

# Relatório do Trabalho de ISA e Microarquitetura

## 1. Especificação da ISA

### 1.1 Introdução

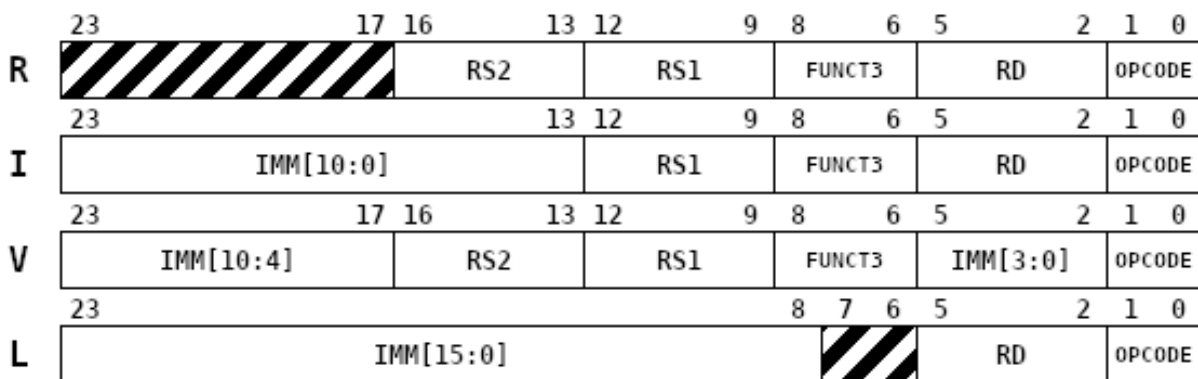
O conjunto de instruções desenvolvido neste trabalho foi fortemente inspirado nas ISAs RV32I e RV32M da arquitetura RISC-V, adaptadas para instruções de 24 bits. Por causa da limitação de tamanho de 24 bits para a instrução, apenas as operações mais essenciais foram incluídas. Ao todo, o conjunto de instruções contém 23 operações.

O objetivo foi manter a arquitetura o mais simples possível, sem comprometer a praticidade de uso. O conjunto de instruções definido cobre operações aritméticas, lógicas, controle de fluxo e acesso à memória, proporcionando suporte completo às estruturas básicas de programação e chamadas de função.

### 1.2 Formatos de instrução

A ISA definida possui quatro formatos de instrução, identificados unicamente pelo opcode:

- **Formato R:** Utilizado para operações lógicas e aritméticas entre registradores. Contém dois registradores fonte (*rs1* e *rs2*) e um registrador destino (*rd*).
- **Formato I:** Usado para operações que envolvem um registrador fonte (*rs1*), um valor imediato (*imm*) e um registrador destino (*rd*). Além de operações lógicas e aritméticas, é usado para operações de salto e acesso à memória.
- **Formato V:** Usado para operações que envolvem dois registradores fonte (*rs1*, *rs2*) e um valor imediato (*imm*), como instruções de *branch* e *store*.
- **Formato L:** Exclusivo para a operação *li* (*load immediate*), usada para carregar um valor imediato de 16 bits (*imm*) em algum registrador (*rd*).



Todos os formatos incluem 2 bits de opcode, suficientes para determinar o tipo da instrução. Além disso, os formatos R, I e V possuem 3 bits adicionais (*funct3*), permitindo 8 instruções únicas para cada formato. O formato L não utiliza *funct3*, liberando espaço para um imediato maior.

Os formatos R e L possuem bits excedentes que poderiam ser usados para expandir o conjunto de instruções, mas optou-se por mantê-los inutilizados para simplificar a implementação.

### 1.3 Registradores

O banco de registradores inclui 16 registradores de propósito geral, cada um com 32 bits, organizados conforme a seguinte convenção de uso:

- *r0* (*zero*): constante zero, imutável
- *r1* (*rad*): endereço de retorno

- `r2 (rbp)`: ponteiro para a base da pilha (base pointer)
- `r3 (rsp)`: ponteiro para o topo da pilha (stack pointer)
- `r4..r7 (rt0..rt3)`: registradores temporários
- `r8..r11 (rs0..rs3)`: registradores salvos
- `r12 (ra0)`: valor de retorno / primeiro argumento de função
- `r13..r15 (ra1..ra3)`: argumentos de função

Há também um registrador especial de 8 bits reservado para o controle do fluxo de execução: o registrador `ip` (instruction pointer). Ele é incrementado automaticamente após a execução de cada instrução, exceto em casos de desvios e saltos, onde o valor é alterado diretamente.

