ có 6 bit thấp của thanh ghi rs2 là chứa giá trị được quan tâm

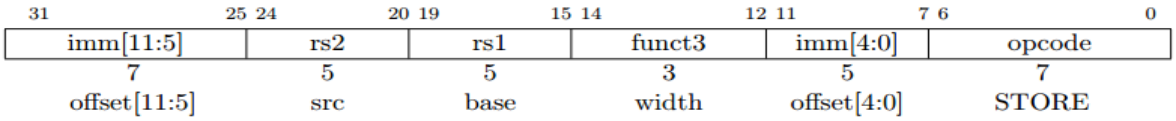
+ SLLW, SRLW và SRAW là các lệnh chỉ có trong RV64I được đinh nghĩa tương tự như ba lệnh trên nhưng toán tử và kết quả đầu ra là 32 bit. Số lần dịch được gán bởi rs2[4:0]

## 3.3. Chỉ thị lệnh load và store

+ RV64I mở rộng không gian địa chỉ thành 64 bit . Môi trường thực thi sẽ xác định những phần của không gian địa chỉ là hợp lệ để truy cập

+ Cấu trúc:




+ Lệnh LD là lệnh tải dữ liệu 64 bit từ bộ nhớ vào thanh ghi rd chỉ có trong RV64I

+ Lệnh LW là lênh tải  biến 32 bit từ bộ nhớ và sau đó mở rộng dấu thành 64 trước khi lưu vào thanh ghi rd,

+ Lệnh LWU cũng tương tự như LW nhưng nó được mở rộng bằng cách đưa thêm bit 0

+ Lệnh LH, LHU cũng tương tự nhưng áp dụng cho giá trị 16 bit

+ Lệnh LB, LBU cũng tương tự và áp dụng cho giá trị 8 bit

+ Các lệnh SD, SW, SWU, SH, SHU, SB, SBU cũng tương tự như các lệnh trên

nhưng nó được dùng để lưu giá trị từ thanh ghi rs2 vào bộ nhớ

## 3.4. Những chỉ thị lệnh với hệ thống

+ Cấu trúc:

| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| csr | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |
| RDCYCLE | 0 | CSRRS | dest | SYSTEM |
| RDTIME | 0 | CSRRS | dest | SYSTEM |
| RDINSTRET | 0 | CSRRS | dest | SYSTEM |

+ RDCYCLE là lệnh điều khiển đếm số xung thực hiện bởi một bộ xử lý mà mạch phần cứng thực hiện từ một thời điểm bất kì trong quá khứ, kết quả được lưu vào trong thanh ghi rd, kết quả là giá trị 64 bit mà không bao giờ tràn.

+ RDTIME là lệnh điều khiển việc đếm thời gian trong hệ thống và lưu vào thanh ghi rd giá trị là một số 64 bit

+ RDINSTRET lệnh điều khiển việc ghi số lần nghỉ của hệ thống vào thanh ghi rd, giá trị là một số 64 bit

# 4. Dạng mở rộng chuẩn "M":Nhân và chia với số nguyên

## 4.1. Phép nhân

+ Cấu trúc :

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|
| funct7 | rs2 | rs1 | funct3 | rd | opcode |
| 7 | 5 | 5 | 3 | 5 | 7 |
| MULDIV | multiplier | multiplicand | MUL/MULH[[S]U] | dest | OP |
| MULDIV | multiplier | multiplicand | MULW | dest | OP-32 |

+ Lệnh MUL: thực hiện việc nhân XLEN-bit x XLEN-bit và kết quả lưu vào XLEN-bit thấp tại thanh ghi đích

+ Lệnh MULH, MULHU và MULHSU cũng thực hiện phép nhân nhưng kết quả lưu vào những bit cao, áp dụng lần lượt cho phép nhân (số có dấu và số có dấu), (số không dấu và số không dấu), (số có dấu và số không dấu).

+ Nếu kết quả yêu cầu cả bit cao và bit thấp thì phải thực hiện liên tiếp 2 dòng code sau:

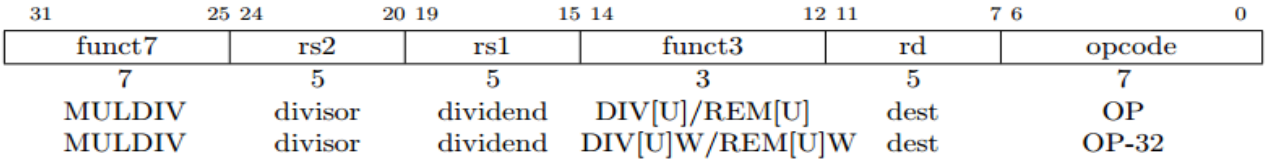
*MULH[[S]U]* *rdh, rs1, rs2*

*MUL* *rdl, rs1, rs2*

+ Lệnh MULW chỉ áp dụng với RV64.

