Instruções de Máquina (ISA Instruction Set Architecture) Uma introdução a RISC-V

MAC0344 - Arquitetura de Computadores
Prof. Siang Wun Song
Slides usados: https://www.ime.usp.br/∼song/mac344/slides07a-isa.pdf

Baseado parcialmente em SiFive -

An Introduction to RISC-V Architecture

Instruções de Máquina ou Instruction Set Architeture

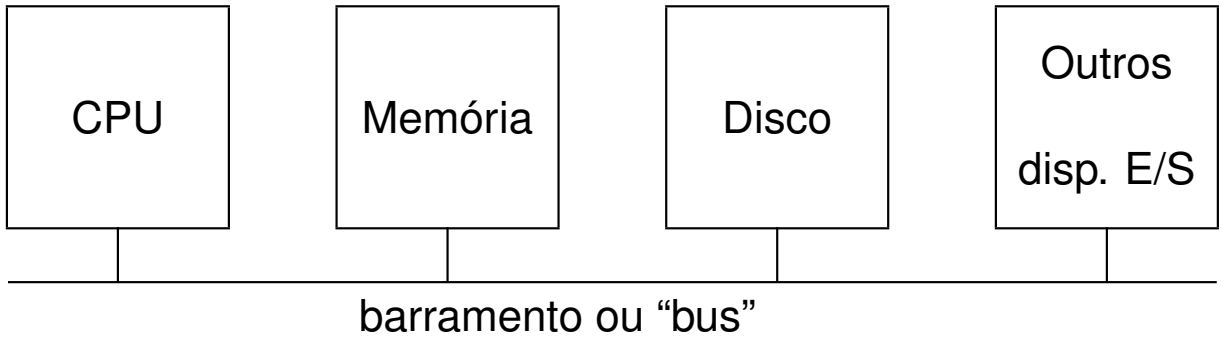
- A operação do processador é determiniada pelas instruções que ele executa, conhecidas como instruções de máquina.
- A coleção de instruções que o processador executa forma o conjunto de instruções (ISA Instruction Set Architecture).
- O número de instruções varia de processador para processador. Um computador com a arquitetura CISC pode conter centenas de instruções algumas muito complexas. Enquanto que um computador com a arquitetura RISC pode conter um número reduzido de instruções simples.
  Veremos:
  - Espaço de endereçamento.
  - Formato de instruções.
  - Códigos de operação (opcodes) que expandem.

Projeto do Formato de Instruções

Na arquitetura de von Neumann o programa está na memória e cada instrução é executada no processador.

O processador (datapath) pode operar dados de um certo número de bits, digamos 64 bits. O barramento (bus), pode transportar em paralelo, um certo número de bits, digamos 64 bits. Nesse caso, diremos que o processador é de 64 bits.

O formato de instruções portanto é importante, uma vez que cada instrução deve ser levada da memória para o processador, para ali ser executada.



Espaço de endereçamento

Um endereço absoluto de m bits pode endereçar 2^m posições de memória.

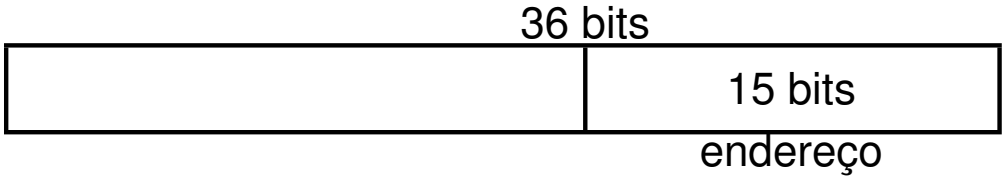
Há várias maneiras de obter tal endereço absoluto.

- Contido na própria instrução
- Uso de registradores base
- Uso de registradores de segmento
- Uso de registradores de banco
- Uso de modos de endereçamento

Contido na própria instrução

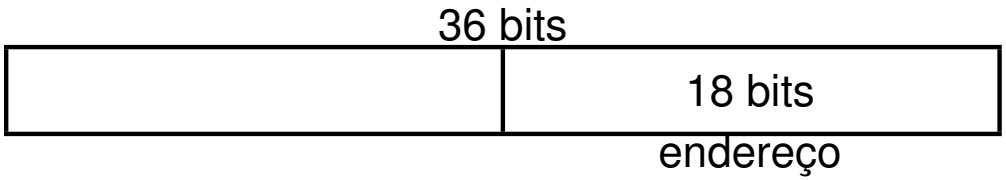
A instrução contém espaço suficiente para incluir todos os m bits do endereço absoluto.

Exemplo - IBM 7094:



2^{15} = 32 K palavras

Exemplo - DEC PDP-11:



2^{18} = 256 K palavras

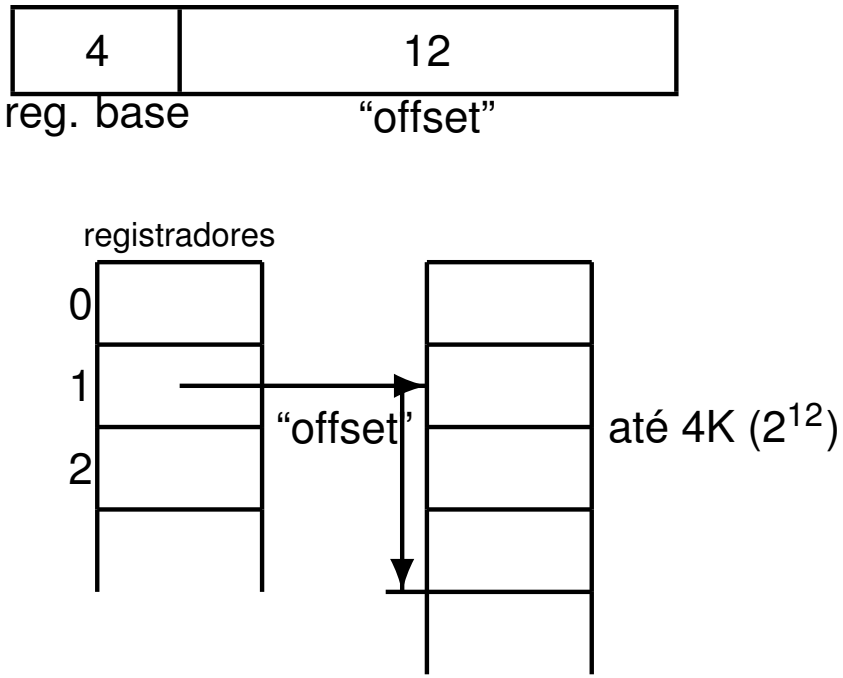
Esse esquema simples apresenta alguns problemas:

