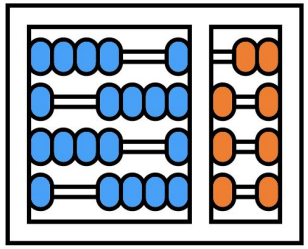


Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

Organização Básica de computadores e linguagem de montagem

Execução de Programas em Computadores

Prof. Edson Borin

https://www.ic.unicamp.br/~edson

Institute of Computing - UNICAMP

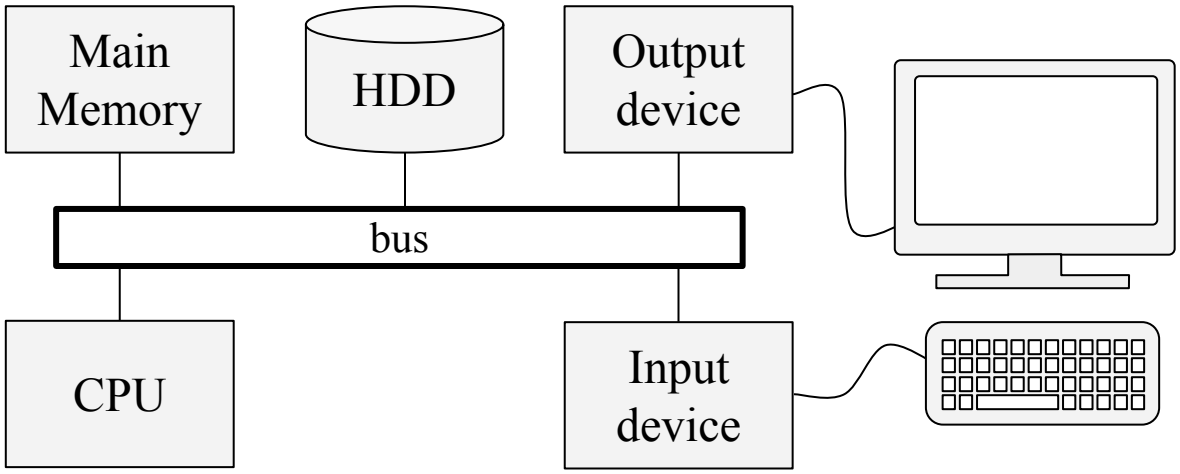
Agenda

- Componentes de um computador
- 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:

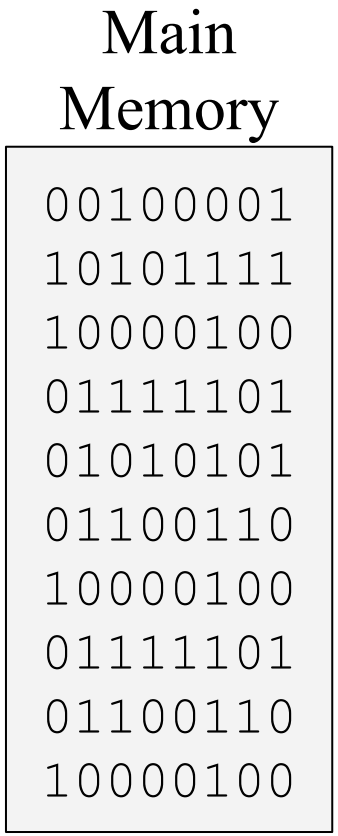
- Memória principal
- CPU - Unidade Central de Processamento
- Memória secundária (persistente)
- Barramento
- Periféricos



A memória principal

A memória principal armazena as instruções e dados de programas que estão sendo executados.