## 1.4 Conjunto de instruções

### 1.4.1 Instruções lógicas e aritméticas

Formato R:

- `add`: soma.
- `sub`: subtração.
- `mul`: multiplicação.
- `div`: divisão inteira.
- `mod`: resto da divisão inteira.
- `and`, `or`, `xor`: operações lógicas.

Formato I:

- `addi`: soma com valor imediato.
- `modi`: resto da divisão inteira com valor imediato.
- `slli`: *shift* lógico para a esquerda com valor imediato.
- `and`, `or`, `xor`: operações lógicas com valor imediato.

### 1.4.2 Instruções de acesso à memória

Formato I:

- `lw`: Carrega uma palavra de 32 bits da memória para o registrador.

Formato V:

- `sw`: Armazena uma palavra de 32 bits na memória a partir de um registrador.

As operações de acesso à memória utilizam a notação '`op rx, imm(ry)`', onde `ry` é o registrador que contém o endereço base e `imm` é um valor de deslocamento.

### 1.4.3 Instruções de salto / desvio de fluxo

Formato I:

- `jalr`: Salto incondicional para o endereço em `rs1` deslocado de `imm`; `rd` recebe o endereço de retorno.

Formato V:

- `beq`, `bne`: Desvio condicional para igualdade e diferença. Se a condição for verdadeira, salta para `ip + imm`.
- `bge`, `blt`: Desvio condicional para maior ou igual e menor.
- `halt`: para a execução do programa, impedindo `ip` de ser incrementado.

É importante notar que as instruções de *branch* fazem um desvio relativo à posição atual no programa, enquanto as instruções de *jump* fazem um desvio absoluto, sem considerar a posição atual no programa.

## 1.4.4 Instruções de carregamento

Formato L:

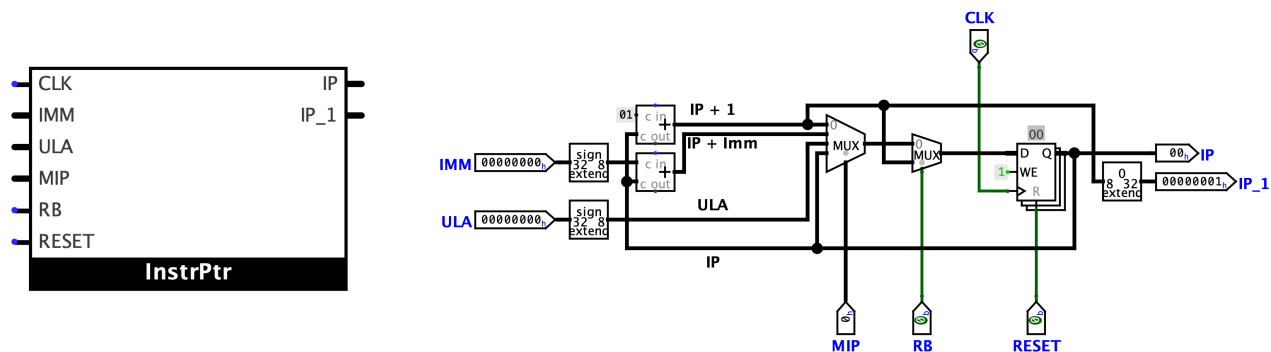
- li: Carrega um imediato de 16 bits (imm) na parte baixa do registrador rd.

## 2. Implementação da ISA

A implementação da ISA proposta foi feita por módulos, que no fim foram conectados em um circuito principal para obter toda a funcionalidade proposta.

### 2.1 InstrPtr

Este circuito controla o registrador ip (instruction pointer), de 8 bits. Com base nas entradas, decide qual será o valor de ip no próximo ciclo de clock.



Entradas:

- CLK (1b): Entrada para o clock;
- IMM (32b): O valor do imediato para a instrução atual;
- ULA (32b): O resultado da operação calculada pela unidade de lógica e aritmética;
- MIP (2b): Valor de seleção calculado pela unidade de controle para o multiplexador que escolhe entre ip + 1, ip + imm, ULA e ip;
- RB (1b): Valor de seleção para o segundo multiplexador, calculado pelo circuito de branch;
- RESET (1b): Caso RESET seja 1, o valor de ip é zerado.

Saídas:

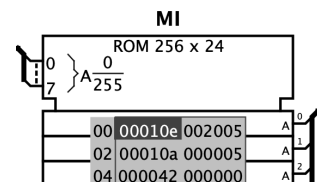
- IP (8b): O valor atual do registrador ip;
- IP\_1 (32b): O valor atual do registrador ip incrementado em 1.