- Requer instruções de muitos bits
- Espaço de endereçamento limitado

Uso de registradores base

O endereço é obtido somando-se o conteúdo de um registrador base e um deslocamento (“offset”).

Exemplo: IBM 360 e 370 possuem 16 registradores, com 24 bits de cada registrador dedicados para endereçamento.



Uso de registradores de segmento

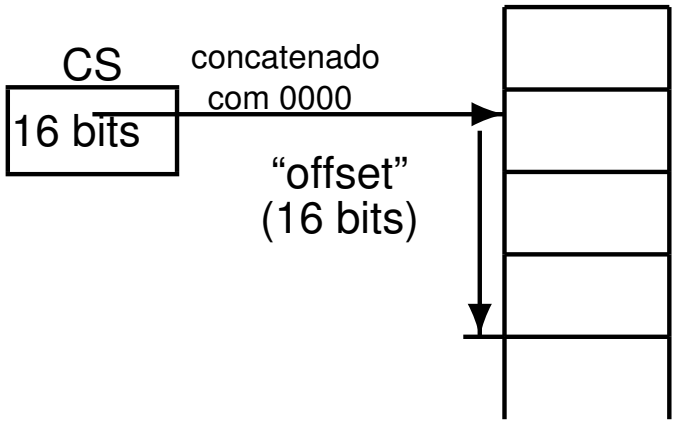
Registradores de segmento apontam para determinados segmentos (programa, dados, pilha, etc.). Um deslocamento ("offset") determina a posição dentro do segmento.

Exemplo - Intel 8088 e 8086 apresentam 4 registradores de segmento chamados

- CS code segment (programa)
- DS data segment (dados)
- SS stack segment (stack ou pilha)
- ES extra segment (outros dados)

Registradores de Segmento Intel 8088/8086

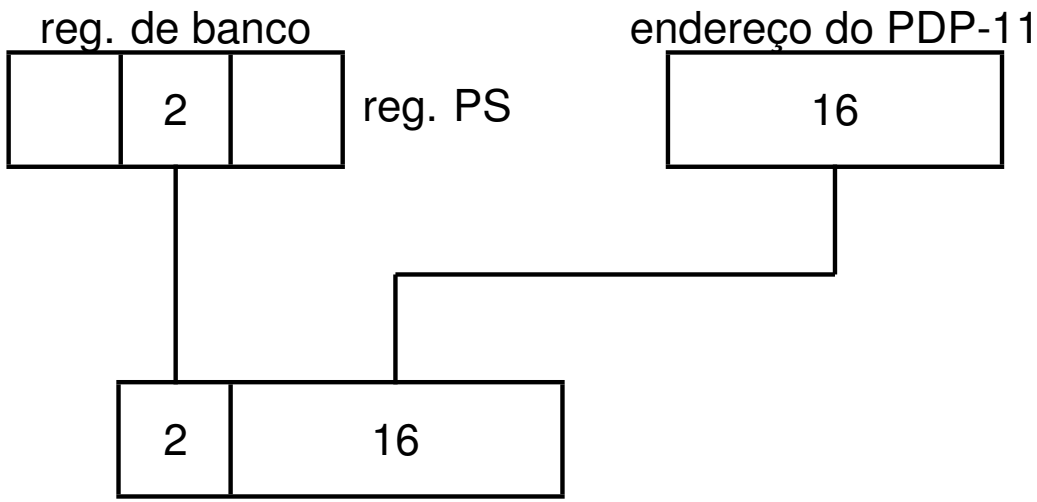
O Intel 8088 ou 8086 usa endereço absoluto de 20 bits, dando 2^{20} ou 1M posições endereçáveis.



O valor do registrador de segmento é primeiro concatenado com 4 bits iguais a 0000. Depois somado ao valor do offset dá o endereço final de 20 bits.

Uso de registradores de banco

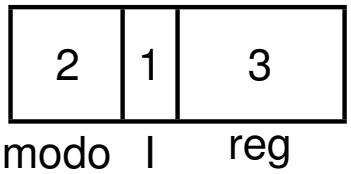
O endereço é o conteúdo do registrador de banco *concatenado* com o campo de offset.



Uso de modos de endereçamento

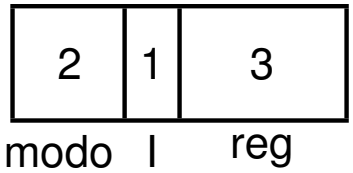
Introduzido pelo PDP-11, esse esqueme permite uma grande economia no tamanho da instrução.
PDP-11: instrução de apenas 16 bits para gerar 2 endereços de operandos.

Cada operando é constituído de 3 campos.



