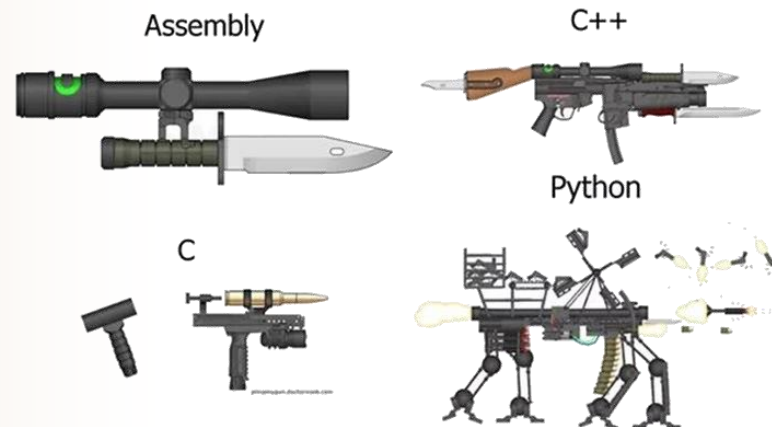




# Aula 4

## Arquiteturas de Processadores



Marcus Vinicius Lamar





# Arquiteturas de Processadores

## Principais arquiteturas hoje

- ARMv7 (32 bits) e ARMv9 (64 bits – compatível com v7)
- x86 (32 bits) e EM64T (64 bits - compatível com x86)

Vamos usar a arquitetura do processador RISC-V como exemplo de aplicação dos conceitos básicos em um projeto completo.

<http://www.riscv.org>



ISA não proprietária desenvolvida por uma comunidade.

Projeto iniciado em 2010 pela Universidade Califórnia - Berkeley.  
RISC-V Foundation fundada em 2015.

## O RISC-V:

- Arquitetura nova, projetada para eficiência em desempenho e consumo
- Open ISA, não necessita de licenciamento (ARM, x86, EM64T,...)
- ISA básica RV32, RV64, RV128 e diversas extensões
- Ferramentas open source de desenvolvimento disponíveis (gcc, simuladores, etc.)
- Chips e placas de desenvolvimento já disponíveis no mercado
- Apoiado por mais de 200 empresas. Excluindo ARM.



A Intel está desenvolvendo processador híbrido com ARM e RISC-V



# Arquiteturas RISC-V

RV32 – registradores de 32 bits

RV64 – registradores de 64 bits

RV128 – registradores de 128 bits

Tipo de instruções	Sufixo
ISA de inteiros	I
Instruções de Multiplicação e Divisão	M
Instruções atômicas (sincronização de memória)	A
Instruções de ponto flutuante (precisão simples)	F
Instruções de ponto flutuante (precisão dupla)	D
ISA Geral	IMAFD = G
Conjunto reduzido para sistemas embarcados	E

Modo normal: Instruções com 32 bits de tamanho

Modo condensado: Instruções com 16 bits de tamanho

Modo expandido: Instruções com  $n \times 16$  bits de tamanho (48,64,96,...)



# Arquitetura RISC-V RV32 : Operandos

Local do operando	Exemplo	Comentários
Banco de 32 Registradores	$x0, x1, x2, \dots, x31$	Local de acesso mais rápido a variáveis
Memória RAM	$M[0], M[8], M[16], \dots, M[2^N-8]$ $M[0], M[4], M[8], \dots, M[2^N-4]$ $M[0], M[2], M[4], \dots, M[2^N-2]$ $M[0], M[1], M[2], \dots, M[2^N-1]$	double word (RV64) word, half-word byte
Acesso imediato	<code>addi x5, x5, 123</code>	Local de acesso mais rápido a constantes

