

Arquitetura de Computadores II

Linguagem de máquina e instruções

Professor Matheus Alcântara Souza

CISC: Complex Instruction Set Computing

- Utilizado no início da história de desenvolvimento de processadores
- Conjunto de instruções com formatos variados
- Uma única instrução do tipo ADD pode:
 - acessar registradores e memória
 - utilizar modos de endereçamento imediato, direto ou indexado
- As instruções possuem uma grande amplitude nos modos de endereçamento

RISC: Reduced Instruction Set Computing

- Uma pequena fração do código de programa precisa acessar memória
- Acessar memória é mais lento do que acessar registradores
- Número reduzido de instruções que acessam a memória
- Manipulações de dados pelo processador são concentradas em registradores
- Instruções de acesso a memória:
 - LOAD: Busca dados em endereços da memória e grava em registradores
 - STORE: Grava dados dos registradores em endereços de memória.

GPP: General Purpose Processor (Processador de propósito geral)

- Possui um conjunto de instruções para utilização em diversas aplicações
- Flexibilidade
- Instruções não otimizadas para execução de tarefas
- Desempenho
- Exemplos:
 - Processadores 386
 - Pentium
 - Itanium
 - MIPS4700
 - Ultra Sparc III

ASIP: Application Specific Instruction Set Processor(Processador aplicação específica)

- Projeto dedicado do conjunto de instruções e da arquitetura
- Flexibilidade
- Desempenho
- Menos flexível que um GPP
- É mais lento que um ASIC (Application Specific Integrated Circuit)
- Exemplos:
- Processadores dedicados para imagens, sinais e redes

Existe diferença no conjunto de instruções de CISC, RISC, GPP e ASIP?

Microarquitetura de processadores

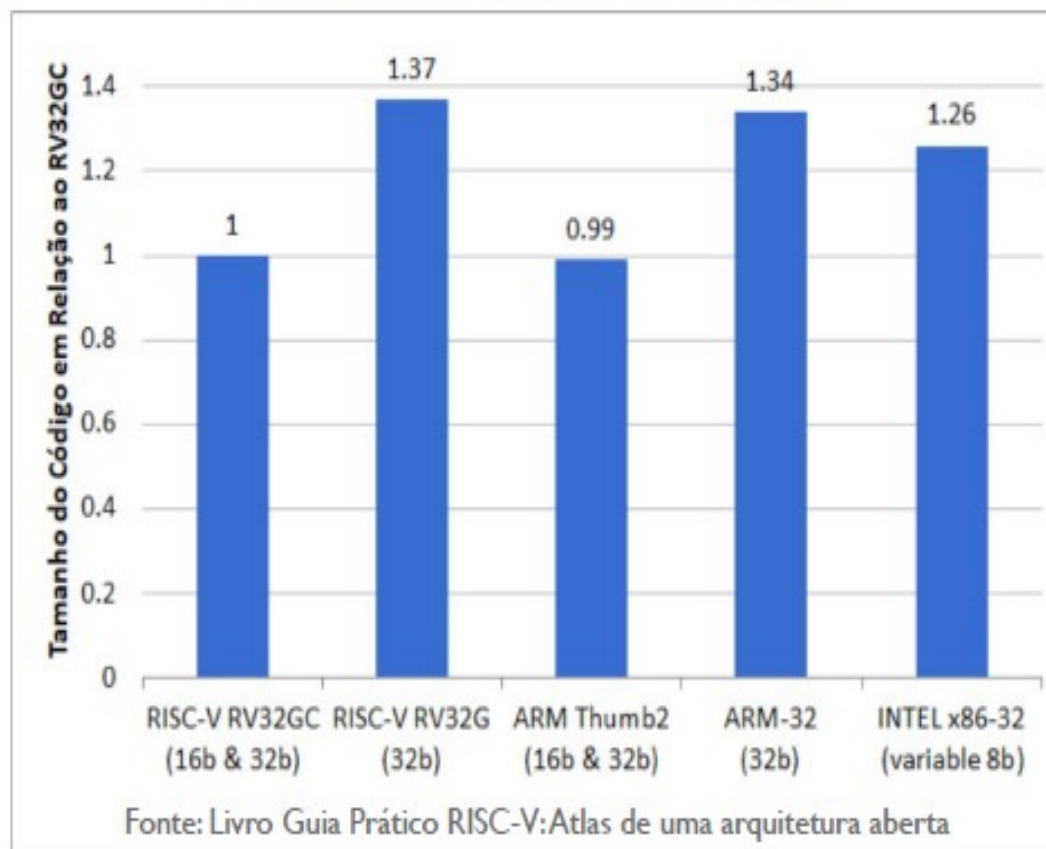
- A microarquitetura é responsável por
 - busca de instruções
 - decodificação de instruções
 - execução de instruções
- Instrução: um comando enviado para o processador
- Vamos estudar a microarquitetura do processador RISC-V!