Obs: I = endereçamento indireto

Modos de endereçamento



- modo 00 - registrador: reg contém o operando
- modo 01 - auto-incr: reg é incrementado após uso como endereço do operando
- modo 10 - auto-decr: reg é decrementado e usado como endereço do operando
- modo 11 - indexado: conteúdo da palavra seguinte à instrução é somado com conteúdo do registrador reg para dar o endereço do operando

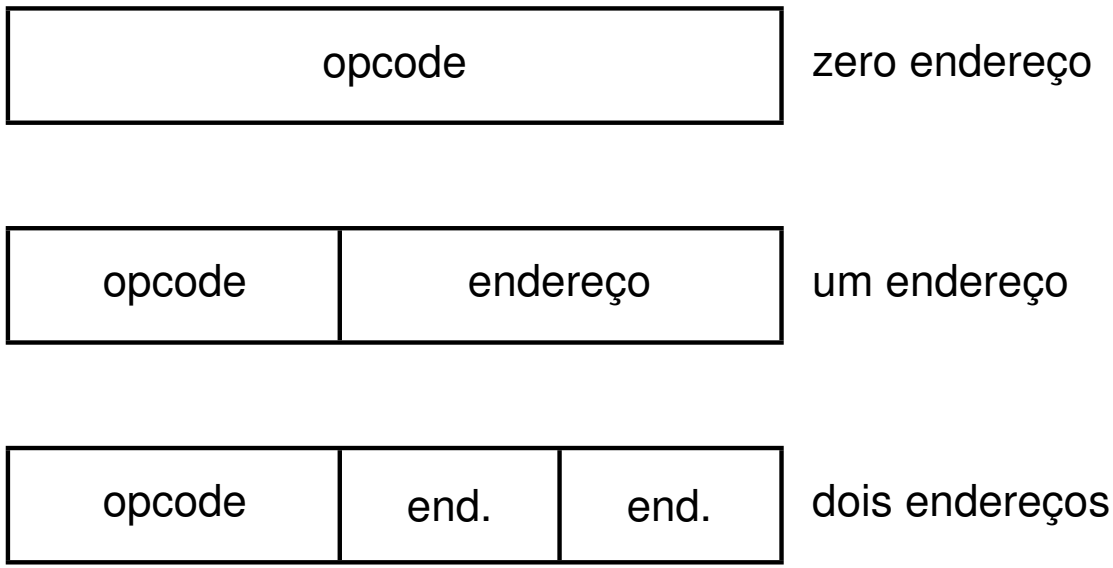
Outros exemplos: Intel e Motorola usam esquemas analógos, porém mais complexos.

Formato de instruções

Uma instrução contém:

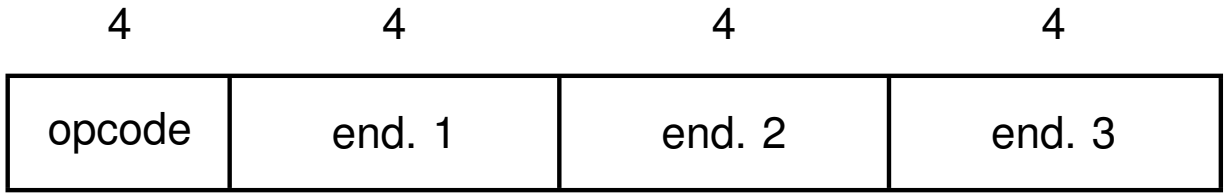
- opcode: código de operação
- endereços de operandos

Pode haver vários tipos de instruções.



“Opcodes que expandem”

Consideremos um exemplo de uma máquina cujas instruções são de 16 bits e endereços de 4 bits.



Cada endereço de 4 bits especificam, por exemplo, um dos 16 registradores da máquina. Com o formato acima podemos ter 16 instruções de 3 endereços.

“Opcodes que expandem”

Suponhamos, porém, que precisamos de

- 15 instruções de 3 endereços
- 14 instruções de 2 endereços
- 31 instruções de 1 endereço
- 16 instruções de 0 endereço

“Opcodes que expandem”

- 15 instruções de 3 endereços:

opcode			
0000	xxxx	yyyy	zzzz
0001	xxxx	yyyy	zzzz
	.		
	.		
1110	xxxx	yyyy	zzzz

- 14 instruções de 2 endereços:

opcode			
1111	0000	yyyy	zzzz
1111	0001	yyyy	zzzz
1111	0010	yyyy	zzzz
	.		
	.		
1111	1101	yyyy	zzzz

“Opcodes que expandem”

- 31 instruções de 1 endereço:

opcode			
1111	1110	0000	zzzz
1111	1110	0001	zzzz
	.		
	.		
1111	1110	1111	zzzz
1111	1111	0000	zzzz
1111	1111	0001	zzzz
	.		
	.		
	.		
1111	1111	1110	zzzz

“Opcodes que expandem”

- 16 instruções de 0 endereço:

opcode			
1111	1111	1111	0000
1111	1111	1111	0001
	.		
	.		
	.		
1111	1111	1111	1111

Em computadores reais, os opcode não são expandidos como no exemplo, de maneira tão regular e limpa, como veremos abaixo.

