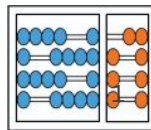


MC404AE - Organização Básica de Computadores e Ling. Montagem

Execução de Programas em Computadores

Prof. Allan M. de Souza



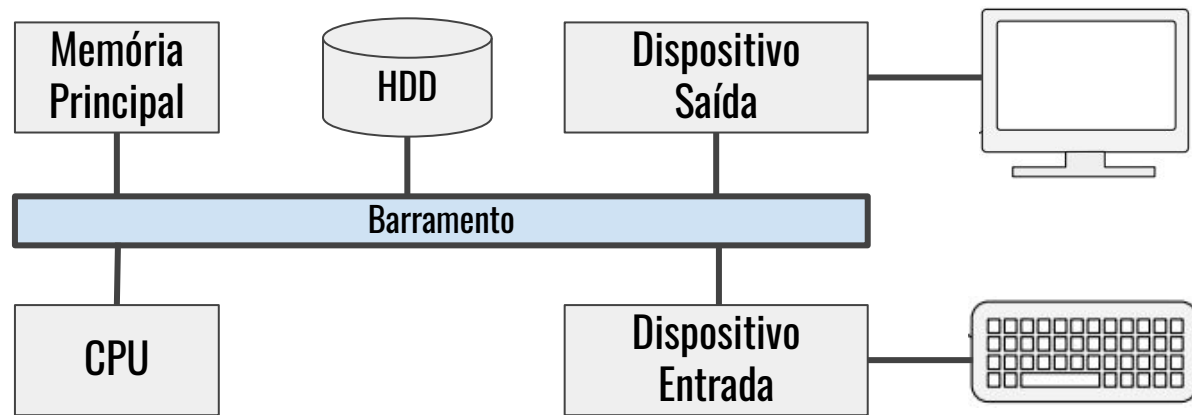
Agenda

- Componentes de um computador
 - Memória principal
 - CPU
 - Memória secundária
 - Barramento
 - Periféricos
- Codificação de programas de computador
- Geração de programas nativos
- Execução de programas nativos

Componentes de um Computador

Um computador é geralmente composto pelos seguintes componentes:

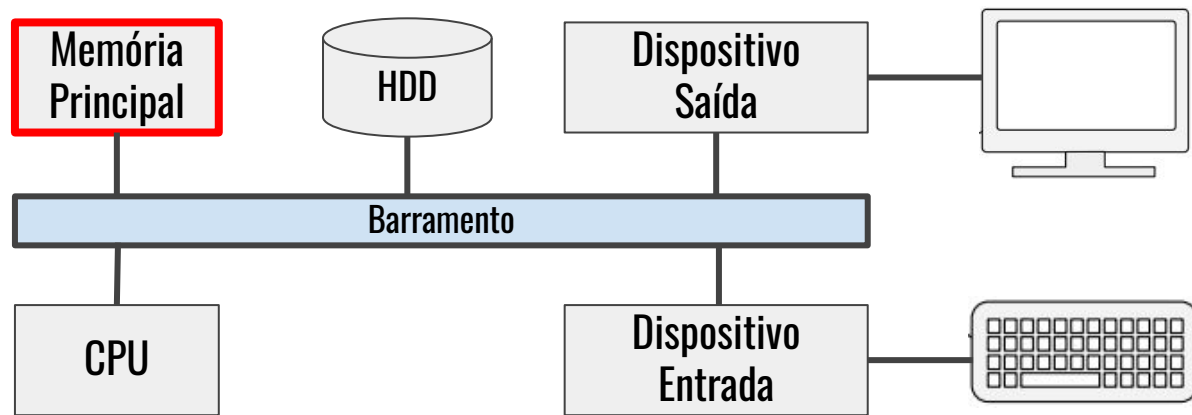
- Memória principal
- CPU – Unidade de Processamento Central
- Memória secundária
- Barramentos
- Periféricos



Componentes de um Computador

Um computador é geralmente composto pelos seguintes componentes:

- Memória principal
- CPU – Unidade de Processamento Central
- Memória secundária
- Barramentos
- Periféricos



A Memória Principal

- A memória principal armazena as instruções e dados de programas que estão sendo executados.
- Dados e instruções são codificados Memory de forma binária (sequências de zeros ou uns)
- A Unidade Central de Processamento (CPU) lê e escreve dados e instruções da memória principal!

Memória Principal

0	00100001
1	10101111
2	10000100
3	01111101
4	01010101
5	01100110
6	10000100
7	01111101
8	01100110
9	10000100

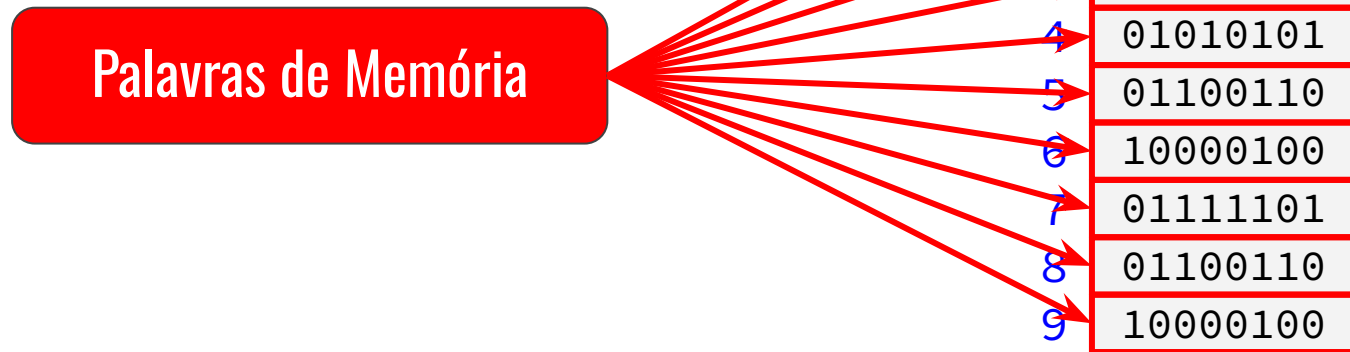
Dados e
Instruções

Dados vs Instruções

Formato	Bits																																	
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
R	func7							rs2					rs1					func3			rd					opcode								
I	imm[11:0]												rs1					func3			rd					opcode								
S	imm[11:5]							rs2					rs1					func3			imm[4:0]					opcode								
SB	[12]	imm[10:5]							rs2					rs1					func3			Imm[4:1]				[11]	opcode							
U	imm[31:12]																	rd					opcode											
UJ	[20]	imm[10:1]										[11]	imm[19:12]										rd					opcode						

A Memória Principal

- A memória principal é organizada em **palavras de memória**

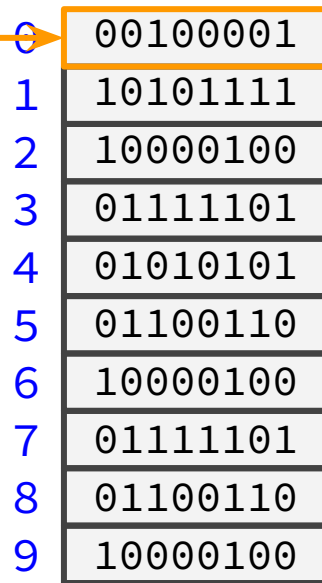


A Memória Principal

- A memória principal é organizada em **palavras de memória**
- Cada palavra de memória armazena uma sequência de bits
- Leituras e escritas na memória são realizadas a nível de palavras

Neste exemplo, a primeira palavra de memória armazena a sequência 00010011

Memória Principal



The diagram illustrates the Main Memory structure. It consists of a vertical list of memory addresses from 0 to 9, each corresponding to a word of memory. The words are represented as 8-bit binary sequences. An orange box highlights the first word at address 0, and an orange arrow points from the text box below to this word.