Introdução à arquitetura RV32

- ISA (Instruction Set Architecture) RISC moderna
 - Introduzida em 2011
- ISA aberta! (uso livre de *royalties*)
- Funcionalidades e características desenvolvidas com base nos acertos e erros de ISAs que já estão no mercado há mais de 30 anos (x86 e ARM)
 - Mais simples do que ARM e x86
 - Figura 2.7 do livro “Guia Prático RISC-V: Atlas de uma arquitetura aberta”
<http://riscvbook.com/portuguese/>

- Mais simples do que ARM e x86

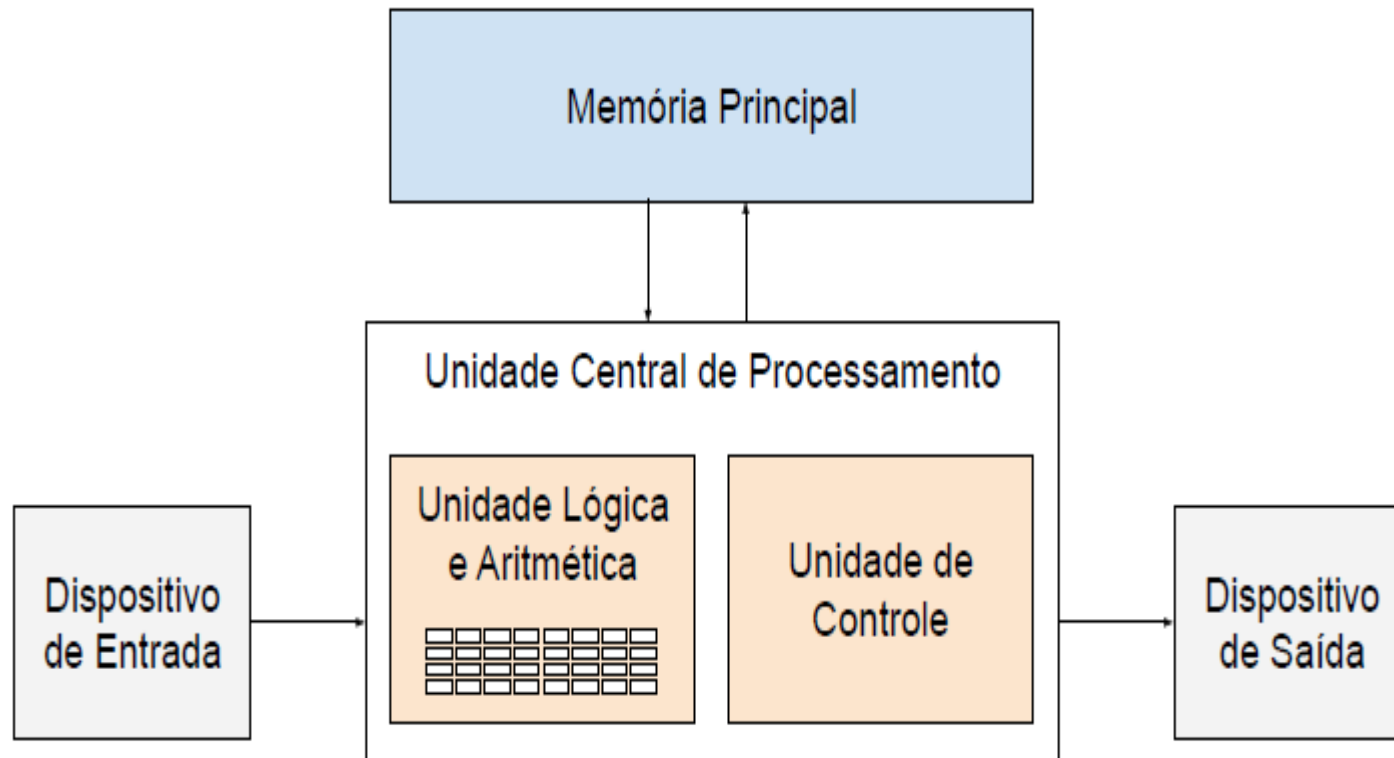
Tamanho relativo de programas do *benchmark* SPEC CPU2006 compilados com o GCC.



- Mantida atualmente pela Fundação RISC-V
<http://www.riscv.org>
- Fundação aberta e sem fins lucrativos