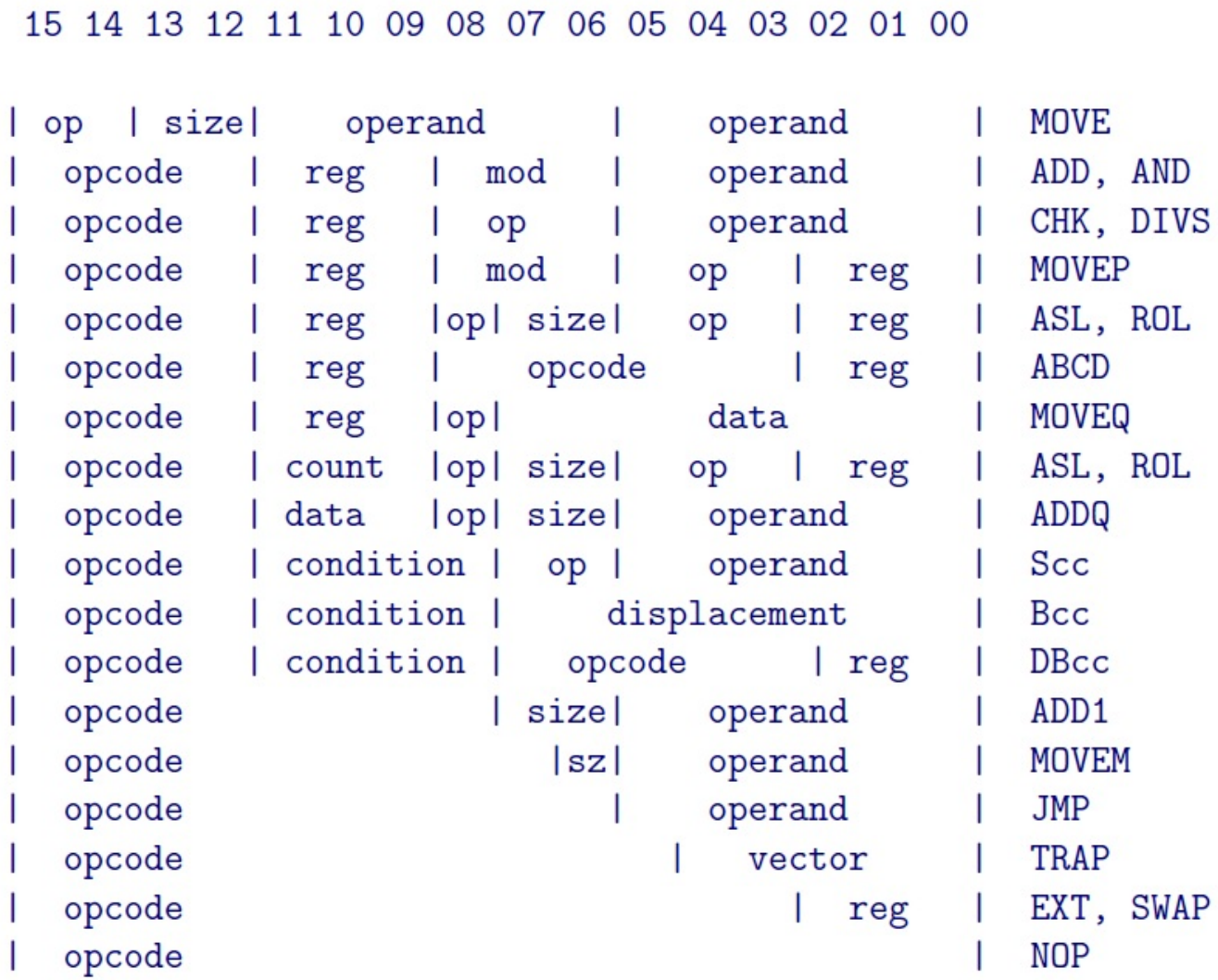
Formato de instrução no Motorola 68030

Teve uma influência grande do formato de instruções do PDP-11, porém muito mais complexo, devido a maior número de instruções. Além disso, o Motorola 68030 deve ainda considerar 3 tipos de operandos:

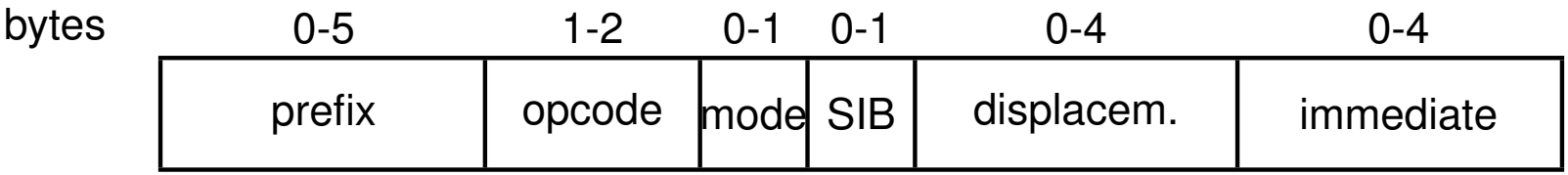
- byte 8 bits
- word 16 bits
- long 32 bits

A figura seguinte mostra as primeiras palavras de 18 formatos diferentes.

Formato de instrução no Motorola 68030



Formato de instrução no Intel 80386



O formato de instrução no Intel 80386 é também muito complicado, sendo a instrução mais curta de 1 byte e a mais comprida até 17 bytes.

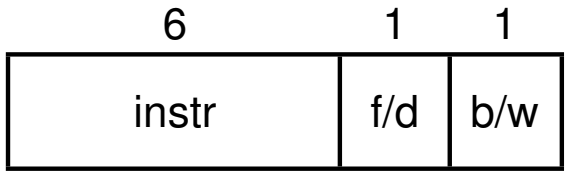
Há pouca estrutura ou regularidade no formato, exceto talvez nos 2 últimos bits do opcode:

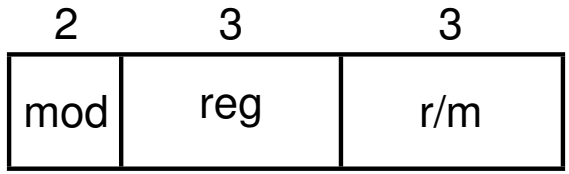
- O bit da direita do opcode indica se o endereço de memória, se usado, é fonte ou destino da operação.
- O penúltimo bit da direita do opcode indica se o operando é tipo byte ou word.