0	00100001
1	10101111
2	10000100
3	01111101
4	01010101
5	01100110
6	10000100
7	01111101
8	01100110
9	10000100

A Memória Principal

- A memória principal é organizada em **palavras de memória**
- Cada palavra de memória armazena uma sequência de bits
- Cada palavra de memória está associada a um identificador numérico conhecido como "endereço"

Memória Principal

0	00100001
1	10101111
2	10000100
3	01111101
4	01010101
5	01100110
6	10000100
7	01111101
8	01100110
9	10000100

Endereço da palavra de memória

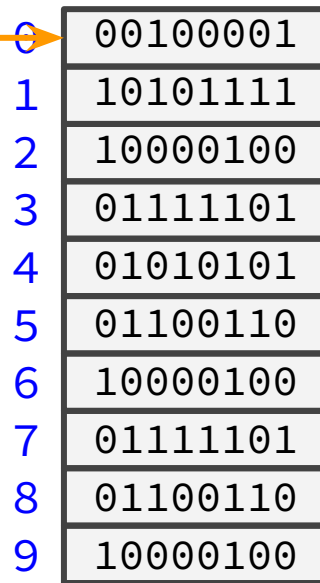
A Memória Principal

A memória principal é organizada em **palavras de memória**

- Memórias endereçáveis por byte são memórias onde cada palavra de memória armazena **um byte**.
- Este diagrama ilustra uma memória endereçável **por byte**

Palavras = 1 byte (i.e., 8 bits)

Memória Principal



The diagram illustrates the Main Memory structure. It consists of a vertical stack of 10 memory cells, indexed from 0 to 9. Each cell contains an 8-bit binary value. An orange line connects the text 'Palavras = 1 byte (i.e., 8 bits)' to the first cell (index 0), indicating that each word in memory is 1 byte (8 bits) long.

0	00100001
1	10101111
2	10000100
3	01111101
4	01010101
5	01100110
6	10000100
7	01111101
8	01100110
9	10000100

A Memória Principal

A memória principal é organizada em **palavras de memória**

- A palavra de memória define a Memory unidade básica de leitura e escrita na memória.
 - A CPU **não** consegue ler ou escrever apenas um **subconjunto dos bits** de uma palavra de memória;
 - Por outro lado, **múltiplas palavras** de memória **podem** ser lidas ou escritas pela CPU em uma única operação de leitura ou escrita!

Memória Principal

0	00100001
1	10101111
2	10000100
3	01111101
4	01010101
5	01100110
6	10000100
7	01111101
8	01100110
9	10000100

A Memória Principal


A memória principal é organizada em **palavras de memória**

- Um dado (p.ex. um número) ou uma instrução pode ocupar múltiplas palavras de memória!

```
...  
addi a0, a0, 1  
addi a1, a1, -1  
...
```

No RISC-V, cada instrução é codificada com **4 bytes** e ocupa **4 palavras** de memória.

Memória Principal



0	00100001
1	10101111
2	10000100
3	01111101
4	01010101
5	01100110
6	10000100
7	01111101
8	01100110
9	10000100

A Memória Principal

— — — —

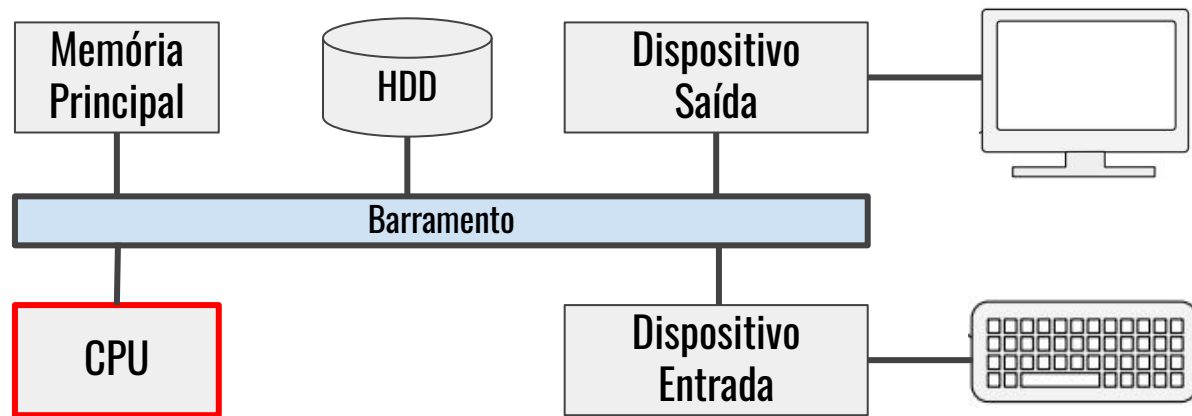
Resumo:

- Armazena dados e instruções como sequências de bits.
- Organizada em palavras de memória. Cada uma:
 - Armazena um conjunto de bits (p.ex.: 8 bits); e
 - É identificada por um endereço único.
- Dados e instruções podem ocupar múltiplas palavras de memória
 - P.ex: No RISC-V, uma instrução é codificada em 32 bits e ocupa 4 palavras de memória.

Componentes de um Computador

Um computador é geralmente composto pelos seguintes componentes:

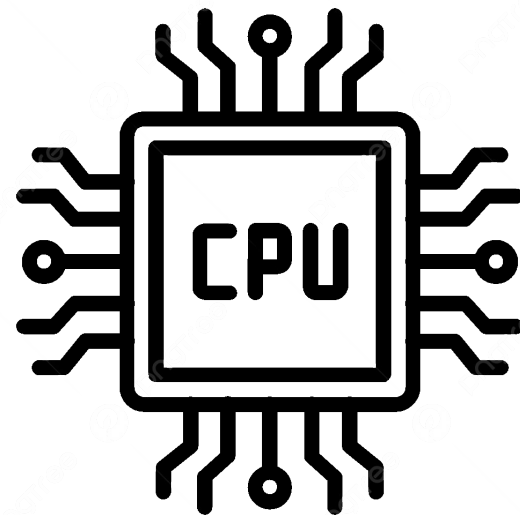
- Memória principal
- **CPU - Unidade de Processamento Central**
- Memória secundária
- Barramentos
- Periféricos



A Unidade Central de Processamento (CPU)

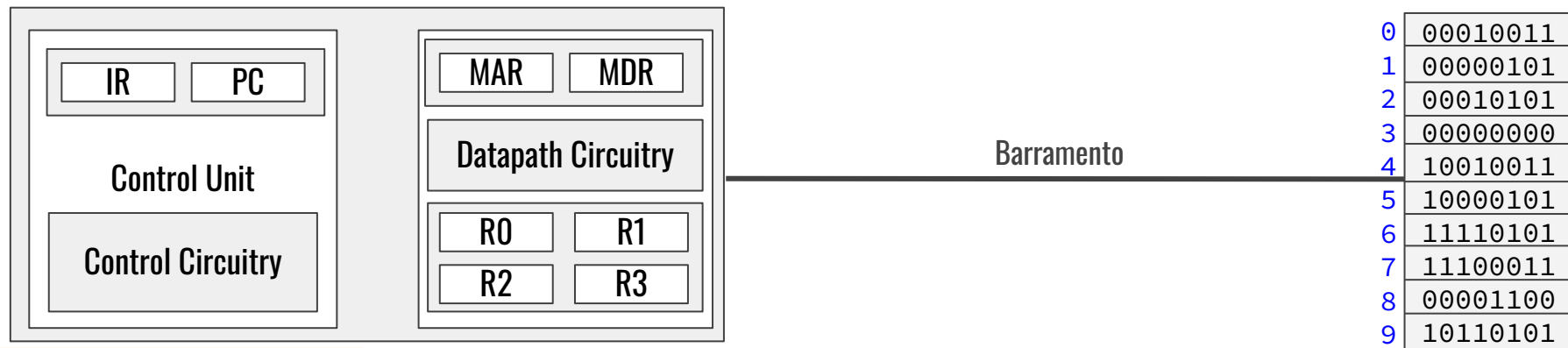
A CPU é responsável por executar os programas do computador.

- "Executar um programa" consiste em executar as instruções do programa!
- A CPU busca as instruções do programa da memória principal e as executa, uma a uma.
- Ao executar uma instrução, a CPU também pode ler ou escrever dados na memória principal!