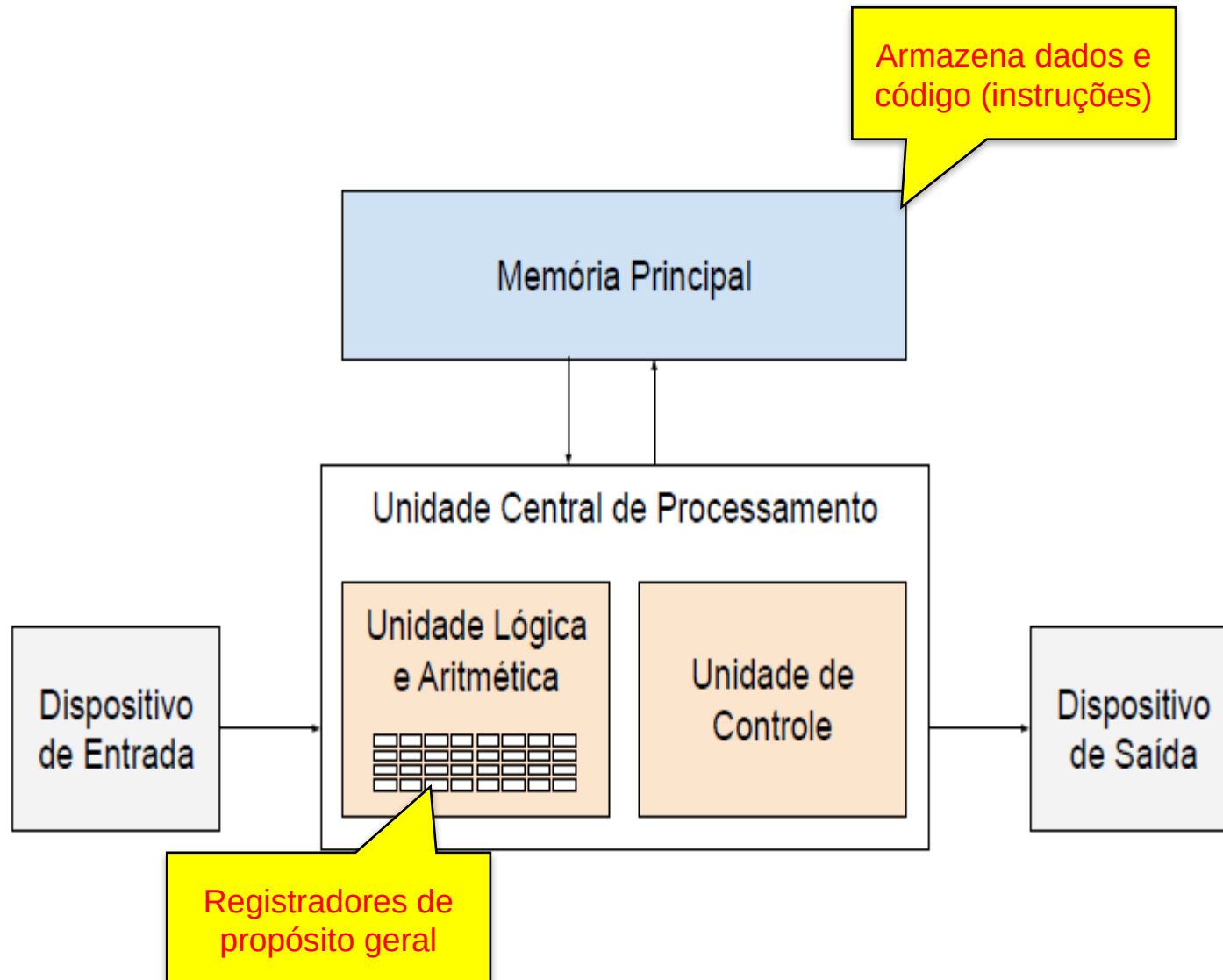
Arquitetura do RISC-V

- Tem elementos da Arquitetura de von Neumann



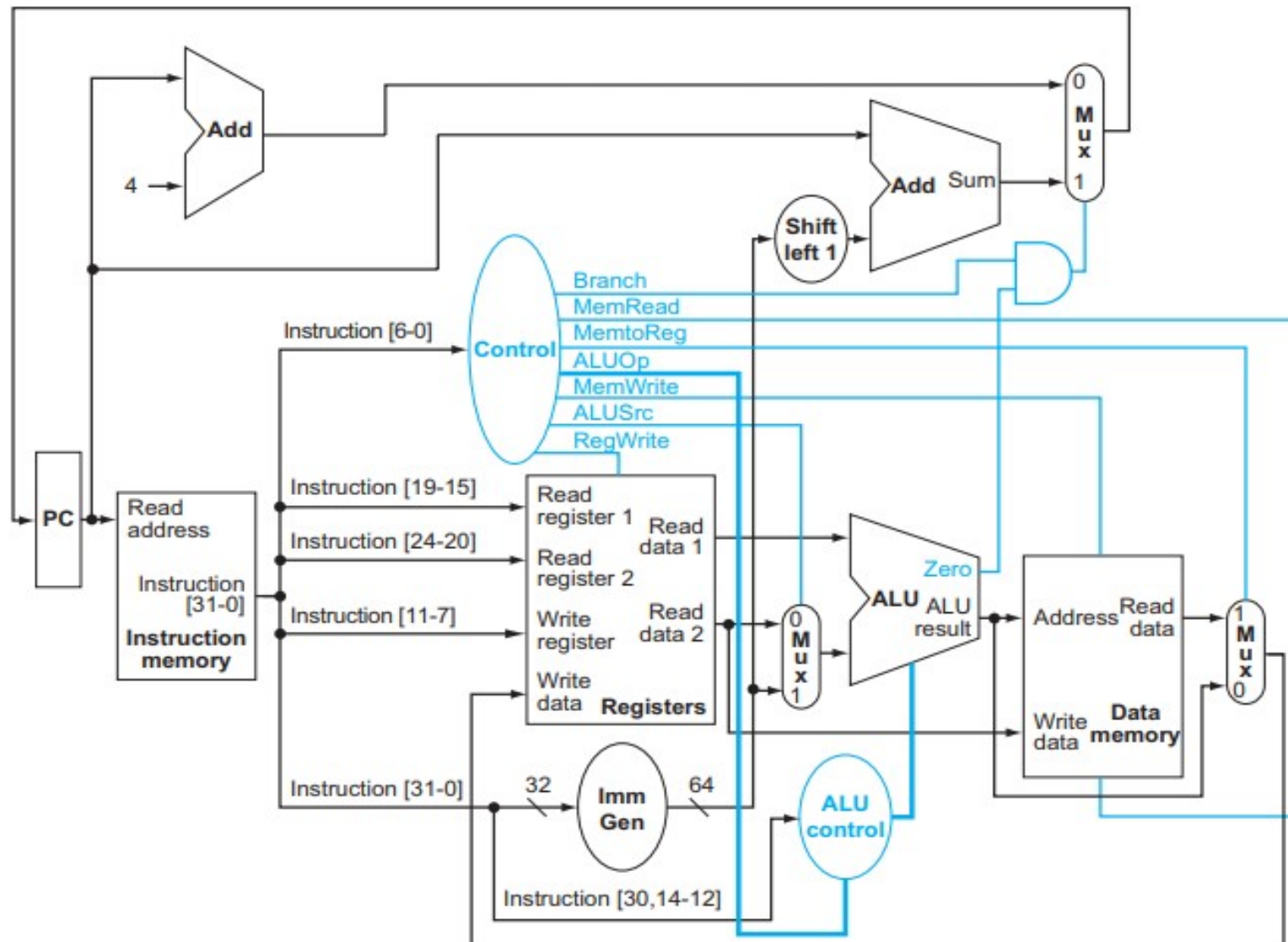
Arquitetura do RISC-V

- Tem elementos da Arquitetura de von Neumann



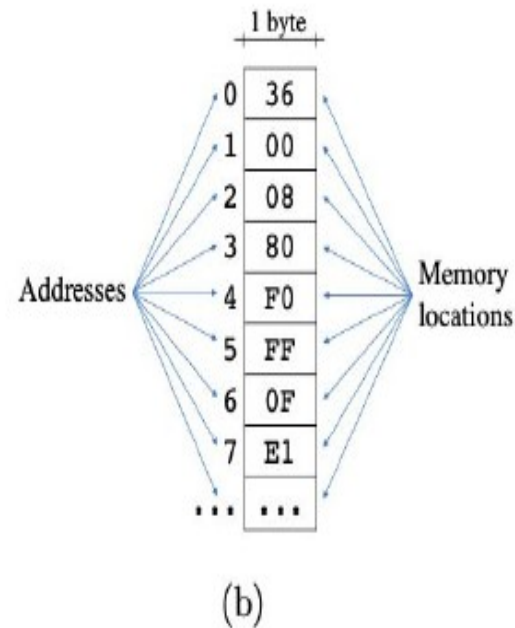
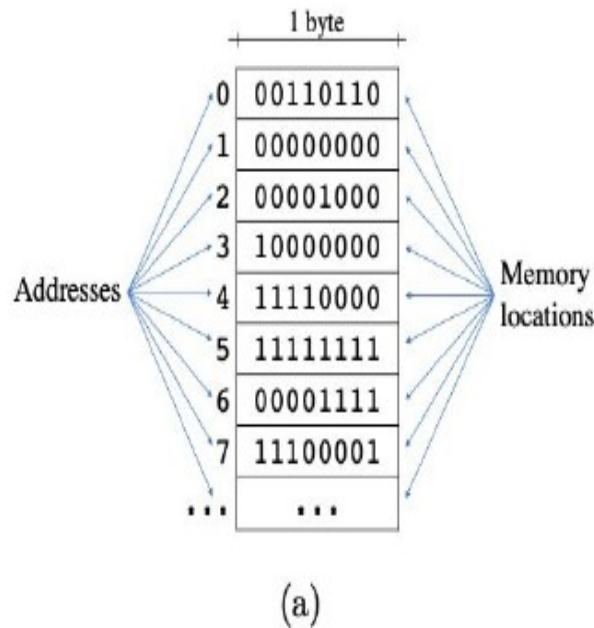
Arquitetura do RISC-V - Microarquitetura

- Caminho de dados simplificado + Unidade de Controle



Arquitetura do RISC-V - Memória

- A memória é endereçada a bytes
 - Cada endereço de memória armazena 1 byte
 - Tipos de dados maiores do que 1 byte ocupam múltiplos endereços de memória, consecutivos



Arquitetura do RISC-V – Conjuntos de instruções

- Existem diversos conjuntos de instruções:
 - **RV32I**: Conjunto base de 32 bits com instruções para operações com números inteiros
 - **RV32M**: Instruções de multiplicação e divisão
 - **RV32F** e **RV32D**: Instruções de ponto-flutuante
 - **RV32A**: Instruções atômicas
 - **RV32C**: Instruções compactas, de 16 bits
 - **RV32V**: Instruções vetoriais (SIMD)

- Nesta disciplina focaremos no conjunto RV32IM
 - Conjunto base de 32 bits + instruções para multiplicação e divisão de números inteiros
 - Instruções de movimentação de dados (LOAD e STORE), operações lógicas e aritméticas, comparação de valores, saltos condicionais e incondicionais, chamadas de funções, ...