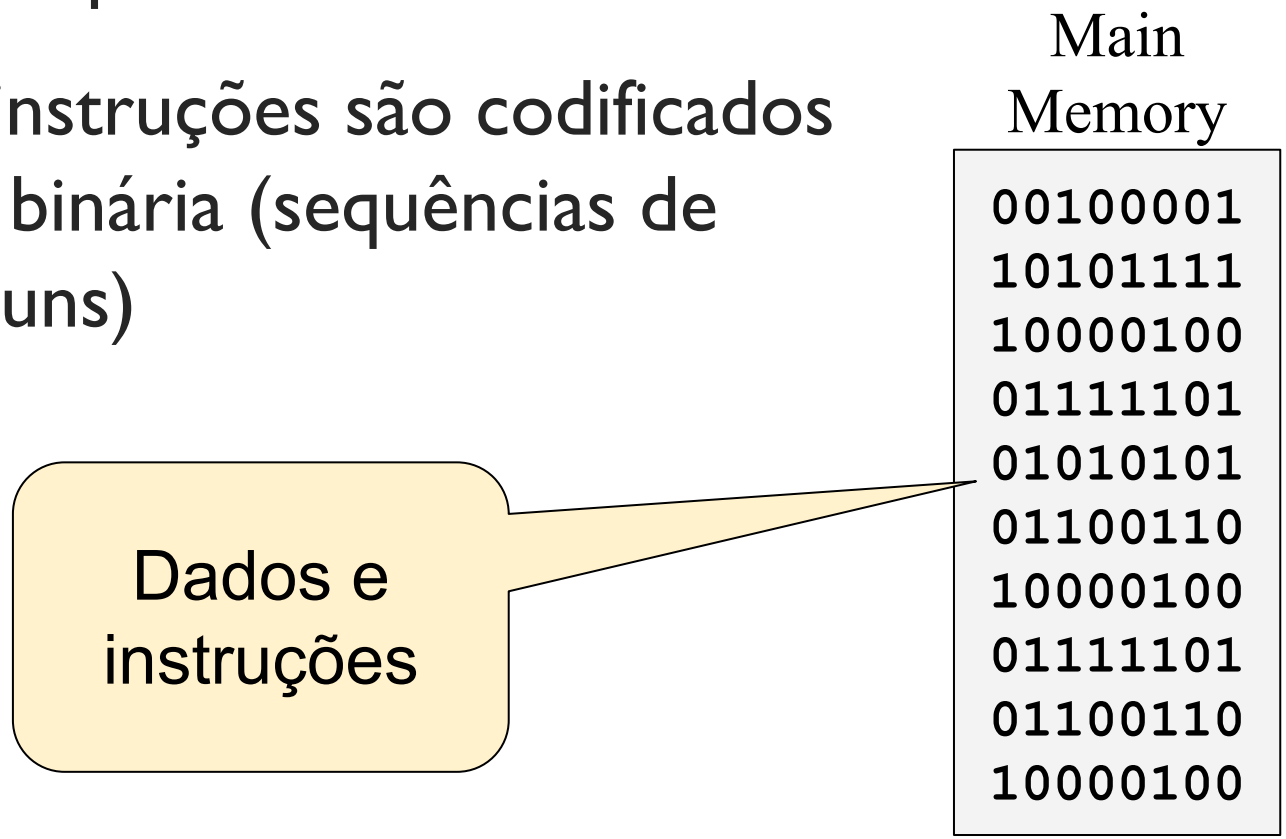
- Dados e instruções são codificados de forma binária (sequências de zeros ou uns)



A memória principal

A memória principal armazena as instruções e dados de programas que estão sendo executados.

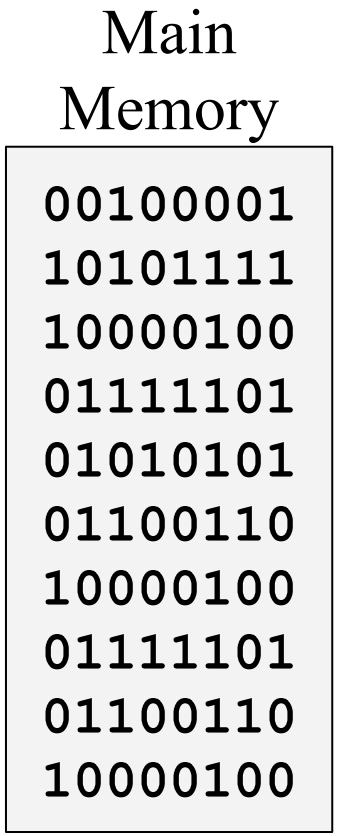
- Dados e instruções são codificados de forma binária (sequências de zeros ou uns)