A Unidade Central de Processamento (CPU)

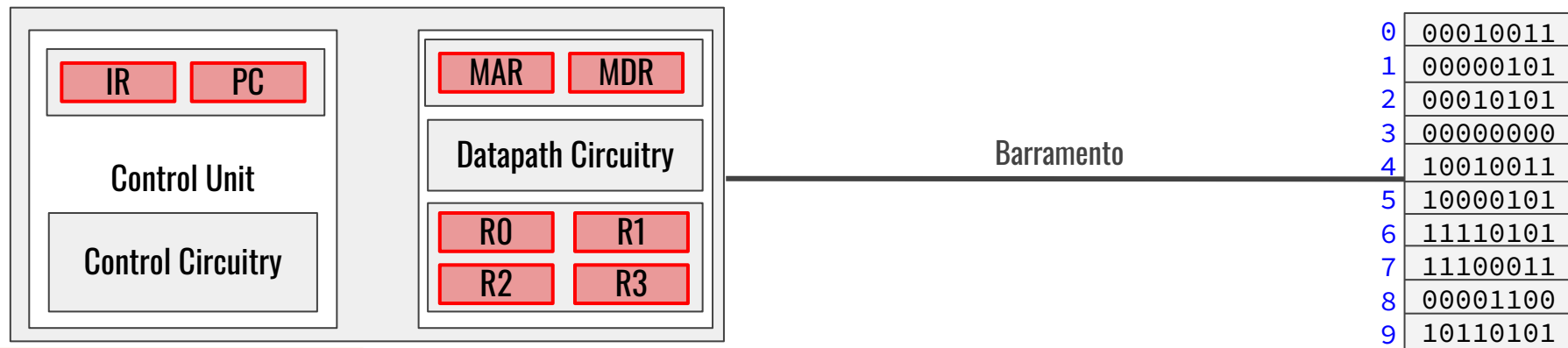
A CPU contém:



A Unidade Central de Processamento (CPU)

A CPU contém:

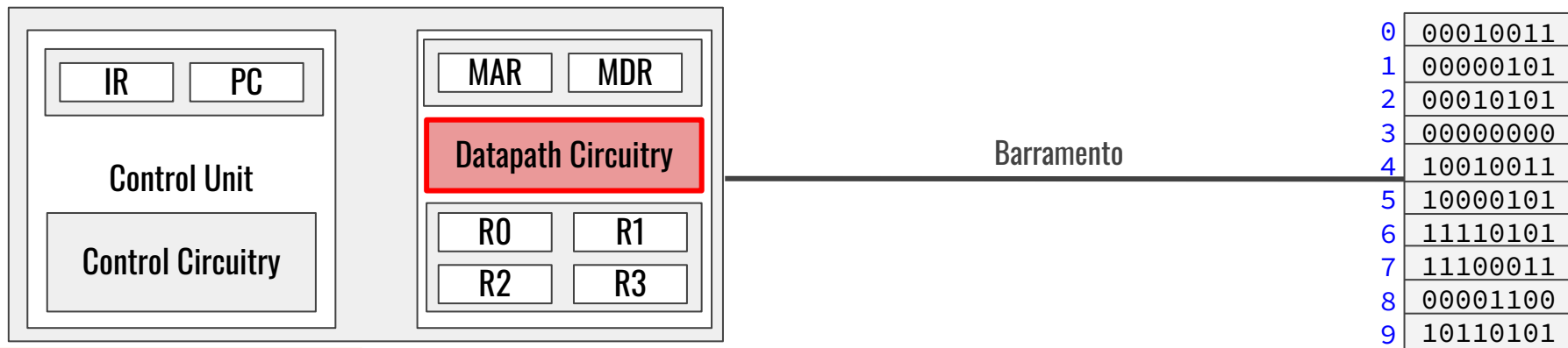
- Registradores: dispositivos de armazenamento de dados.
 - Propósito específico - Ex. RISC-V: PC (Program Counter)
 - Propósito geral - Ex. RISC-V: x1, x2, ..., x31



A Unidade Central de Processamento (CPU)

A CPU contém:

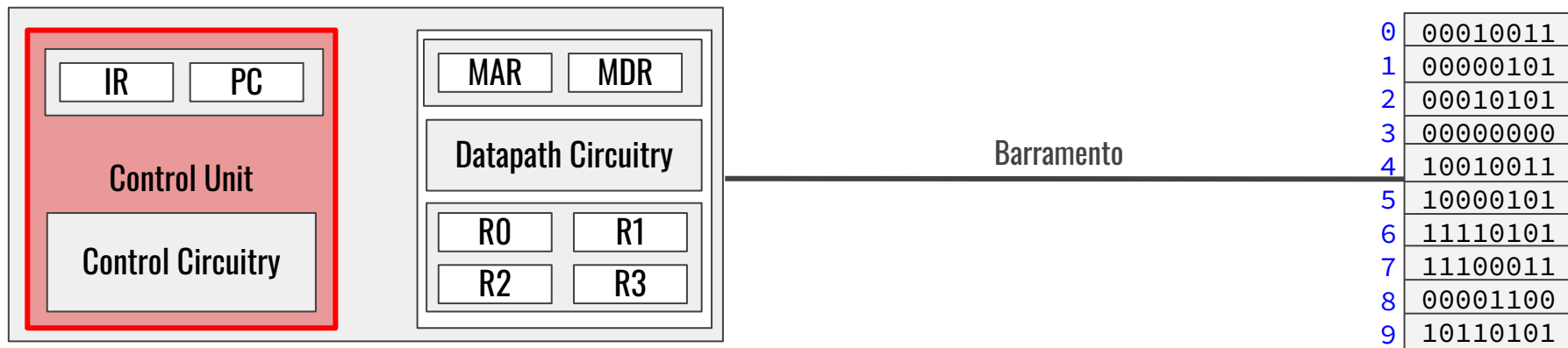
- **Uma via de dados (Datapath):** Realiza operações aritméticas e lógicas. As operações são geralmente realizadas com dados de registradores e o resultado armazenado em registradores.



A Unidade Central de Processamento (CPU)

A CPU contém:

- Uma unidade de controle (Control Unit): Orquestra o funcionamento do computador, enviando sinais de controle para os diversos componentes.



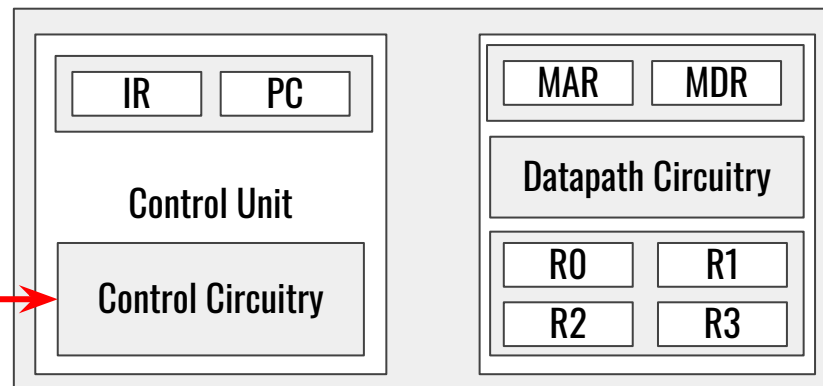
Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end
  
```

Algoritmo executado pelo circuito de controle



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

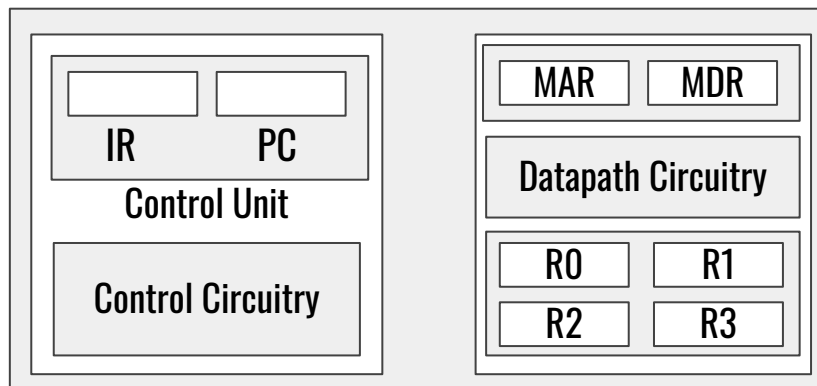
Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

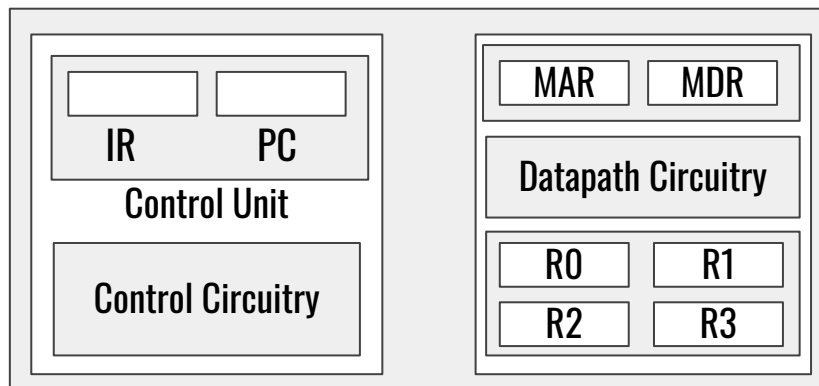
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 1: Buscar instrução a ser executada da memória principal!



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