Formato de instrução no Intel 80386

A figura mostra os campos opcode e mode.

SIB é presente apenas no Motorola 68030, para dar um byte adicional ao modo.





f/d=fonte/destino

b/w=byte/word

RISC-V - ISA aberto

RISC-V (pronucia-se RISC-five) é um conjunto de instruções aberto (open ISA Instruction Set Architecture).

É livre para ser usado para qualquer finalidade, por qualquer pessoa ou empresa, sem precisa pagar licenças ou direitos autorais *royalties*

- O projeto começou em 2010 na Universidade de California, em Berkeley, tornando-se depois em RISC-V Foundation em 2015 e finalmente RISC-V International em 2019, uma organização sem fins lucrativos na Suíca.
- Foi projetado para processadores RISC costumizados para computações modernas, como aparelhos móveis, sistemas embarcados, internet das coisas, etc.
- RISC-V visa implementações de alto desempenho e baixo consumo de energia, com bases de 32, 64 e 128 bits.
- Até 2023, mais de 10 bilhões de processadores RISC-V foram produzidos.

RISC-V - uma breve introdução

O que se segue é baseado no texto
SiFive - An Introduction to RISC-V Architecture

- Mantida pela RISC-V International, em Suiça, com mais de 3 mil membros participantes.
- Adequado para uma variedade de sistemas de computação, desde microcontroladores até supercomputadores.

Nomes padronizados

RISC-V usa nomes padronizados para descrever cada ISA.

Format do nome: RV[###][letras]

- RV indica arquitetura RISC-V
- ### pode ser 32, 64, 128 para indicar o tamanho do registrador
- letra define uma extensão: I, M, A, F, D, G, C

Extensão	Descrição
I	Inteiro
M	Multiplicação e divisão de inteiros
A	Leitura/escrita atômica
F	Ponto flutuante precisão simples
D	Ponto flutuante precisão dupla
G	Propósito Geral: IMAFG
C	Instução 16 bits comprimidas

- Exemplos:
  RV32I = Implementação RISC-V mais básica
  RV32IMAC = registradores de 32 bits, extensões IMAC
  RV64GC = registradores de 64 bits, extensões IMAFGC

Registradores

- RV32I e RV64I tem 32 registradores de inteiros
- RV32E (E de Embarcado) tem 16 registradores de inteiros (para dispositivos embarcados com restrição de espaço)
- Extensões F e D tem 32 registradores de ponto flutuante
- ABI (Application Binary Interface) nomes padronizados para registradores.

Registrador	ABI Nome	Descrição
x0	zero	Valor zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5-7	t0-2	Temporaries
x8	s0/fp	Saved register/Frame pointer
x9	s1	Saved register
x10-11	a0-1	Function arguments/Return values
x12-17	a2-7	Function arguments
x18-27	s2-11	Saved registers
x28-31	t3-6	Temporaries

RISC-V modos

RISC-V tem 4 níveis de privilégios chamados modos.

Machine modo é o nível mais elevado.

Nível	Nome do modo	Abreviatura
0	User/application	U
1	Supervisor	S
2	Hypervisor	H
3	Machine	M

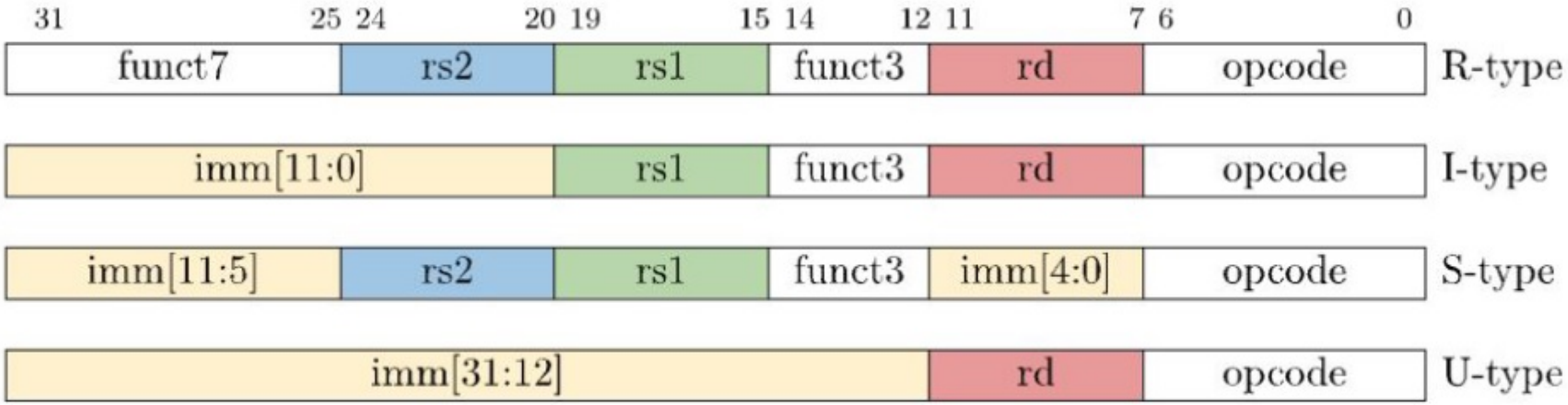
Formato de instruções de inteiros

RISC-V é uma arquitetura LOAD/STORE: Os valores tem que ser carregados nos registradores antes de qualquer operação.

Não tem operaões diretamente em memória.