A memória principal

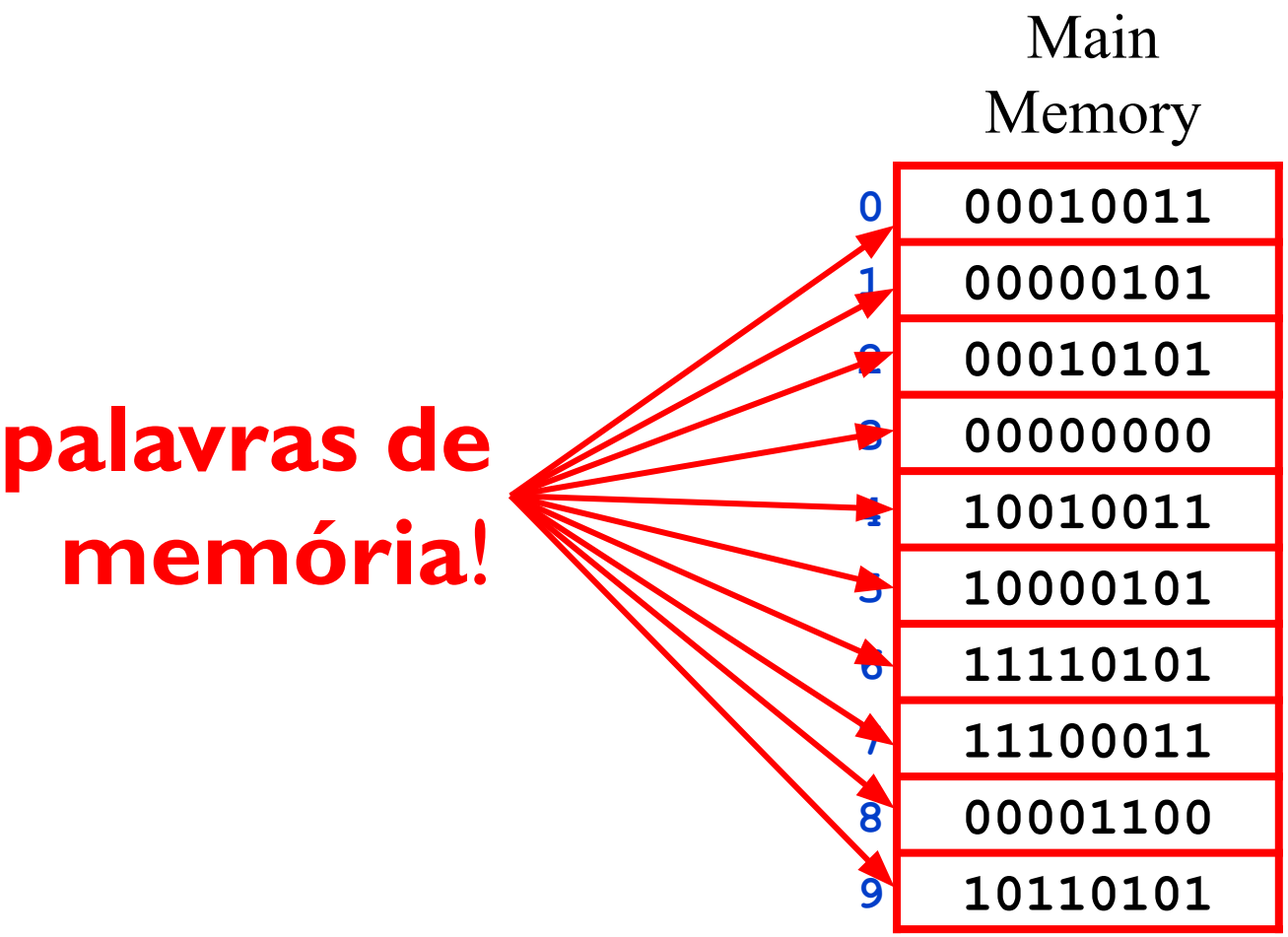
A memória principal armazena as instruções e dados de programas que estão sendo executados.