Arquitetura do RV32 – “Tipos de dados”

- Tipos básicos de dados da arquitetura

byte: 1 byte

unsigned byte: 1 byte (sem sinal)

halfword: 2 bytes

unsigned halfword: 2 bytes (sem sinal)

word: 4 bytes

unsigned word: 4 bytes (sem sinal)

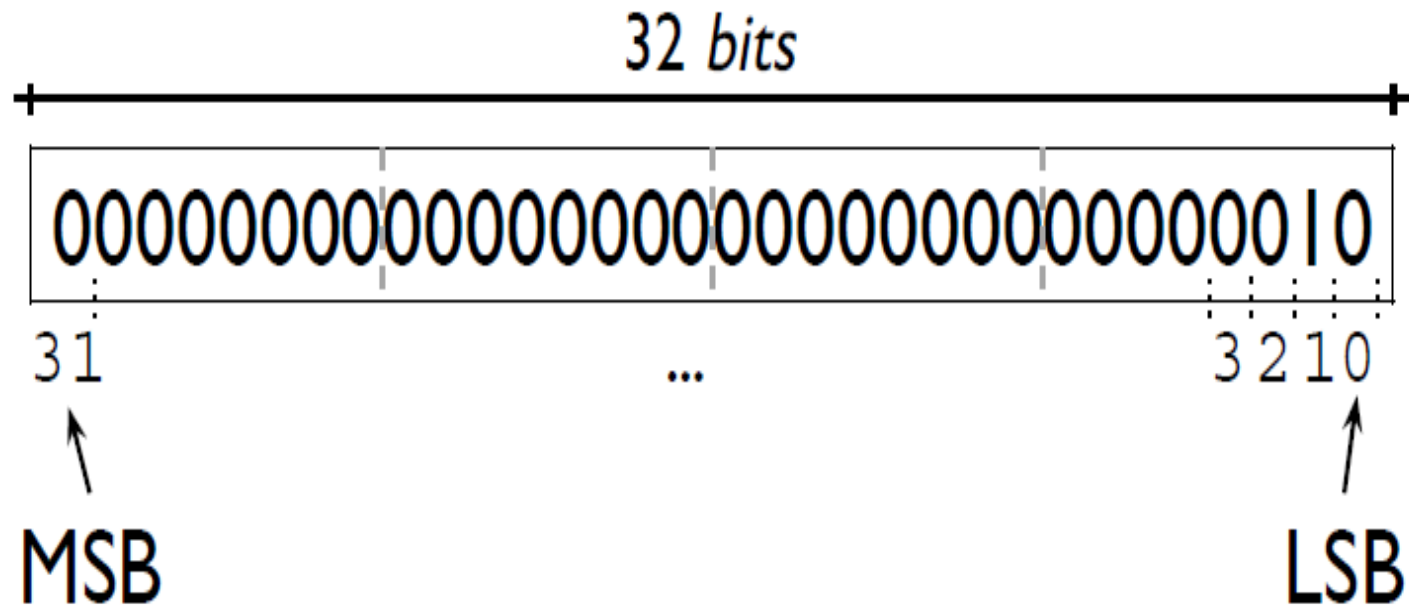
Arquitetura do RV32 – “Tipos de dados”

- Mapeamento de tipos de dados da linguagem C para tipos básicos de dados na arquitetura RV32

C datatype	RV32I native datatype	size in bytes
bool	byte	1
char	byte	1
unsigned char	unsigned byte	1
short	halfword	2
unsigned short	unsigned halfword	2
int	word	4
unsigned int	unsigned word	4
long	word	4
unsigned long	unsigned word	4
void*	unsigned word	4