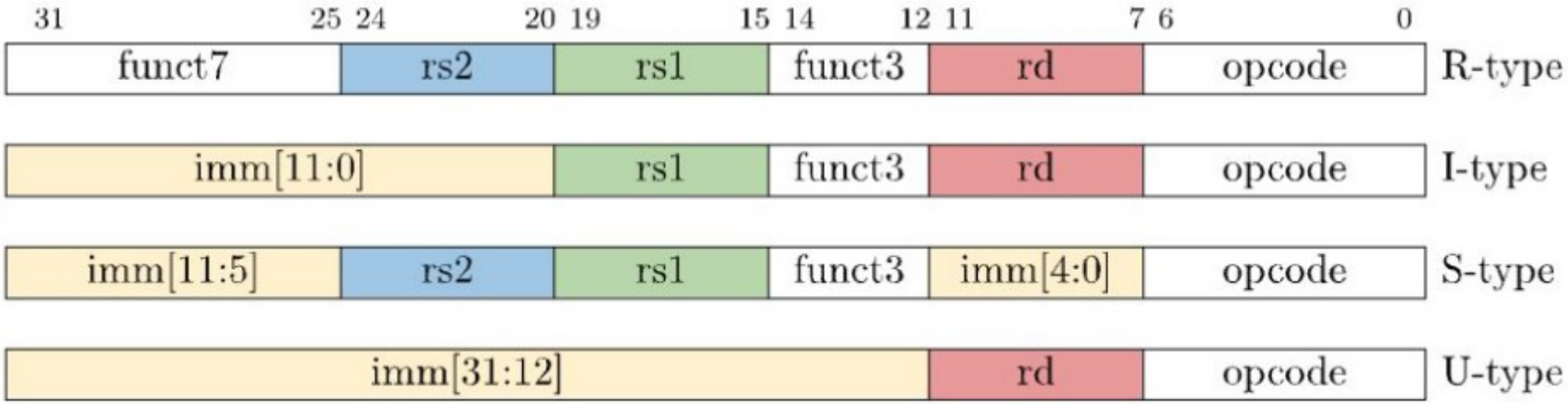
Há 4 formatos:

- R-type Register
- I-type Immedate
- S-type Store
- U-type Load with immediate



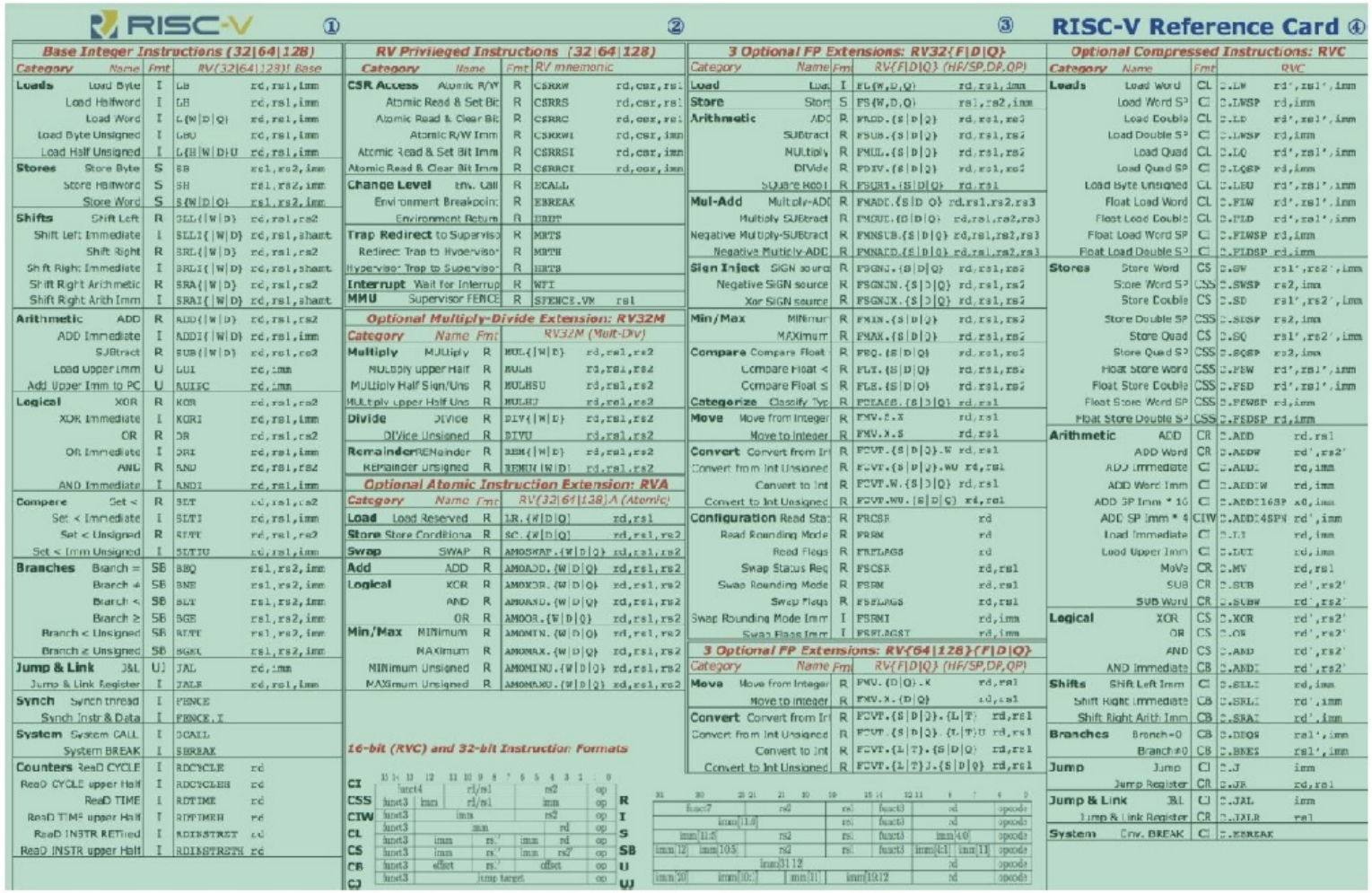
Formato de instruções de inteiros

- rs1 e rs2: registradores source
- rd: registrador destination
- Exemplo: add rd, rs1, rs2 (x[rd] = x[rs1] + x[rs2])
  Somar registrador x[rs1] com registrador x[rs2] e coloca o resultado no registrador x[rd]