- Dados e instruções são codificados de forma binária (sequências de zeros ou uns)
- A Unidade Central de Processamento (CPU) lê e escreve dados e instruções da memória principal!



A memória principal

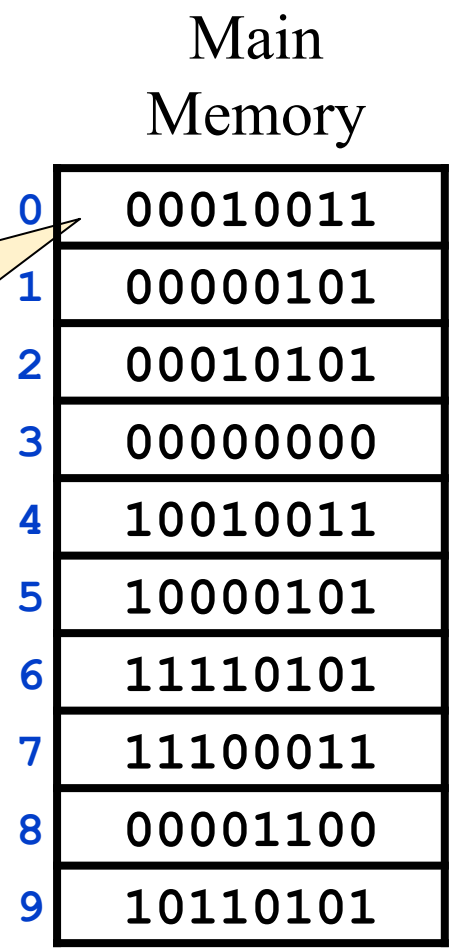
A memória principal é organizada em **palavras de memória**!



A memória principal

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

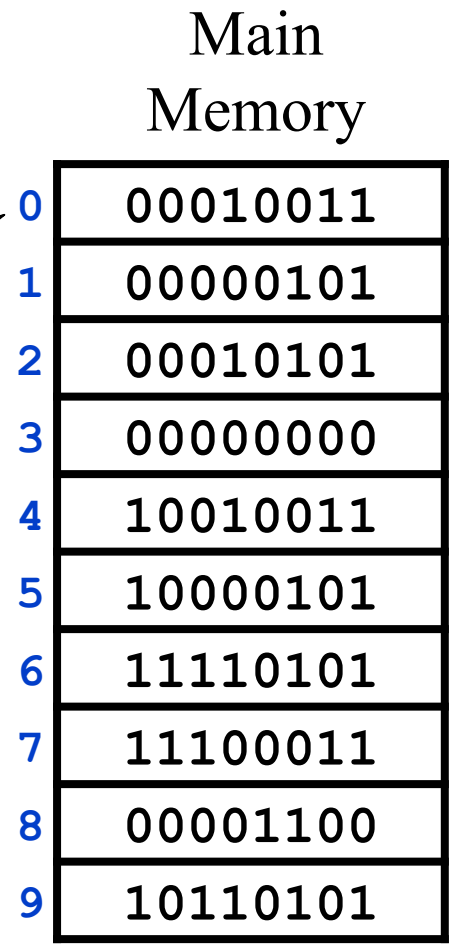
Cada palavra de memória armazena uma sequência de *bits*.
Neste exemplo, a primeira palavra de memória armazena a sequência `00010011`