Arquitetura do RV32 – Registradores

- Um registrador



Arquitetura do RV32 – Registradores

- Banco de registradores

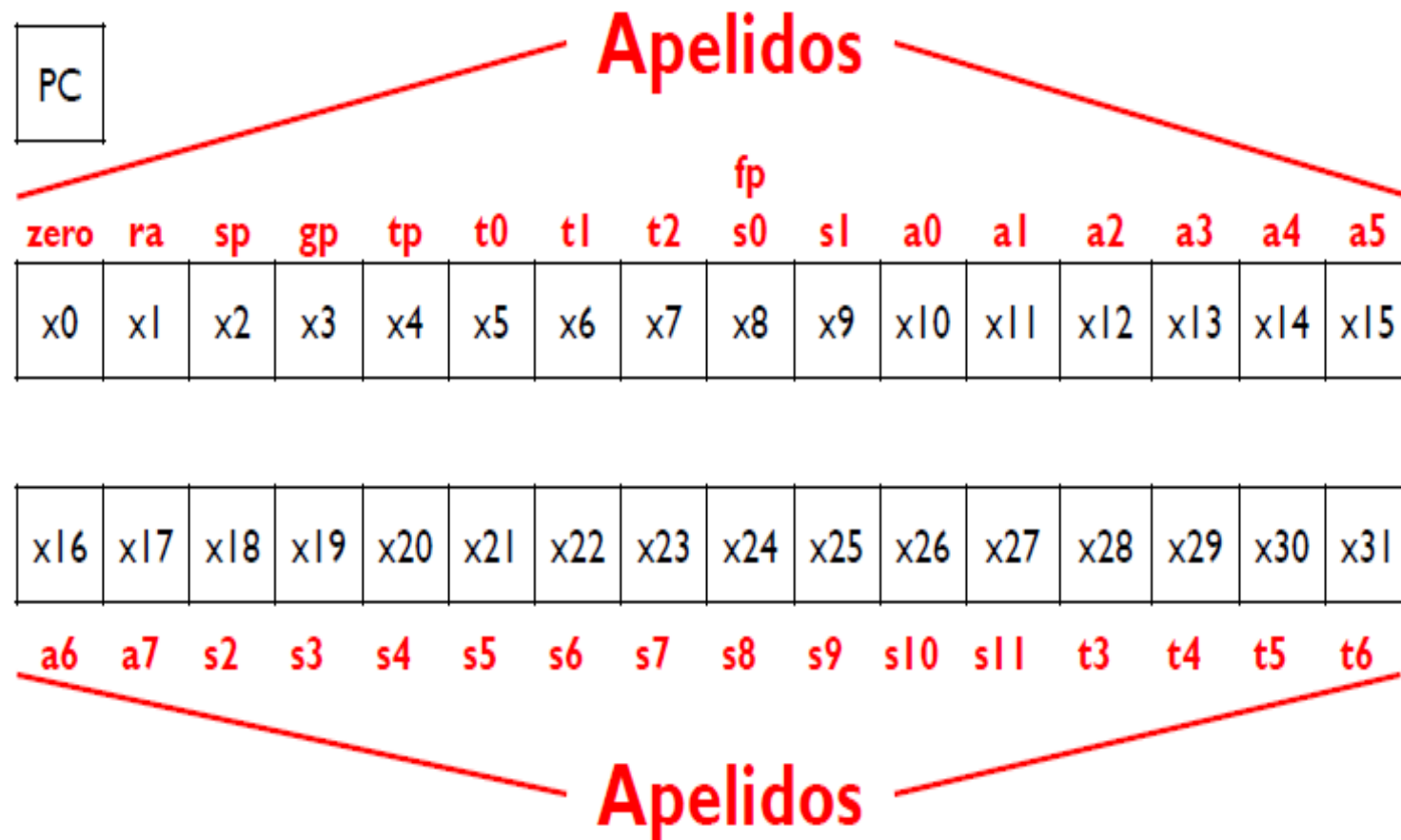
PC

x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----

x16	x17	x18	x19	x20	x21	x22	x23	x24	x25	x26	x27	x28	x29	x30	x31
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Arquitetura do RV32 – Registradores

- Banco de registradores



Arquitetura do RV32 – Registradores

- Banco de registradores

Apelido	Significado
pc	Program Counter (Apontador/Contador de programa)
a0, a1	Argumentos de função / retorno de função
a2 - a7	Argumentos de função
s0 - s11	Registradores salvos
t0 - t6	Registradores temporários
zero	Contém sempre o valor 0 (zero)
ra	Endereço de retorno
sp	Ponteiro de pilha
gp	Ponteiro global
tp	Ponteiro de thread

Arquitetura do RV32 – Transferência de dados

- **LOAD/STORE:** Os valores devem que ser carregados nos registradores antes de realizar as operações
 - Não há instruções que operam diretamente em valores na memória!

```
lw  a5, 0(a0)           # a5 <= Mem[a0]
add a6, a5, a5           # a6 <= a5+a5
sw  a6, 0(a0)           # Mem[a0] <= a6
```

Arquitetura do RV32 – Transferência de dados

- LOAD/STORE

Registradores

a0	00	00	00	00
a1	00	00	00	04
a2				
a3				

Memória

06	0
00	1
00	2
00	3
00	4
00	5
00	6
00	7
...	..
	.

PC →

```
lw    a5, 0(a0)
add   a6, a5, a5
sw    a6, 0(a1)
```


Arquitetura do RV32 – Transferência de dados

- LOAD/STORE

Registradores

a0	00	00	00	00
a1	00	00	00	04
a2	00	00	00	06
a3				

Memória

06	0
00	1
00	2
00	3
00	4
00	5
00	6
00	7
...	..
	.