```

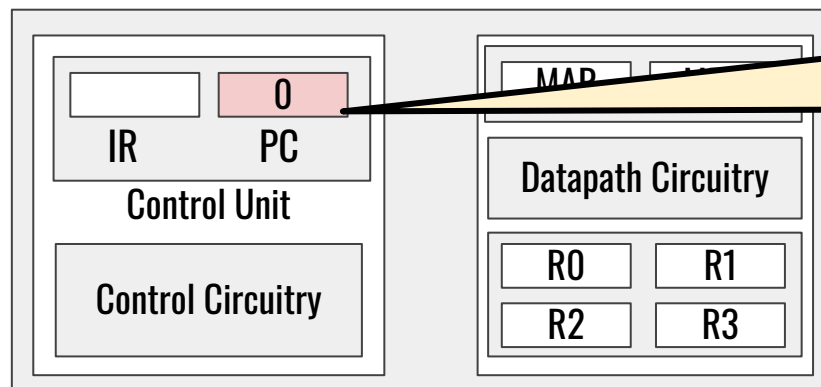
1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 1: Buscar instrução a ser executada da memória principal!

PC: Registrador que guarda o endereço de memória da próxima instrução que deve ser executada.

Exemplo: Endereço 0



0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

```

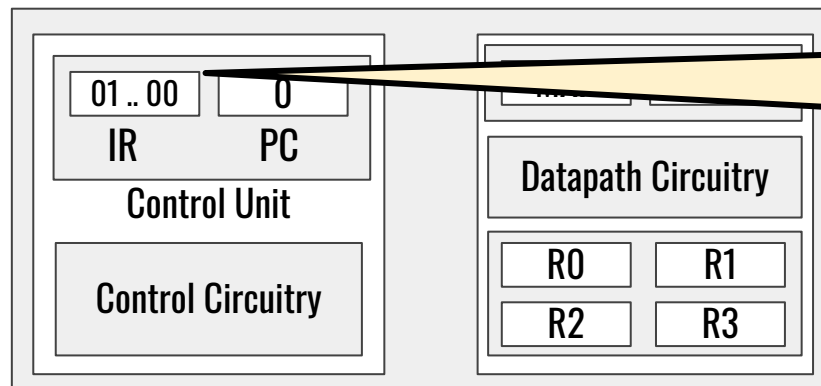
1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 1: Buscar instrução a ser executada da memória principal!

IR (Instruction Register) Registrador que guarda a instrução lida da memória

00000000 00010101 00000101 00010011



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

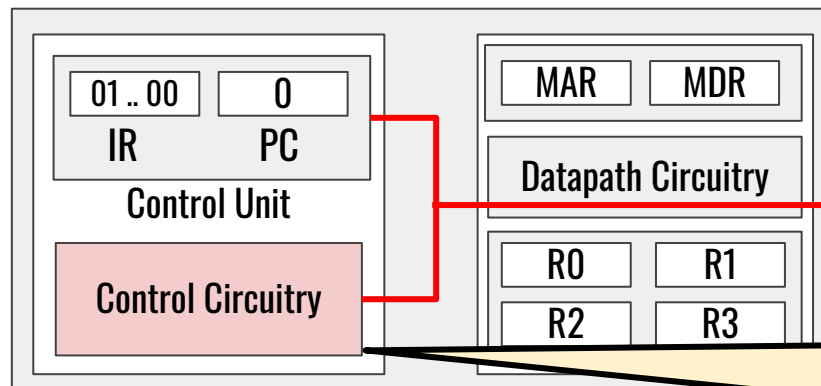
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 1: Buscar instrução a ser executada da memória principal!



Read - addr 0 Barramento

Circuito de controle envia comando de leitura para a memória principal com o endereço armazenado em PC

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

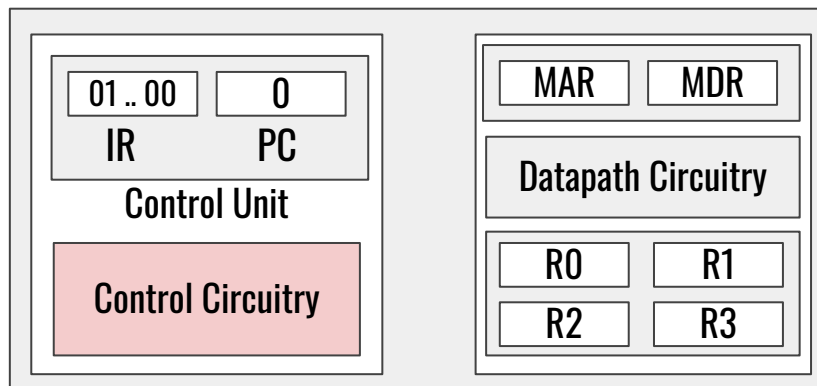
```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 1: Buscar instrução a ser executada da memória principal!

Memória principal lê 4 bytes (1 instrução) a partir do endereço solicitado e devolve à CPU