## 4.2. Phép chia

+ Cấu trúc:




+ Lệnh DIV chia số nguyên có dấu XLEN bit cho số nguyên có dấu XLEN bit

+ Lệnh DIVU cũng tương tự như DIV nhưng chỉ áp dụng cho số nguyên không dấu

+ Lệnh REM và lệnh REMU cung cấp số dư của phép chia

+ Nếu thương và số dư được yêu cầu thì phải thực hiện liên tiếp 2 lệnh sau:

*DIV[U]* *rdq, rs1, rs2*

*REM[U]* *rdr, rs1, rs2*

+ Lệnh DIVW ,lệnh DIVUW, lệnh REM, lệnh REMU cũng tương tự nhưng chỉ áp dụng với RV64

+ Trường hợp chia cho 0 và trường hợp tràn,  kết quả sẽ cho trong bảng sau:

| Trường hợp | Số bị chia (Dividend) | Số chia (Divisor) | DIVU | REMU | DIV | REM |
|---|---|---|---|---|---|---|
| Chia cho 0 | x | 0 | 2^(XLEN-1) | x | -1 | x |
| Tràn (chỉ áp dụng với số có dấu) | -2^(XLEN-1) | -1 | don't care | don't care | -2^(XLEN-1) | 0 |

# 5. Các dạng mở rộng khác

## 5.1. Dạng mở rộng chuẩn "A"

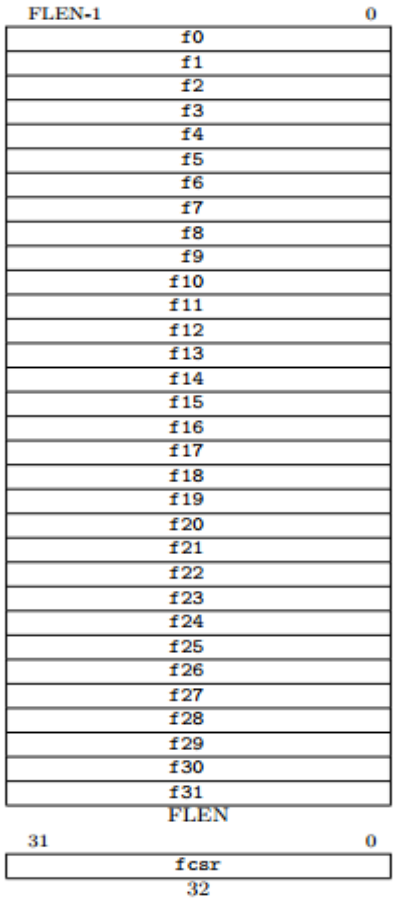
+ Bao gồm các lệnh dùng để phân chia việc đọc và ghi trên bộ nhớ, để hỗ trợ việc đồng bộ giữa nhiều luồng cùng chạy trên cùng một đơn vị bộ nhớ

+ Để thực hiện được điều đó thì mỗi lệnh sẽ đưa thêm vào 2 bit là *aq* và *rl*

## 5.2. Dạng mở rộng chuẩn "F"

+ Bổ sung thêm các lệnh để áp dụng cho số thực dấu phẩy động độ chính xác đơn

+ Dạng này bổ sung thêm 32 thanh ghi số thực dấu phẩy động, từ f0-f31, mỗi thanh ghi 32 bit và một thanh ghi trạng thái và điều khiển fcsr (bao gồm các chế độ hoạt động, các trạng thái ngoại lệ)




+ Trong trường hợp này: người ta dùng FLEN để chỉ độ dài thanh ghi. Đối với số thực dấu phẩy động độ chính xác đơn FLEN = 32

+ Khi đó, các hoạt động đều dựa trên các giá trị 32 bit là số thực dấu phẩy động độ chính xác đơn

## 5.3. Dạng mở rộng chuẩn "D"

+ Bổ sung các chỉ thị lệnh áp dụng cho số thực dấu phẩy động độ chính xác kép

+ Dạng này được xây dựng dựa trên dạng mở rộng dạng "F"

+ Các thanh ghi trong dạng này là f0-f31 nhưng mỗi thanh ghi có 64 bit

(FLEN = 64)

## 5.4. Dạng mở rộng chuẩn "Q"

+ Bổ sung các lệnh để áp dụng cho các số thực dấu phẩy động có độ dài 128 bit

+ FLEN = 128

## 5.5. Dạng mở rộng chuẩn "L"

+ Áp dụng cho số thực dấu phẩy động dạng decimal

(Decimal Floating-Point)

## 5.6. Dạng mở rộng chuẩn "C"

+ Dạng mở rộng cung cấp các chỉ thị lệnh mà nó đã được nén lại

+ Qua đó giúp giảm thiểu kích thước code

+ Dạng này cho phép thêm các chỉ thị lệnh 16 bit để mã hoá cho các hoạt động chung cho các số nguyên

+ Các chỉ thị lệnh rút gọn có thể được thêm vào ở cả RV32I và RV64I

## 5.7. Dạng mở rộng chuẩn "B"

+ Bổ sung thêm các chỉ thị lệnh tương tác lên các bit,

+ Bao gồm các chỉ thị lệnh để chèn, kiểm tra trường bit, phép quay,...

## 5.8. Dạng mở rộng chuẩn "T"

+ Bổ sung thêm các chỉ thị lệnh để tương tác với bộ nhớ

# 6. So sánh, đánh giá

+ Ta thấy RISC-V đã mở rộng cung cấp định nghĩa về RISC-V một cách cơ bản dựa trên số nguyên (với RV32I và RV64I). Từ đó, có rất nhiều dạng mở rộng khác để có thể áp dụng cho số thực dấu phẩy động độ chính xác đơn, đôi,...

+ RISC-V rất phù hợp để có thể áp dụng trong học tập, nghiên cứu cũng như cho nền công nghiệp

+ RISC-V có khả năng phát triển cao hơn.