**RISC-V assembly language**

Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	add x5, x6, x7	$x5 = x6 + x7$	Three register operands; add
	Subtract	sub x5, x6, x7	$x5 = x6 - x7$	Three register operands; subtract
	Add immediate	addi x5, x6, 20	$x5 = x6 + 20$	Used to add constants
Data transfer	Load doubleword	ld x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Doubleword from memory to register
	Store doubleword	sd x5, 40(x6)	$\text{Memory}[x6 + 40] = x5$	Doubleword from register to memory
	Load word	lw x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Word from memory to register
	Load word, unsigned	lwu x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Unsigned word from memory to register
	Store word	sw x5, 40(x6)	$\text{Memory}[x6 + 40] = x5$	Word from register to memory
	Load halfword	lh x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Halfword from memory to register
	Load halfword, unsigned	lhu x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Unsigned halfword from memory to register
	Store halfword	sh x5, 40(x6)	$\text{Memory}[x6 + 40] = x5$	Halfword from register to memory
	Load byte	lb x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Byte from memory to register
	Load byte, unsigned	lbu x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Byte unsigned from memory to register
	Store byte	sb x5, 40(x6)	$\text{Memory}[x6 + 40] = x5$	Byte from register to memory
	Load reserved	lr.d x5, (x6)	$x5 = \text{Memory}[x6]$	Load; 1st half of atomic swap
	Store conditional	sc.d x7, x5, (x6)	$\text{Memory}[x6] = x5; x7 = 0/1$	Store; 2nd half of atomic swap
	Load upper immediate	lui x5, 0x12345	$x5 = 0x12345000$	Loads 20-bit constant shifted left 12 bits
Logical	And	and x5, x6, x7	$x5 = x6 \& x7$	Three reg. operands; bit-by-bit AND
	Inclusive or	or x5, x6, x8	$x5 = x6   x8$	Three reg. operands; bit-by-bit OR
	Exclusive or	xor x5, x6, x9	$x5 = x6 \wedge x9$	Three reg. operands; bit-by-bit XOR
	And immediate	andi x5, x6, 20	$x5 = x6 \& 20$	Bit-by-bit AND reg. with constant
	Inclusive or immediate	ori x5, x6, 20	$x5 = x6   20$	Bit-by-bit OR reg. with constant
	Exclusive or immediate	xori x5, x6, 20	$x5 = x6 \wedge 20$	Bit-by-bit XOR reg. with constant
Shift	Shift left logical	sll x5, x6, x7	$x5 = x6 \ll x7$	Shift left by register
	Shift right logical	srl x5, x6, x7	$x5 = x6 \gg x7$	Shift right by register
	Shift right arithmetic	sra x5, x6, x7	$x5 = x6 \gg x7$	Arithmetic shift right by register
	Shift left logical immediate	slli x5, x6, 3	$x5 = x6 \ll 3$	Shift left by immediate
	Shift right logical immediate	srl i x5, x6, 3	$x5 = x6 \gg 3$	Shift right by immediate
	Shift right arithmetic immediate	srai x5, x6, 3	$x5 = x6 \gg 3$	Arithmetic shift right by immediate
Conditional branch	Branch if equal	beq x5, x6, 100	if (x5 == x6) go to PC+100	PC-relative branch if registers equal
	Branch if not equal	bne x5, x6, 100	if (x5 != x6) go to PC+100	PC-relative branch if registers not equal
	Branch if less than	blt x5, x6, 100	if (x5 < x6) go to PC+100	PC-relative branch if registers less
	Branch if greater or equal	bge x5, x6, 100	if (x5 >= x6) go to PC+100	PC-relative branch if registers greater or equal
	Branch if less, unsigned	bltu x5, x6, 100	if (x5 < x6) go to PC+100	PC-relative branch if registers less, unsigned
	Branch if greater or equal, unsigned	bgeu x5, x6, 100	if (x5 >= x6) go to PC+100	PC-relative branch if registers greater or equal, unsigned
Unconditional branch	Jump and link	jal x1, 100	$x1 = \text{PC}+4$ ; go to PC+100	PC-relative procedure call
	Jump and link register	jalr x1, 100(x5)	$x1 = \text{PC}+4$ ; go to x5+100	Procedure return; indirect call

Cuidar pois o Patterson usa a RV64 na 1ª Edição do livro e RV32 na 2ª Edição

O Rars14 é RV32

O Rars15 pode executar RV32 ou RV64





# Convenção do Uso dos Registradores

Banco com 32 registradores   RV32: 32 bits cada um   RV64: 64 bits cada um

Todos os registradores são fisicamente iguais (exceção x0).