RISC-V Reference Card

Source: SiFive



RISC-V Reference Card

① Base Integer Instructions (32|64|128)

Category	Name	Fmt	RV{32|64|128}I Base
Loads	Load Byte	I	LB rd,rs1,imm
	Load Halfword	I	LH rd,rs1,imm
	Load Word	I	L{W|D|Q} rd,rs1,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm
	Load Half Unsigned	I	L{H|W|D}U rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm
	Store Halfword	S	SH rs1,rs2,imm
	Store Word	S	S{W|D|Q} rs1,rs2,imm
Shifts	Shift Left	R	SLL{W|D} rd,rs1,rs2
	Shift Left Immediate	I	SLLI{W|D} rd,rs1,shamt
	Shift Right	R	SRL{W|D} rd,rs1,rs2
	Shift Right Immediate	I	SRLI{W|D} rd,rs1,shamt
	Shift Right Arithmetic	R	SRA{W|D} rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI{W|D} rd,rs1,shamt
Arithmetic	ADD	R	ADD{W|D} rd,rs1,rs2
	ADD Immediate	I	ADDI{W|D} rd,rs1,imm
	SUBtract	R	SUB{W|D} rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm
	Add Upper Imm to PC	U	AUIPC rd,imm
Logical	XOR	R	XOR rd,rs1,rs2
	XOR Immediate	I	XORI rd,rs1,imm
	OR	R	OR rd,rs1,rs2
	OR Immediate	I	ORI rd,rs1,imm
	AND	R	AND rd,rs1,rs2
	AND Immediate	I	ANDI rd,rs1,imm
Compare	Set <	R	SLT rd,rs1,rs2
	Set < Immediate	I	SLTI rd,rs1,imm
	Set < Unsigned	R	SLTU rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm
Branches	Branch =	SB	BEQ rs1,rs2,imm
	Branch ≠	SB	BNE rs1,rs2,imm
	Branch <	SB	BLT rs1,rs2,imm
	Branch ≥	SB	BGE rs1,rs2,imm
	Branch < Unsigned	SB	BLTU rs1,rs2,imm
	Branch ≥ Unsigned	SB	BGEU rs1,rs2,imm
Jump & Link	J&L	UJ	JAL rd,imm
	Jump & Link Register	I	JALR rd,rs1,imm
Synch	Synch thread	I	FENCE
	Synch Instr & Data	I	FENCE.I
System	System CALL	I	SCALL
	System BREAK	I	SBREAK
Counters	ReaD CYCLE	I	RDCYCLE rd
	ReaD CYCLE upper Half	I	RDCYCLEH rd
	ReaD TIME	I	RDTIME rd
	ReaD TIME upper Half	I	RDTIMEH rd
	ReaD INSTR RETired	I	RDINSTRET rd
	ReaD INSTR upper Half	I	RDINSTRETH rd

② RV Privileged Instructions (32|64|128)

Category	Name	Fmt	RV mnemonic
CSR Access	Atomic R/W	R	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	R	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	R	CSRRC rd,csr,rs1
	Atomic R/W Imm	R	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	R	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	R	CSRRCI rd,csr,imm
Change Level	trn. call	R	ECALL
	Environment Breakpoint	R	EBREAK
	Environment Return	R	ERET
Trap Redirect to Supervisor		R	MRTS
	Redirect Trap to Hypervisor	R	MRTH
	Hypervisor Trap to Supervisor	R	HRTS
Interrupt	Wait for Interrupt	R	WFI
MMU	Supervisor FENCE	R	SFENCE.VM rs1

Optional Multiply-Divide Extension: RV32M

Category	Name	Fmt	RV32M (Mult-Div)
Multiply	MULtiply	R	MUL{W|D} rd,rs1,rs2
	MULtiply upper Half	R	MULH rd,rs1,rs2
	MULtiply Half Sign/Uns	R	MULHSU rd,rs1,rs2
	MULtiply upper Half Uns	R	MULHU rd,rs1,rs2
Divide	DIVide	R	DIV{W|D} rd,rs1,rs2
	DIVide Unsigned	R	DIVU rd,rs1,rs2
Remainder	REMainder	R	REM{W|D} rd,rs1,rs2
	REMainder Unsigned	R	REMU{W|D} rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV{32|64|128}A (Atomic)
Load	Load Reserved	R	LR.{W|D|Q} rd,rs1
Store	Store Conditional	R	SC.{W|D|Q} rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.{W|D|Q} rd,rs1,rs2
Add	ADD	R	AMOADD.{W|D|Q} rd,rs1,rs2
Logical	XOR	R	AMOXOR.{W|D|Q} rd,rs1,rs2
	AND	R	AMOAND.{W|D|Q} rd,rs1,rs2
	OR	R	AMOOR.{W|D|Q} rd,rs1,rs2
Min,/Max	MINimum	R	AMOMIN.{W|D|Q} rd,rs1,rs2
	MAXimum	R	AMOMAX.{W|D|Q} rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.{W|D|Q} rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.{W|D|Q} rd,rs1,rs2