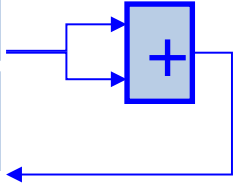
PC → **lw a5, 0(a0)**
add a6, a5, a5
sw a6, 0(a1)

Arquitetura do RV32 – Transferência de dados

- LOAD/STORE

Registradores

a0	00	00	00	00
a1	00	00	00	04
a2	00	00	00	06
a3	00	00	00	0C



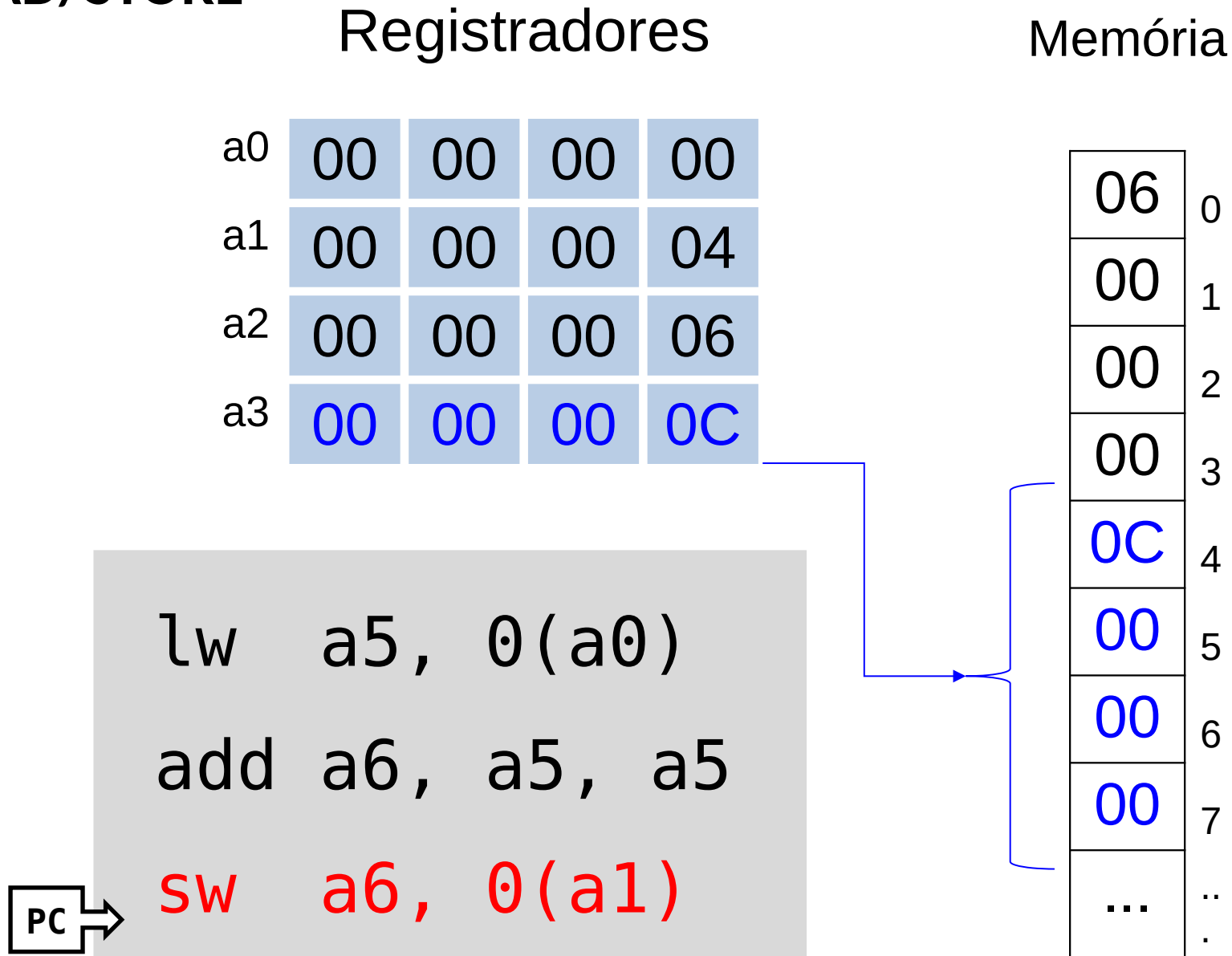
Memória

06	0
00	1
00	2
00	3
00	4
00	5
00	6
00	7
...	..
	.

PC → `lw a5, 0(a0)`
`add a6, a5, a5`
`sw a6, 0(a1)`

Arquitetura do RV32 – Transferência de dados

- LOAD/STORE



A linguagem de montagem

- Programas em linguagem de montagem possuem:
 - Rótulos
 - Instruções de montagem
 - Diretivas de montagem
 - Comentários

- **Comentários** são anotações no código
 - São descartados pelo pré-processador do montador.
 - 2 tipos: Comentários de linha e multi-linha

- **Comentários de linha (GNU Assembler)**
 - Delimitado pelo caracter # no caso do RV32I
 - Tudo entre a primeira ocorrência de # e o fim da linha