A memória principal

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

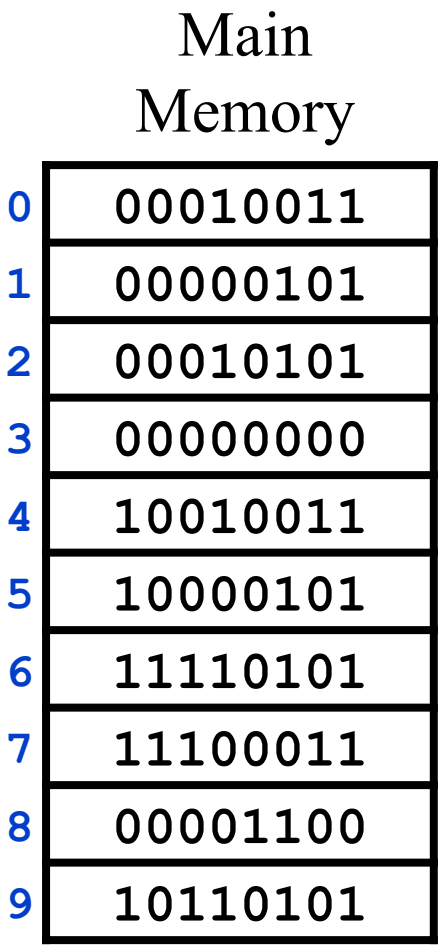
Cada palavra de memória está associada a um identificador numérico conhecido como "**endereço**" da palavra de memória.



A memória principal

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

- Memórias endereçáveis por *byte* (*Byte-addressable memories*) são memórias onde cada palavra de memória armazena um *byte*.

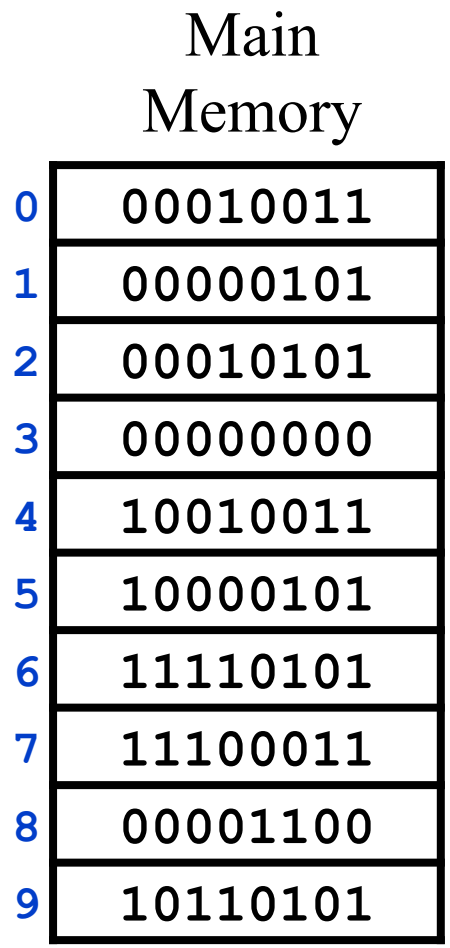


A memória principal

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

- Memórias endereçáveis por *byte* (*Byte-addressable memories*) são memórias onde cada palavra de memória armazena um *byte*.