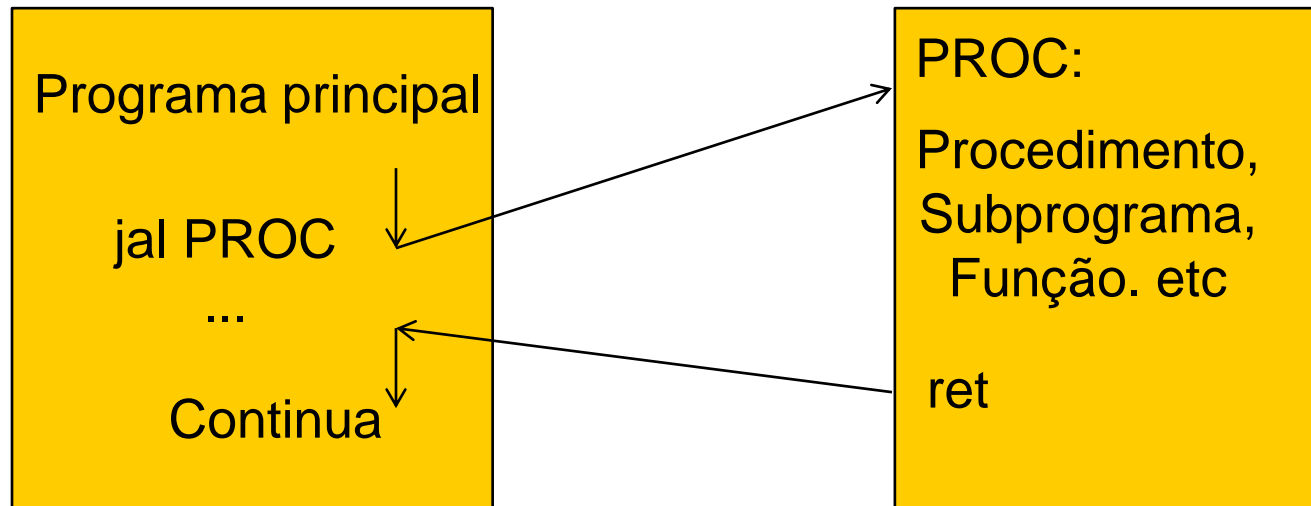
Logo: Convenção é o uso sugerido para fins de padronização

- x0 (zero): valor constante 0
- x1 (ra): endereço de retorno (*return address*)
- x2 (sp): ponteiro da pilha (*stack pointer*)
- x3 (gp): ponteiro global (*global pointer*)
- x4 (tp): ponteiro de thread (*thread pointer*)
- x5~x7 (t0~t2): registradores temporários
- x8 (s0/fp): registrador salvo ou ponteiro de frame (*frame pointer*)
- x9 (s1): registrador salvo
- x10~x11 (a0~a1): argumentos/resultados de funções
- x12~x17 (a2~a7): argumentos de funções
- x18~x27 (s2~s11): registradores salvos
- x28~x31 (t3~t6): registradores temporários



# Chamada de Procedimentos

## ...Convenção do uso dos Registradores



Registradores não-preserved: a0~a7, t0~t6

Registradores preserved: ra, sp, gp, tp, s0~s11





# Organização da Memória

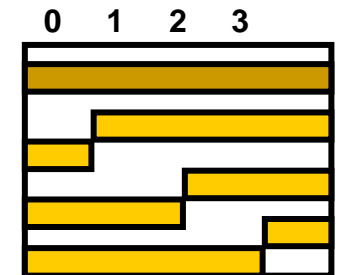
Endereço	Dado
0	8 bits
1	8 bits
2	8 bits
3	8 bits
4	8 bits
5	8 bits
6	8 bits
7	8 bits
8	8 bits
9	8 bits
...	...
$2^N-1$	8 bits

- Grande *array* unidimensional
- "*Byte addressing*" significa que um endereço aponta para um *byte* na memória

- **Double Word: 64 bits → 8 endereços**
- **Word: 32 bits → 4 endereços**
- **Half Word: 16 bits → 2 endereços**
- **Byte: 8 bits → 1 endereço.**

Alinhado

Não alinhado



- Processador:
  - **32 bits: N=32**
  - **64 bits: N=36, 40, 48, ..., 64**



# Ordenamento de Bytes (*Byte Order- Endianness*)

Número de 32 bits: <sup>MSB</sup>0x1<sup>LSB</sup>2345678

como é armazenado na memória no endereço 0 ?