16-bit (RVC) and 32-bit Instruction Formats

③ 3 Optional FP Extensions: RV32{F|D|Q}

Category	Name	Fmt	RV{F|D|Q} (HF/SP,DP,QP)
Load	Load	I	FL{W,D,Q} rd,rs1,imm
Store	Store	S	FS{W,D,Q} rs1,rs2,imm
Arithmetic	ADD	R	FADD.{S|D|Q} rd,rs1,rs2
	SUBtract	R	FSUB.{S|D|Q} rd,rs1,rs2
	MULtiply	R	FMUL.{S|D|Q} rd,rs1,rs2
	DIVide	R	FDIV.{S|D|Q} rd,rs1,rs2
	SQUare ROOt	R	FSQRT.{S|D|Q} rd,rs1
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q} rd,rs1,rs2,rs3
	Multiply SUBtract	R	FMSUB.{S|D|Q} rd,rs1,rs2,rs3
	Negative Multiply-SUBtract	R	FNMSUB.{S|D|Q} rd,rs1,rs2,rs3
	Negative Multiply-ADD	R	FNMADD.{S|D|Q} rd,rs1,rs2,rs3
Sign Inject	SIGN source	R	FSGNJ.{S|D|Q} rd,rs1,rs2
	Negative SIGN source	R	FSGNJN.{S|D|Q} rd,rs1,rs2
	Xor SIGN source	R	FSGNJX.{S|D|Q} rd,rs1,rs2
Min/Max	MINimum	R	FMIN.{S|D|Q} rd,rs1,rs2
	MAXimum	R	FMAX.{S|D|Q} rd,rs1,rs2
Compare	Compare Float =	R	FEQ.{S|D|Q} rd,rs1,rs2
	Compare Float <	R	FLT.{S|D|Q} rd,rs1,rs2
	Compare Float ≤	R	FLE.{S|D|Q} rd,rs1,rs2
Categorize	Classify Typ	R	FCLASS.{S|D|Q} rd,rs1
Move	Move from Integer	R	FMV.S.X rd,rs1
	Move to Integer	R	FMV.X.S rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.W rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.WU rd,rs1
	Convert to Int	R	FCVT.W.{S|D|Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{S|D|Q} rd,rs1
Configuration	Read Status	R	FRCSR rd
	Read Rounding Mode	R	FRRM rd
	Read Flags	R	FRFLAGS rd
	Swap Status Reg	R	FSCSR rd,rs1
	Swap Rounding Mode	R	FSRM rd,rs1
	Swap Flags	R	FSFLAGS rd,rs1
	Swap Rounding Mode Imm	I	FSRMI rd,imm
	Swap Flags Imm	I	FSFLAGSI rd,imm

3 Optional FP Extensions: RV{64|128}{F|D|Q}

Category	Name	Fmt	RV{F|D|Q} (HF/SP,DP,QP)
Move	Move from Integer	R	FMV.{D|Q}.X rd,rs1
	Move to Integer	R	FMV.X.{D|Q} rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.{L|T} rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.{L|T}U rd,rs1
	Convert to Int	R	FCVT.{L|T}.{S|D|Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.{L|T}U.{S|D|Q} rd,rs1

Optional Compressed Instructions: RVC

Category	Name	Fmt	RVC
Loads	Load Word	CL	C.LW rd',rs1',imm
	Load Word SP	C	C.LWSP rd,imm
	Load Double	CL	C.LD rd',rs1',imm
	Load Double SP	C	C.LWSP rd,imm
	Load Quad	CL	C.LQ rd',rs1',imm
	Load Quad SP	C	C.LQSP rd,imm
	Load Byte Unsigned	CL	C.LBU rd',rs1',imm
	Float Load Word	CL	C.FLW rd',rs1',imm
	Float Load Double	CL	C.FLD rd',rs1',imm
	Float Load Word SP	C	C.FLWSP rd,imm
	Float Load Double SP	C	C.FLDSP rd,imm
Stores	Store Word	CS	C.SW rs1',rs2',imm
	Store Word SP	CSS	C.SWSP rs2,imm
	Store Double	CS	C.SD rs1',rs2',imm
	Store Double SP	CSS	C.SDSP rs2,imm
	Store Quad	CS	C.SQ rs1',rs2',imm
	Store Quad SP	CSS	C.SQSP rs2,imm
	Float Store Word	CSS	C.FSW rs2',rs1',imm
	Float Store Double	CSS	C.FSD rs2',rs1',imm
	Float Store Word SP	CSS	C.FSWSP rs2,imm
	Float Store Double SP	CSS	C.FSDSP rs2,imm
Arithmetic	ADD	CR	C.ADD rd,rs1
	ADD Word	CR	C.ADDW rd',rs2'
	ADD Immediate	CI	C.ADDI rd,imm
	ADD Word Imm	CI	C.ADDIW rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd',imm
	Load Immediate	CI	C.LI rd,imm
	Load Upper Imm	CI	C.LUI rd,imm
	MoVe	CR	C.MV rd,rs1
	SUB	CR	C.SUB rd',rs2'
	SUB Word	CR	C.SUBW rd',rs2'
Logical	XOR	CS	C.XOR rd',rs2'
	OR	CS	C.OR rd',rs2'
	AND	CS	C.AND rd',rs2'
	AND Immediate	CB	C.ANDI rd',rs2'
Shifts	Shift Left Imm	CI	C.SLLI rd,imm
	Shift Right Immediate	CB	C.SRLI rd',imm
	Shift Right Arith Imm	CB	C.SRAI rd',imm
Branches	Branch=0	CB	C.BEQZ rs1',imm
	Branch≠0	CB	C.BNEZ rs1',imm
Jump	Jump	CJ	C.J imm
	Jump Register	CR	C.JR rd,rs1
Jump & Link	J&L	CJ	C.JAL imm
	Jump & Link Register	CR	C.JALR rs1
System	Env. BREAK	CI	C.EBREAK

SiFive