Barramento

data: 000 ... 0011

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

```

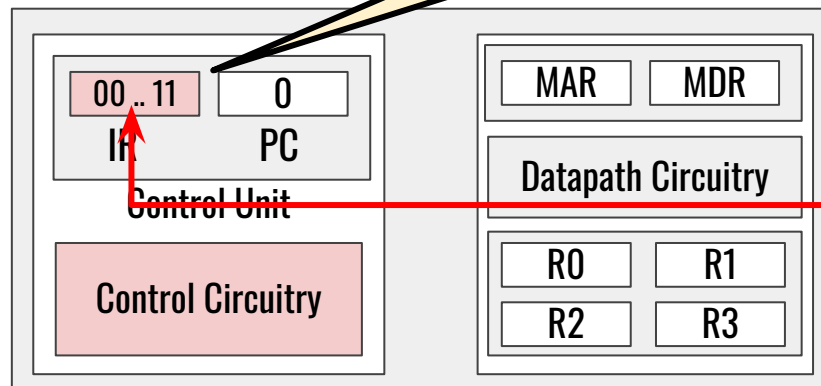
1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 1: Buscar instrução a ser executada da memória principal!

A CPU guarda a instrução em IR

Memória principal lê 4 bytes (1 instrução) a partir do endereço solicitado e devolve à CPU



Barramento

data: 000 ... 0011

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

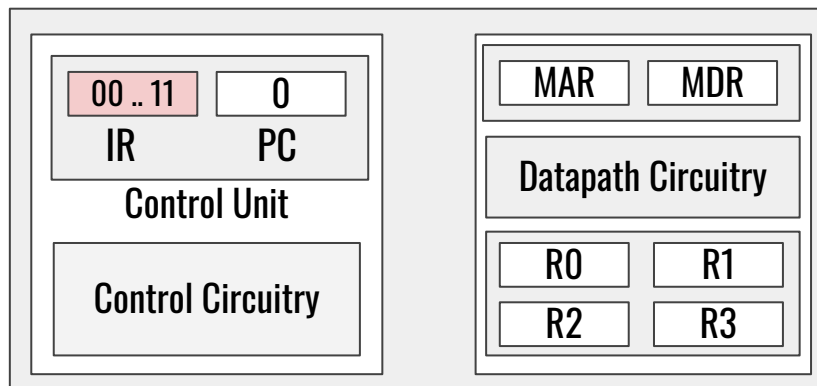
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 1: Buscar instrução a ser executada da memória principal!



0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

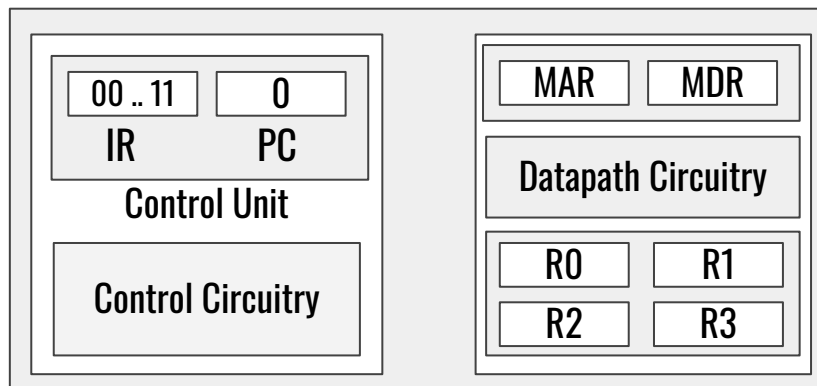
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC :
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 2: Atualizar o endereço da próxima instrução a ser executada!



0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

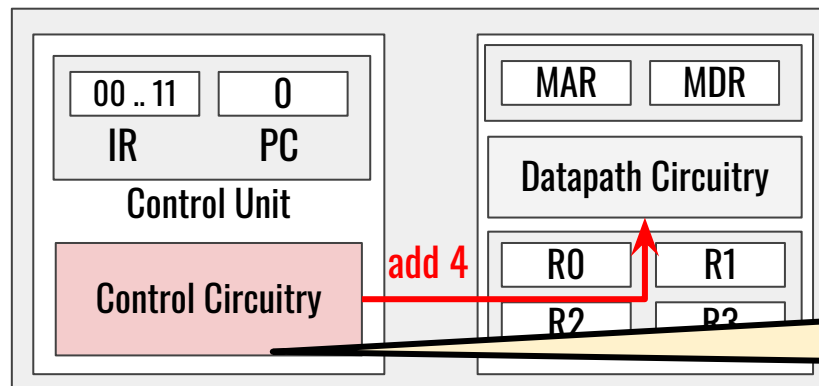
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC :
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 2: Atualizar o endereço da próxima instrução a ser executada!



Barramento

Circuito de controle envia comando para via de dados ler o valor de PC, somar 4 e gravar o resultado em PC

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

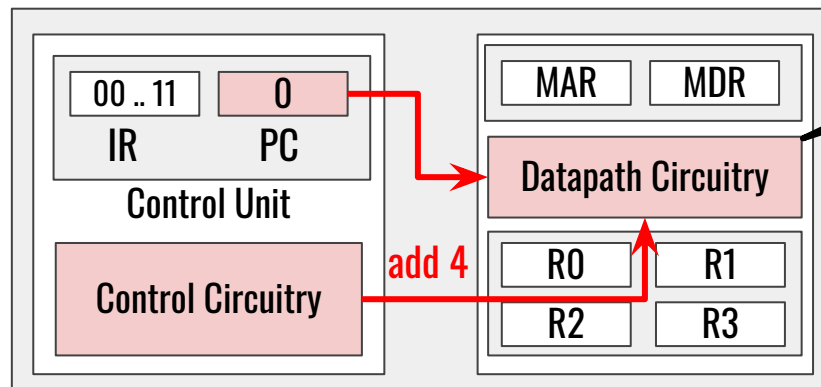
```

1 while True do
2   // Fetch instruction and update PC :
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 2: Atualizar o endereço da próxima instrução a ser executada!

A via de dados lê o valor de PC



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

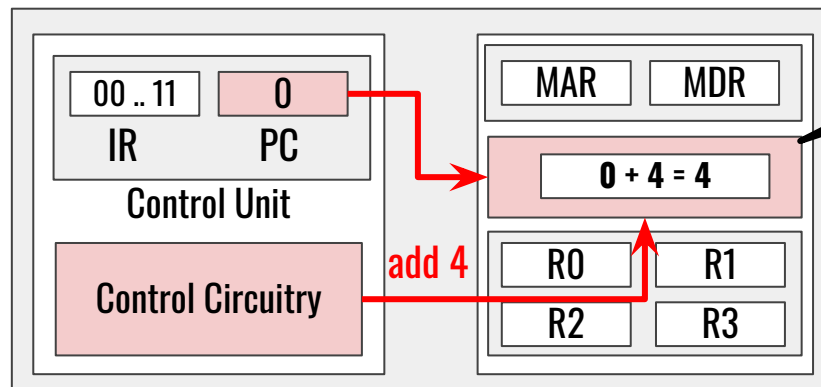
```

1 while True do
2   // Fetch instruction and update PC :
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 2: Atualizar o endereço da próxima instrução a ser executada!

A via de dados lê o valor de PC
Soma com 4



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

Algorithm 1: RV32I instructions execution cycle.

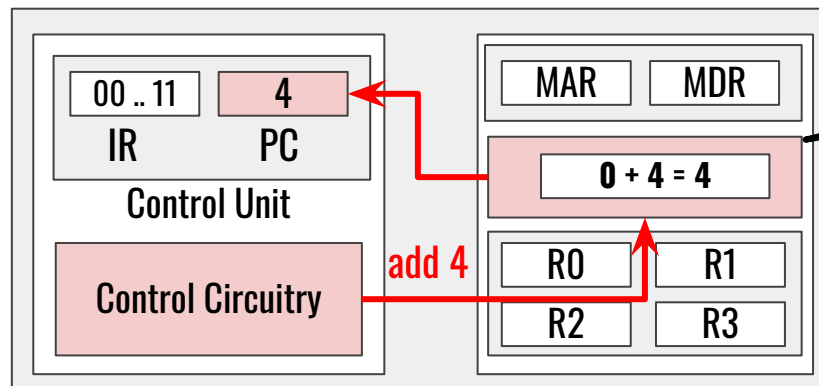
```

1 while True do
2   // Fetch instruction and update PC :
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 2: Atualizar o endereço da próxima instrução a ser executada!

A via de dados lê o valor de PC
Soma com 4
Escreve o resultado em PC



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

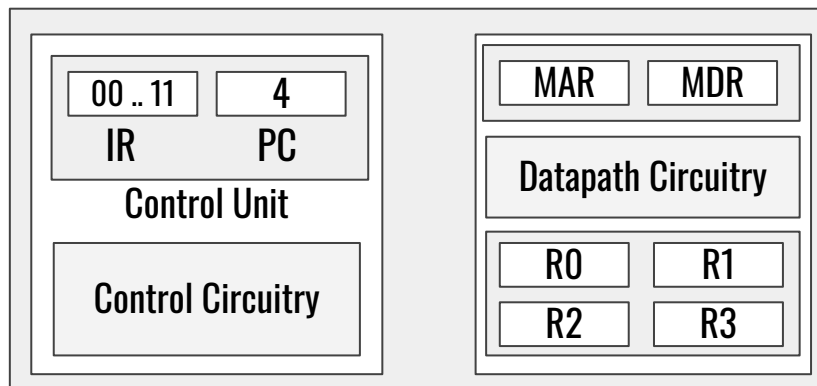
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC :
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 2: Atualizar o endereço da próxima instrução a ser executada!