**Big-Endian**  
MSB primeiro

Endereço	Dado
0	0x12
1	0x34
2	0x56
3	0x78
4	...
...	...

Ex.: IBM 360/370, Motorola 68000,  
AVR 32, HP PA, MIPS32, MIPS64

**Little-Endian**  
LSB primeiro

Endereço	Dado
0	0x78
1	0x56
2	0x34
3	0x12
4	...
...	...

Ex.: x86, EM64T, Z80, 8051, Vax, DEC  
Alpha, Atmel AVR, RISC-V(preferencial)

**Bi-Endian:** configurável

MIPS(R2000), Sparc v9, ARM, RISC-V (não recomendado)

# Arquitetura Load / Store

- Típica de processadores RISC
- Apenas as instruções **load** e **store** têm acesso à memória.
- As outras instruções operam apenas com registradores.
- Modo básico de definição de um endereço da memória:

**offset (register)**      endereço = registrador + offset

Outras formas podem ser permitidas pelo programa montador (Assembler) ou pelo próprio processador (ISA). Ex.: ISA ARM



# Princípios básicos de projeto de uma ISA

Utilizado no projeto das ISAs MIPS e RISC-V:

- Simplicidade favorece regularidade.
- Menor significa mais rápido.
- Bons projetos exigem bons compromissos.

*Objetivos de projeto de uma ISA:*

*maximizar o desempenho,  
minimizar o custo,  
reduzir o tempo de projeto.*



# Arquitetura RISC-V

## Instruções aritméticas

- Instruções **tipo-R**: 3 operandos regulares
- A ordem dos operandos é fixa: Destino, Origem 1, Origem 2

Exemplos:

Código C :

```
int a,b,c;  
a = b + c;
```

Código RISC-V:

```
add s0, s1, s2
```

Registradores são associados às variáveis pelo **compilador**

Código C:

```
int f,g,h,i,j;  
f=(g+h) - (i+j);
```

Código RISC-V:

```
add t0, s1, s2  
add t1, s3, s4  
sub s0, t0, t1
```



# Arquitetura RV32I

## Instruções de acesso à memória

- Instruções:                      LOAD                      STORE
- Exemplos:
 

word :	<code>lw s0, 16(t0)</code>	<code>sw s0, 16(t0)</code>
half-word:	<code>lh s0, 16(t0)</code>	<code>sh s0, 16(t0)</code>
byte:	<code>lb s0, 16(t0)</code>	<code>sb s0, 16(t0)</code>
- Variáveis em C são armazenadas na memória RAM!
- Exemplo:
 

Código C:	<code>int h, A[100];</code> <code>A[12] = h + A[8];</code>
Código RV32I:	<code>lw t0, 32(s1)</code> <code>add t0, s0, t0</code> <code>sw t0, 48(s1)</code>

# Arquiteturas RISC-V

## Instruções com operando Imediato

- Instruções **tipo-I**: Imediato
- É comum a operação com constantes, logo agilize!
- Exemplo:

Código C:

```
int a,b;  
b++;  
a=b-37;
```

Código RISC-V:

```
addi s1,s1,1  
addi s0,s1,-37
```

Obs.: Nesta arquitetura não existe `subi` !



# Exemplo de procedimento – RV32I

## Compilar o código:

```
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

```
swap:
    slli t0,a1,2      # calcula o offset em bytes
    add t0,t0,a0      # calcula o endereço em bytes
    lw t1,0(t0)       # lê o valor v[k]
    lw t2,4(t0)       # lê o valor v[k+1]
    sw t2,0(t0)       # escreve em v[k]
    sw t1,4(t0)       # escreve em v[k+1]
    jalr zero,ra,0    # retorna da função
```