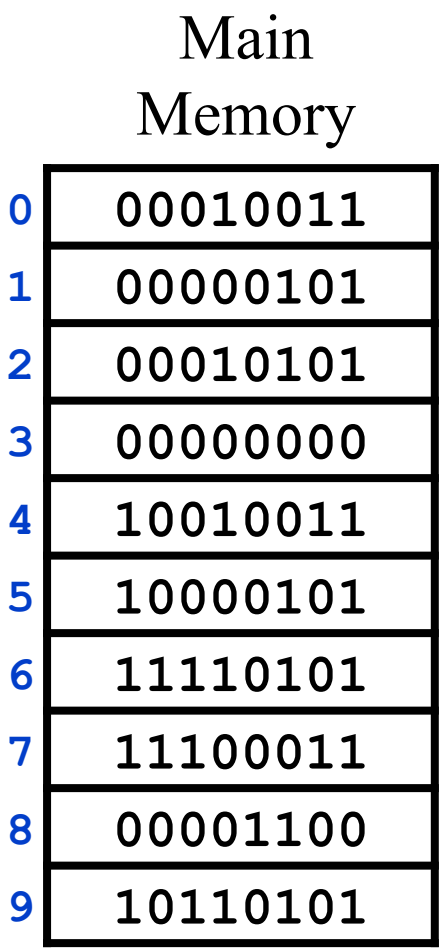
Este diagrama ilustra uma memória endereçável por *byte* - note que cada palavra de memória armazena um *byte* (*i.e.*, 8 *bits*).



A memória principal

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

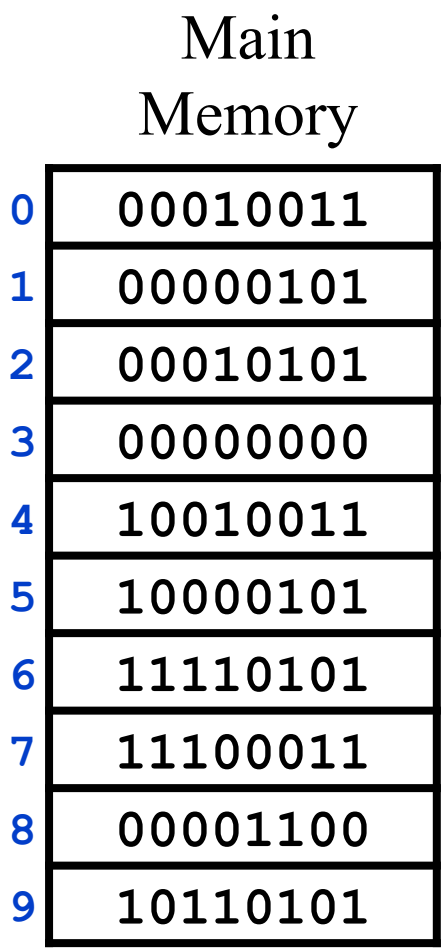
- A palavra de memória define a unidade básica de leitura e escrita na memória.
  - A CPU não consegue ler ou escrever apenas um subconjunto dos *bits* de uma palavra de memória;



A memória principal

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

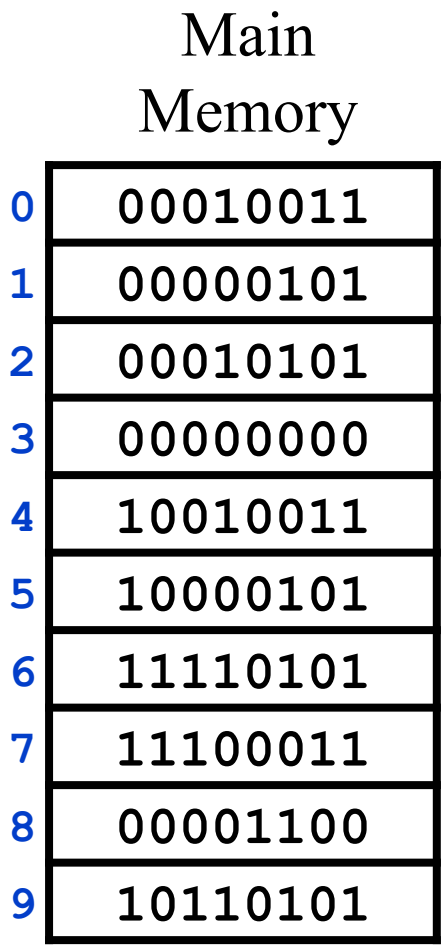
- A palavra de memória define a unidade básica de leitura e escrita na memória.
  - A CPU não consegue ler ou escrever apenas um subconjunto dos *bits* de uma palavra de memória;
  - Por outro lado, múltiplas palavras de memória podem ser lidas ou escritas pela CPU em uma única operação de leitura ou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



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!