0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

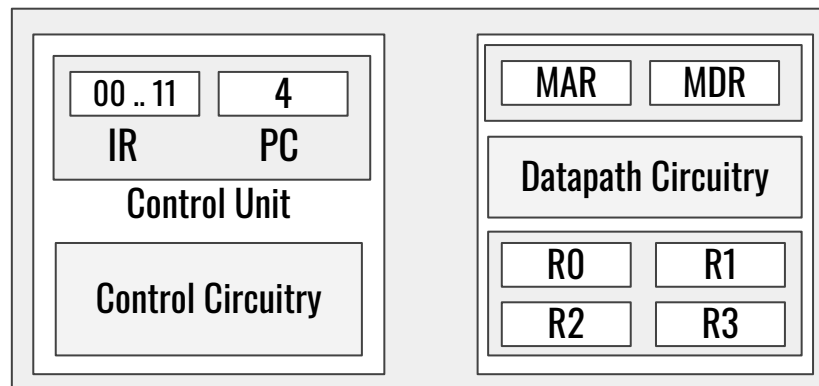
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 3: Executar a instrução armazenada em IR



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

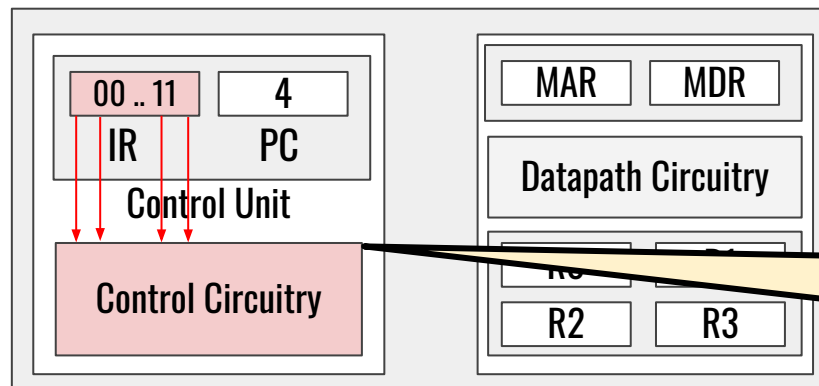
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 3: Executar a instrução armazenada em IR



A sequência de comandos (sinais) a serem enviados pelo circuito de controle depende da instrução armazenada no registrador IR.

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

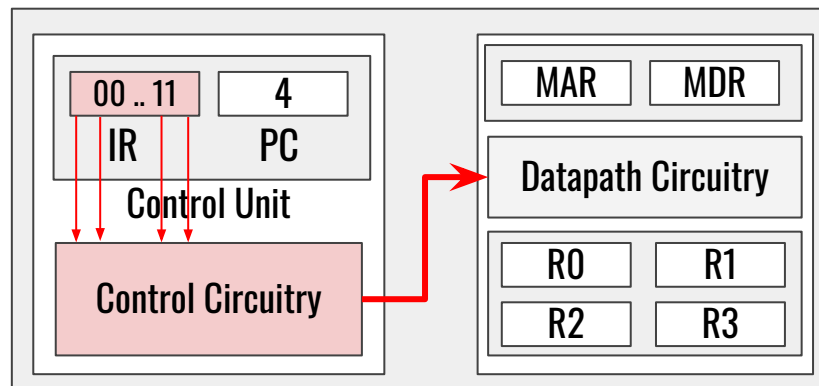
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 3: Executar a instrução armazenada em IR
add R3, R1, R2



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

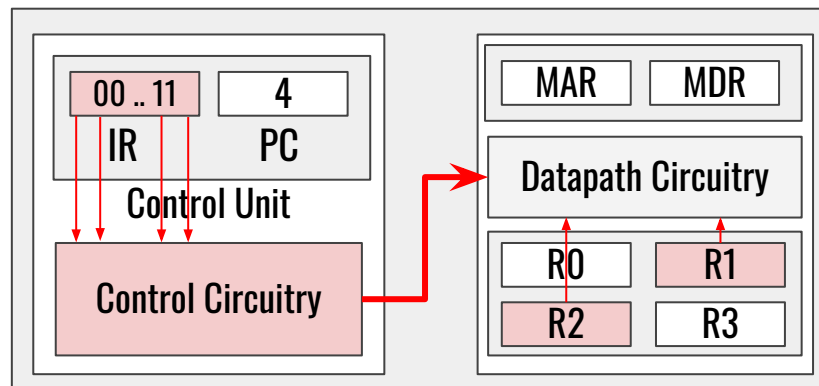
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 3: Executar a instrução armazenada em IR
add R3, R1, R2



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

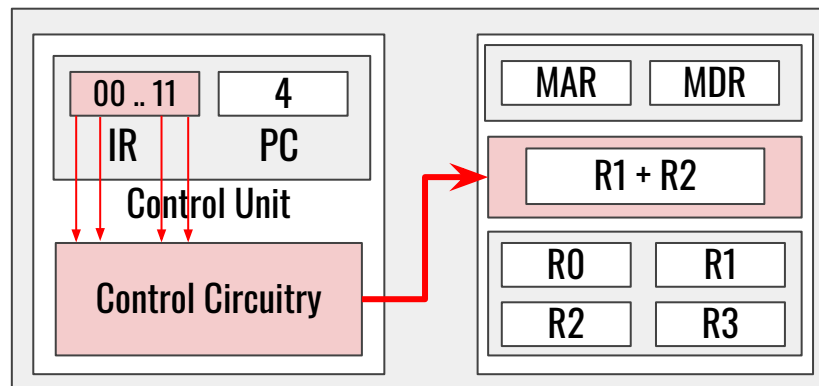
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 3: Executar a instrução armazenada em IR
add R3, R1, R2



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

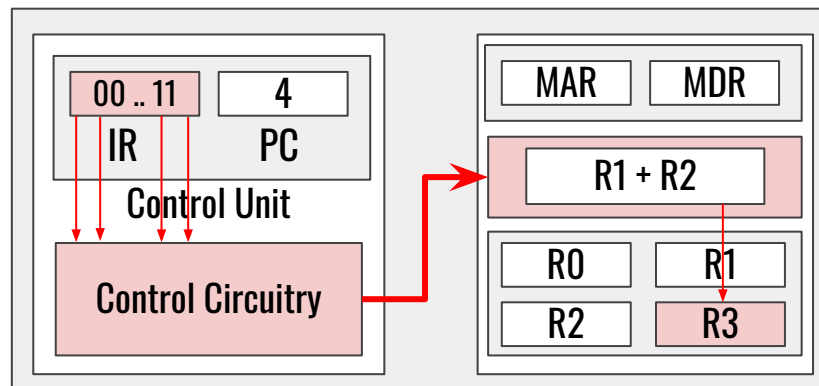
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 3: Executar a instrução armazenada em IR
add R3, R1, R2



Barramento

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

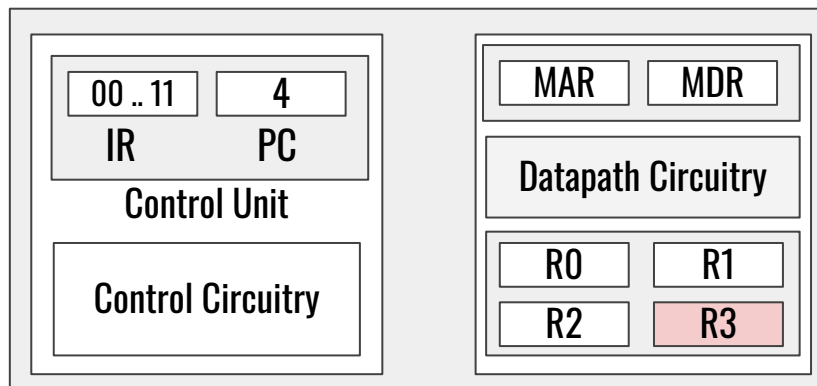
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC ;
3   IR ← MainMemory[PC] ;
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Passo 3: Executar a instrução armazenada em IR
add R3, R1, R2



0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

Executando Instruções

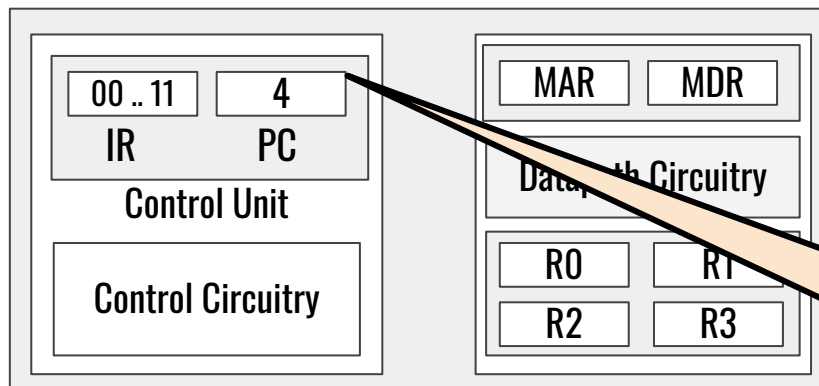
Algorithm 1: RV32I instructions execution cycle.

```

1 while True do
2   // Fetch instruction and update PC
3   IR ← MainMemory[PC]
4   PC ← PC+4;
5   ExecuteInstruction(IR);
6 end