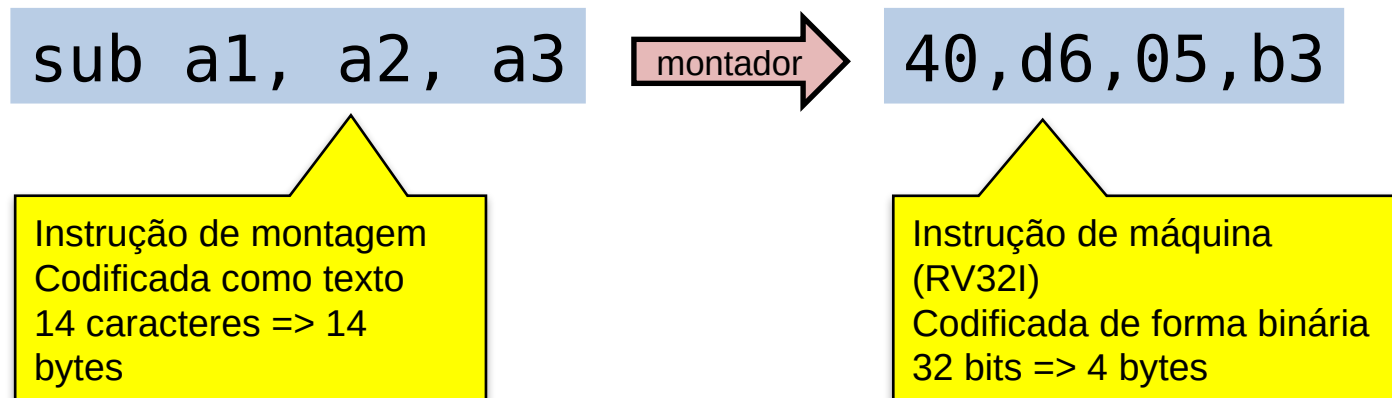
```
sub a1, a2, a3      # subtrai x de y
### Variável x ###
x:  .word 10
# add a0, a1, a2      # soma z e y
```

- **Comentários multi-linha (GNU Assembler)**
 - Delimitado pelos pares de `/*` e `*/`
 - Similar a C/C++

```
x:    .word 10    /* Variável x */  
  
/* Rotina trunc42:  
   Entrada: valor a ser truncado em a0  
   Retorno? Se a0>42 então 42, senão a0  
*/  
trunc42:
```


RISC-V – Linguagem de montagem – Instruções de montagem

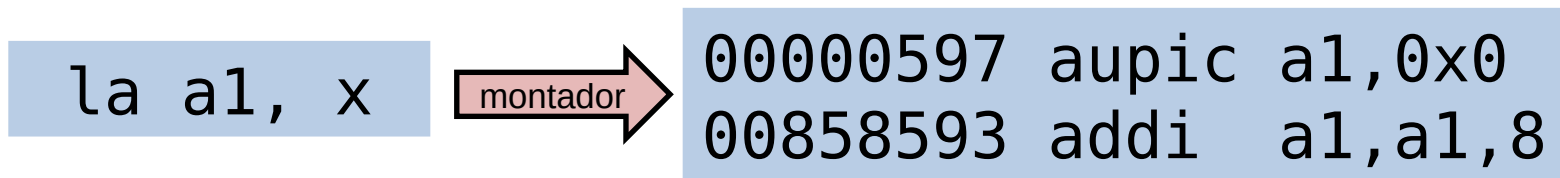
- **Instruções de montagem** são as instruções do programa
 - São codificadas como texto
- Instruções de montagem são traduzidas para instruções de máquina pelo montador:



- **Instruções de montagem** são traduzidas para instruções de máquina pelo montador
 - Geralmente uma instrução de montagem é traduzida para uma instrução de máquina
 - **Pseudo-instruções** são instruções em linguagem de montagem que não existem na linguagem de máquina
 - São traduzidas pelo montador para uma ou mais instruções de máquina.



- **Instruções de montagem** são traduzidas para instruções de máquina pelo montador
 - Geralmente uma instrução de montagem é traduzida para uma instrução de máquina
 - **Pseudo-instruções** são instruções em linguagem de montagem que não existem na linguagem de máquina
 - São traduzidas pelo montador para uma ou mais instruções de máquina.



- Sintaxe de instruções de montagem RV32I
 - Mnemônico + parâmetros (operandos)
 - Mnemônico identifica a operação
 - Ex: add => soma

RISC-V – Linguagem de montagem – Instruções de montagem

- Parâmetros das instruções de montagem RV32I
 - lab: Símbolos (ex: nomes de rótulos)
 - Imm: Constantes numéricas
 - rs1, rs2, rd: Registradores
 - Nome oficial (x0-x31) ou apelidos

RV32IM registers (prefix x) and their aliases

x0 zero	x1 ra	x2 sp	x3 gp	x4 tp	x5 t0	x6 t1	x7 t2	x8 s0	x9 s1	x10 a0	x11 a1	x12 a2	x13 a3	x14 a4	x15 a5
x16 a7	x17 a8	x18 s2	x19 s3	x20 s4	x21 s5	x22 s6	x23 s7	x24 s8	x25 s9	x26 s10	x27 s11	x28 t3	x29 t4	x30 t5	x31 t6

Main control status registers

CSRs:	mtvec	mepc	mcause	mtval	mstatus	mscratch
Fields of mstatus:	mie	mpie	mip			

RISC-V – Linguagem de montagem – Instruções de montagem

- Parâmetros das instruções de montagem RV32I
 - lab: Símbolos (ex: nomes de rótulos)
 - Imm: Constantes numéricas
 - rs1, rs2, rd: Registradores
 - Nome oficial (x0-x31) ou apelidos

/ Exemplos de instruções de montagem */*

```
sub    a1, a2, a3
addi   a0, a1, 12
la     a0, x
ret
```