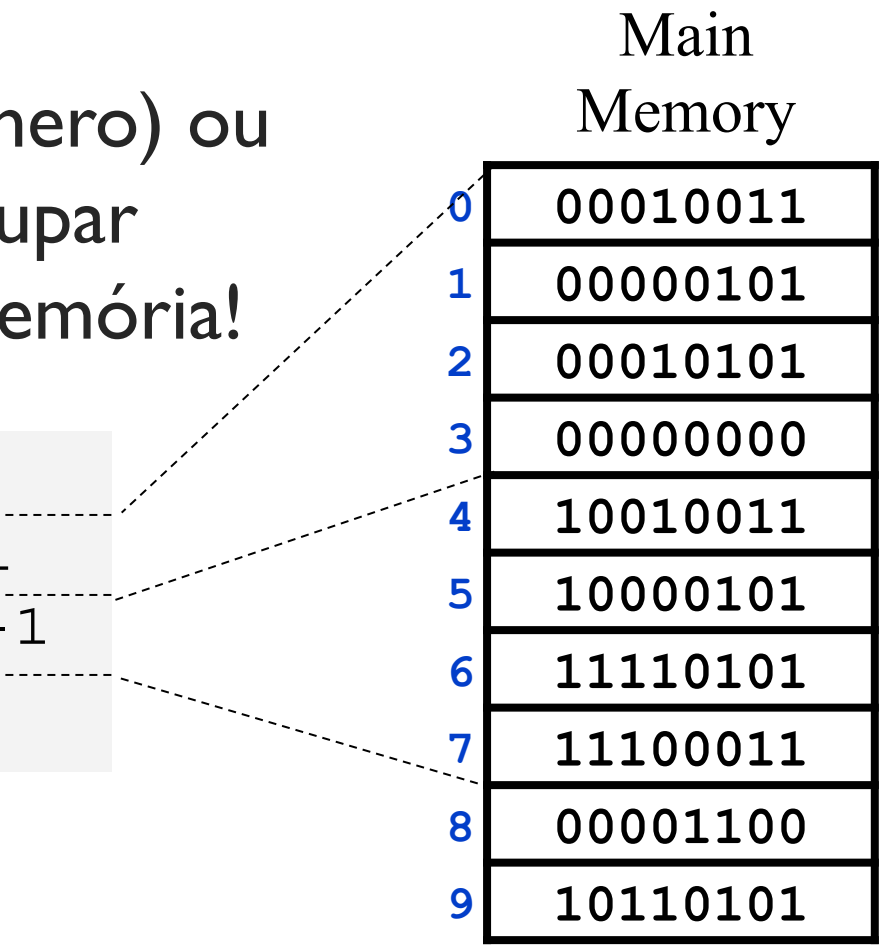
No RISC-V, cada instrução é codificada com 4 *bytes* e ocupa 4 palavras de memória.