```

Completamos o ciclo de execução de uma instrução - Retornamos ao passo 1 para executar a próxima instrução



Barramento

A próxima instrução será buscada da memória a partir do **endereço 4!**

0	00010011
1	00000101
2	00010101
3	00000000
4	10010011
5	10000101
6	11110101
7	11100011
8	00001100
9	10110101

A Unidade Central de Processamento (CPU)

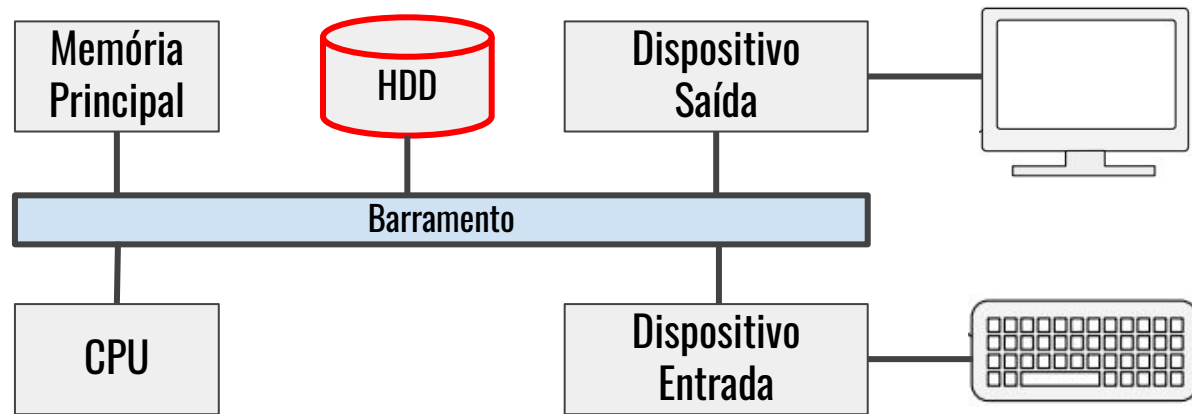
Resumo:

- Responsável por executar os programas do computador - Executa instruções uma a uma
- Possui **registradores** que servem para armazenar endereços, dados e instruções do programa
- Possui uma **unidade de controle** que orquestra a execução das instruções
- Possui uma **via de dados** que é capaz de realizar operações lógicas (and, or, xor, ...) e aritméticas (+, -, x, ...) com os dados e endereços.

Componentes de um Computador

Um computador é geralmente composto pelos seguintes componentes:

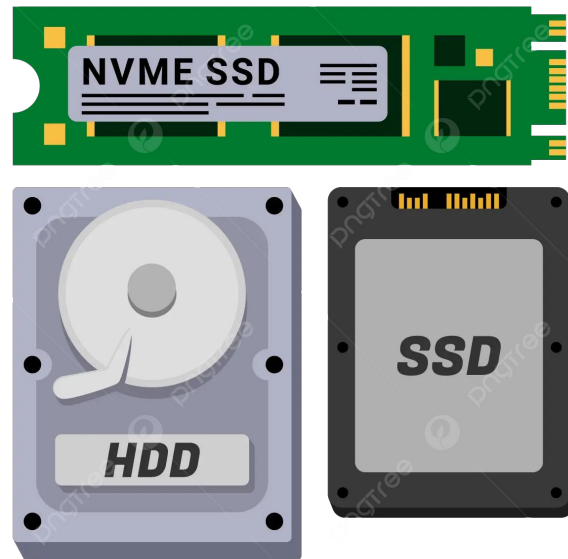
- Memória principal
- CPU – Unidade de Processamento Central
- **Memória secundária**
- Barramentos
- Periféricos



Memória secundária

A memória **principal** é geralmente **volátil**, i.e., seu conteúdo é perdido quando o sistema é desligado. A **memória secundária** é **persistente** e serve para armazenar os dados e programas de forma persistente.

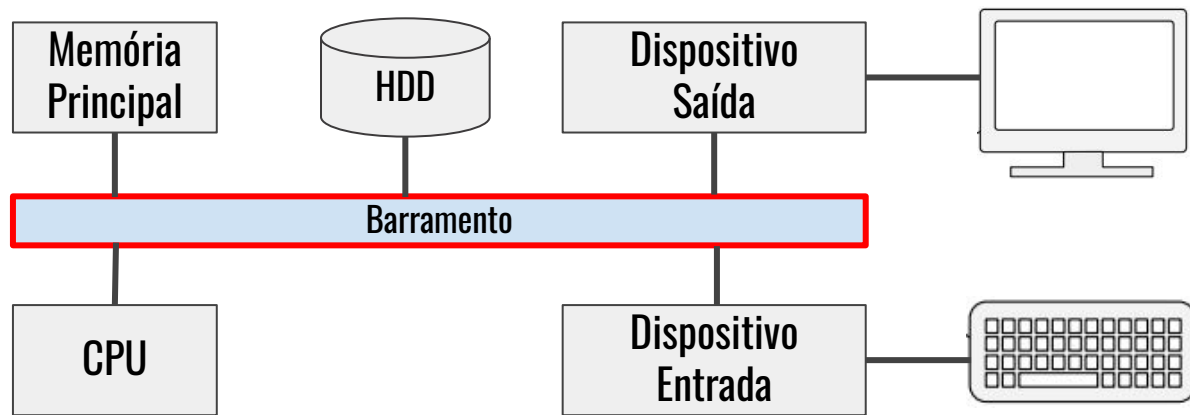
- Geralmente muito mais lenta que a memória principal;
- Programas/dados são carregados da memória secundária para a memória primária antes de serem executados/processados.



Componentes de um Computador

Um computador é geralmente composto pelos seguintes componentes:

- Memória principal
- CPU – Unidade de Processamento Central
- Memória secundária
- **Barramentos**
- Periféricos



Barramentos

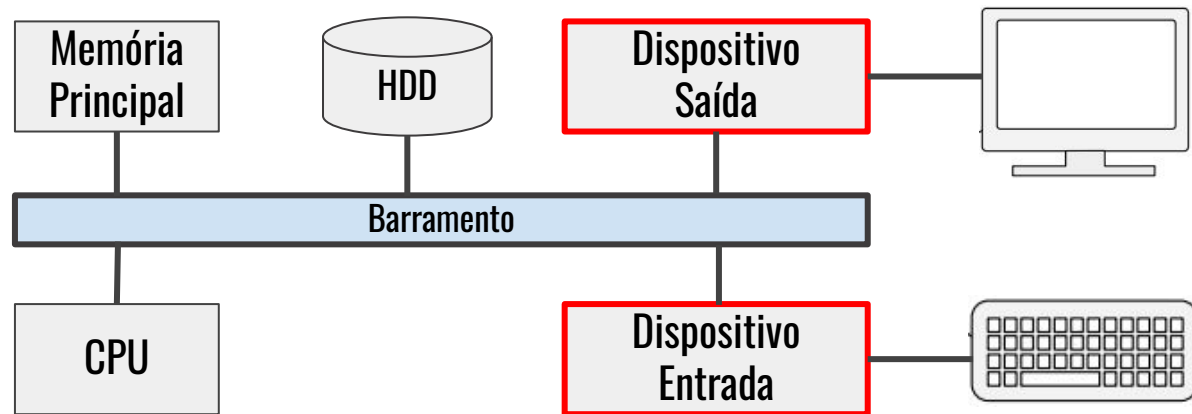
O barramento é um sistema de comunicação que transfere informação entre os componentes do computador (p.ex.: entre a memória e a CPU).

- Geralmente implementado com fios metálicos e circuitos associados que são responsáveis por transmitir a informação de forma elétrica.