MIP e RB são usados em conjunto para decidir se a instrução atual irá incrementar o registrador ip ou se irá fazer um desvio/salto. A saída IP\_1 é estendida para 32 bits, pois é usada pela instrução jalr (que guarda o endereço de retorno no registrador de destino, de 32 bits).

### 2.2 MI (Memória de Instruções)

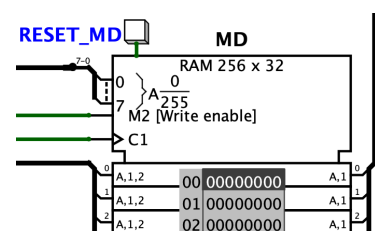
Uma memória ROM com 256 endereços de 24 bits.

Cada endereço guarda uma instrução.



### 2.3 MD (Memória de Dados)

Uma memória RAM com 256 endereços de 32 bits.



## 2.4 Imm

Um circuito que calcula o valor do imediato com base na instrução atual, considerando os formatos definidos. O resultado é o valor do imediato estendido para 32 bits.



## 2.5 BancoReg

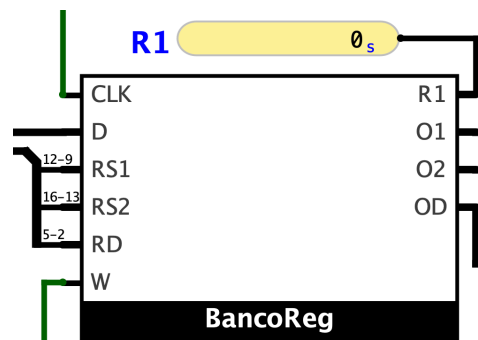
Um banco de registradores com 16 registradores de 32 bits.

Entradas:

- CLK (1b): Entrada para o clock;
- D (32b): Valor de 32 bits que para ser escrito em RD;
- RS1 (4b): Código do registrador  $r s 1$ ;
- RS2 (4b): Código do registrador  $r s 2$ ;
- RD (4b): Código do registrador  $r d$ ;
- W (1b): *write enable* (1 habilita a escrita, 0 desabilita).

Saídas:

- R1 (32b): O valor atual do registrador  $r 1$ ;
- O1 (32b): O valor atual do registrador  $r s 1$ ;
- O2 (32b): O valor atual do registrador  $r s 2$ ;
- OD (32b): O valor atual do registrador  $r d$ .

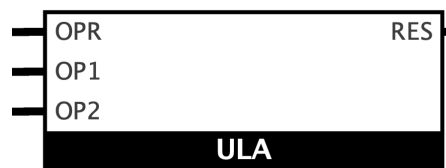


A saída R1 foi implementada com o único propósito de facilitar o debugging, pois permite visualizar o valor de um registrador sem ter que executar instruções de acesso à memória.

A saída OD foi implementada para possibilitar que a operação *li* (load immediate) altere a parte baixa [15:0] do registrador  $r d$  sem alterar sua parte alta [31:16].

## 2.6 ULA (Unidade de Lógica e Aritmética)

Um circuito que, dados um código de operação e dois operandos de 32 bits, calcula o resultado da operação lógica ou aritmética representada por OPR.



Entradas:

- OPR (4b): Código de operação;
- OP1 (32b): Primeiro operando;
- OP2 (32b): Segundo operando.

Saídas:

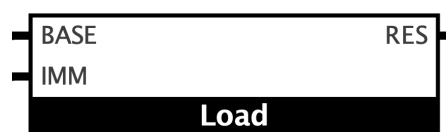
- RES (32b): O resultado da operação.

Existem operações com mais de um código associado. Isso foi feito para facilitar a implementação da unidade de controle. Como havia a necessidade de implementar 9 operações na ULA, seriam necessários 4 bits para representar uma operação. Os códigos de operação foram alocados de modo a coincidirem com o valor formado pelo primeiro bit do opcode seguido do *funct3*.

OPR	EF	OPR	EF
0000	add	1000	add
0001	sub	1001	add
0010	mul	1010	add
0011	div	1011	sll
0100	mod	1100	mod
0101	and	1101	and
0110	or	1110	or
0111	xor	1111	xor

## 2.7 Load

Um circuito usado exclusivamente pela instrução *li* (load immediate). A saída RES é o valor de 32 bits formado por BASE[31:16] e IMM[15:0].



## 2.8 Branch

Circuito usado pelas instruções de branch. Dados um código de operação e dois operandos de 32 bits, calcula o valor para a entrada RB do circuito InstrPtr.