Componentes de um Computador

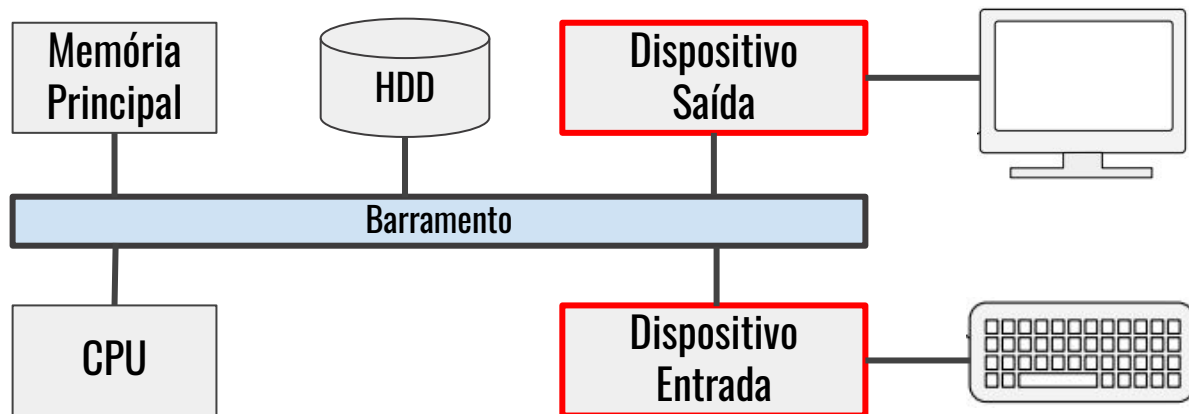
Um computador é geralmente composto pelos seguintes componentes:

- Memória principal
- CPU – Unidade de Processamento Central
- Memória secundária
- Barramentos
- **Periféricos**



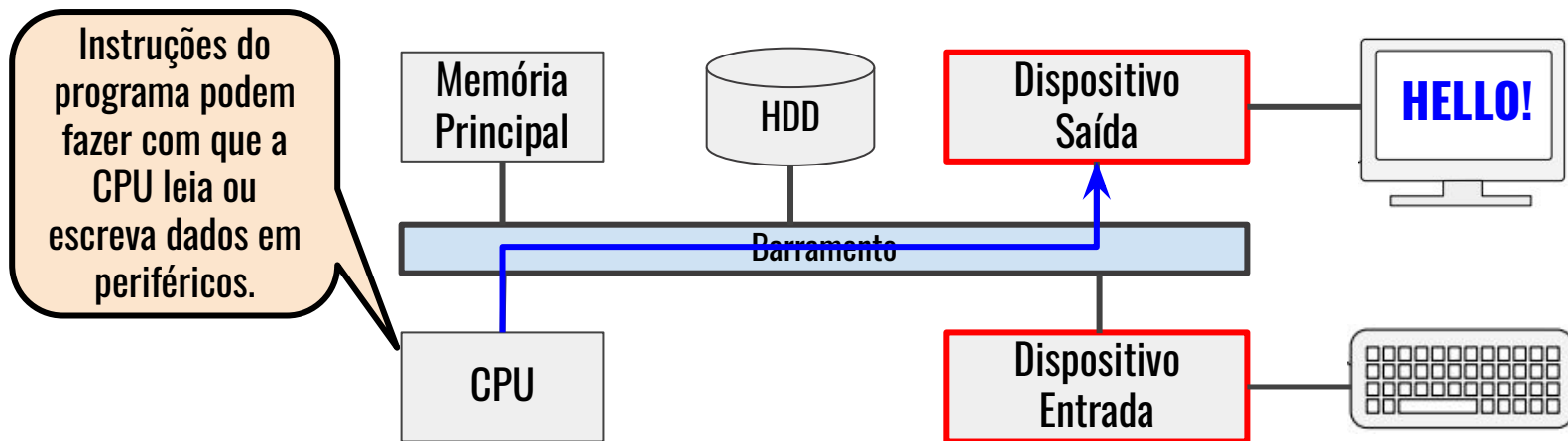
Periféricos

- Periféricos são dispositivos de entrada e saída (E/S) de dados e são conectados ao restante dos componentes através do barramento.



Periféricos

- Periféricos são dispositivos de entrada e saída (E/S) de dados e são conectados ao restante dos componentes através do barramento.



Codificação de programas de computador

Codificação de programas de computador

Existem diversas formas de se codificar programas de computadores:

- Código fonte
- Scripts
- Código binário para arquiteturas virtuais
- Código binário para outras arquiteturas
- Código nativo

Codificação de programas de computador

Existem diversas formas de se codificar programas de computadores:

- **Código fonte**

- Linguagem de alto nível
- Arquivos texto (sequência de caracteres) ◦ Devem ser transformados em outros formatos para serem executados.
- Compilado para formato executável.

```
#include<stdio.h>

int main(){
    printf("Hello World");
    return 0;
}
```

Codificação de programas de computador

Existem diversas formas de se codificar programas de computadores:

- **Scripts**

- Linguagem de alto nível
- Arquivos texto (sequência de caracteres)
- São executados por outros programas de computador
- Com o interpretador do Bash e/ou Python.

```
#!/bin/bash
```

```
echo "Hello World"
```

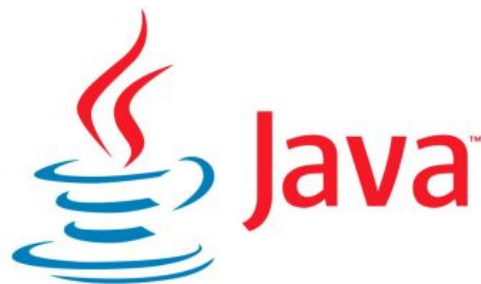
```
#!/bin/python
```

```
print("Hello World")
```

Codificação de programas de computador

Existem diversas formas de se codificar programas de computadores:

- **Código binário para arquiteturas virtuais**
 - Linguagem de máquina (Máquina virtual)
 - Arquivos binários ○ Sequência de instruções codificadas de forma binária!
 - São executados por outros programas de computador
 - P.ex: Máquina Virtual Java



Codificação de programas de computador

Existem diversas formas de se codificar programas de computadores:

- **Código nativo**

- Linguagem de máquina (CPU nativa)
- Arquivos binários
- Sequência de instruções nativas codificadas de forma binária!
- Instruções nativas são as instruções que a CPU do computador entende!
- Estes programa podem ser executados diretamente pela CPU!

Geração de programas nativos

Geração de programas nativos

Programas nativos são escritos em linguagem de máquina

- Geralmente gerados a partir de outros programas (em ling. de montagem ou de alto nível) com o auxílio de ferramentas (montador ou compilador).

Geração de programas nativos

Laços, variáveis, objetos, ...
Independente de máquina

Linguagem de baixo nível
Sequência de instruções,
registradores, posições de
memória

Dependente de máquina

Codificada de forma binária (0s
e 1s)

Dependente de máquina

Compilador
gcc, ...

```
int func(int a){  
    return a*113;  
}
```

Montador
as, ...

```
func:  
    slli a5,a0,3  
    sub a5,a5,a0
```

```
01010101  
10001001  
01010101
```

Execução de programas nativos

Execução de programas nativos

Para executar um programa nativo:

- Carrega-se o programa na memória principal;
- Grava-se no registrador PC o endereço de memória da primeira instrução do programa.

Estas tarefas são geralmente realizadas pelo sistema operacional

- A carga do programa é geralmente realizada por um módulo do SO chamado de Loader

Execução de programas nativos

- E quem carrega o SO na memória quando o computador é ligado?

Execução de programas nativos

- E quem carrega o SO na memória quando o computador é ligado?
- Geralmente é um programa que fica armazenado em uma memória não volátil (p.ex. BIOS) em um endereço fixo
 - Ao ser ligada, a CPU inicia o valor de PC com este endereço!
- Este programa procura por um **boot loader** e o carrega da memória secundária para a memória principal!
- **GRUB**

Informações Importantes

— — — —

- Acesso a memória **É LENTO!**
- Diversos **níveis de hierarquia** são criados para otimizar o processamento da CPU (Cache L1, L2 e L3)
- O processo de execução na CPU é feito em **pipelining**
 - **IF -> ID -> EX -> MEM -> WB**
- Conteúdo de **MC732**