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
   ...
```



A memória principal

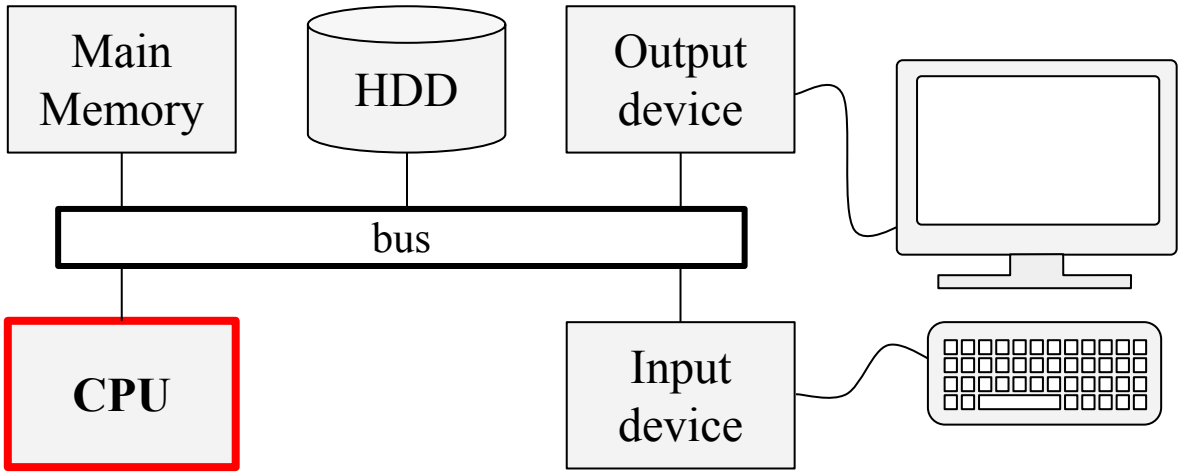
Resumo:

- 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 Central de Processamento**
- Memória secundária (persistente)
- Barramento
- Periféricos



A Unidade Central de Processamento (CPU)

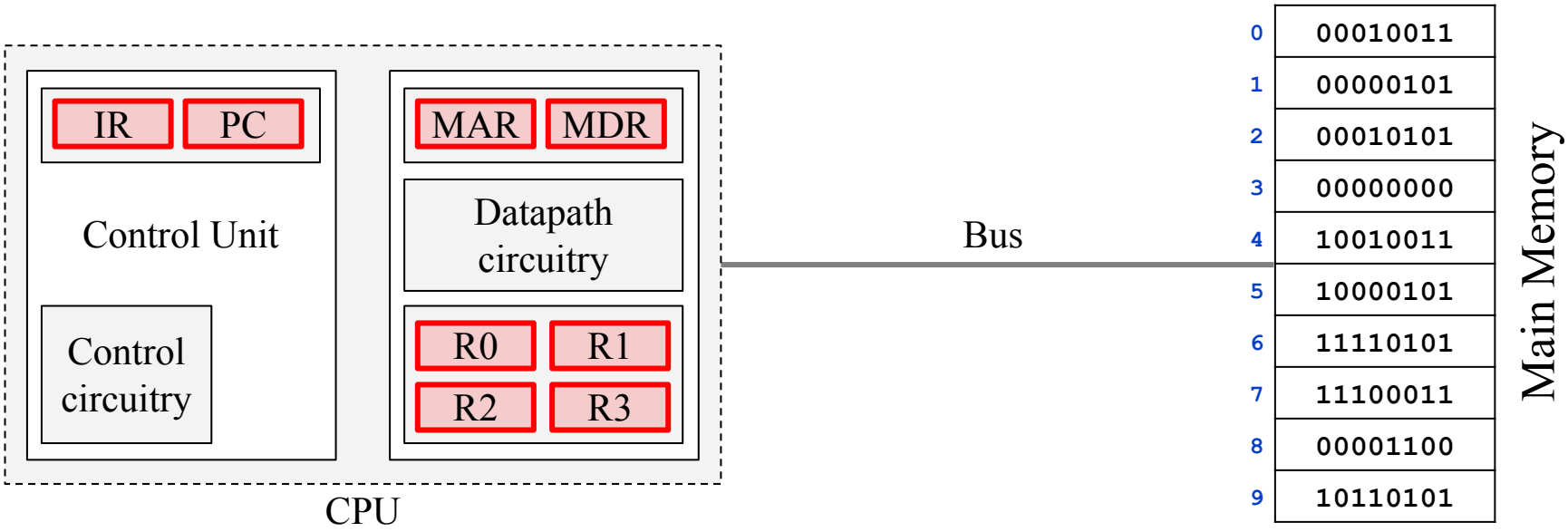
A CPU é responsável por executar os programas do computador.

- "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:

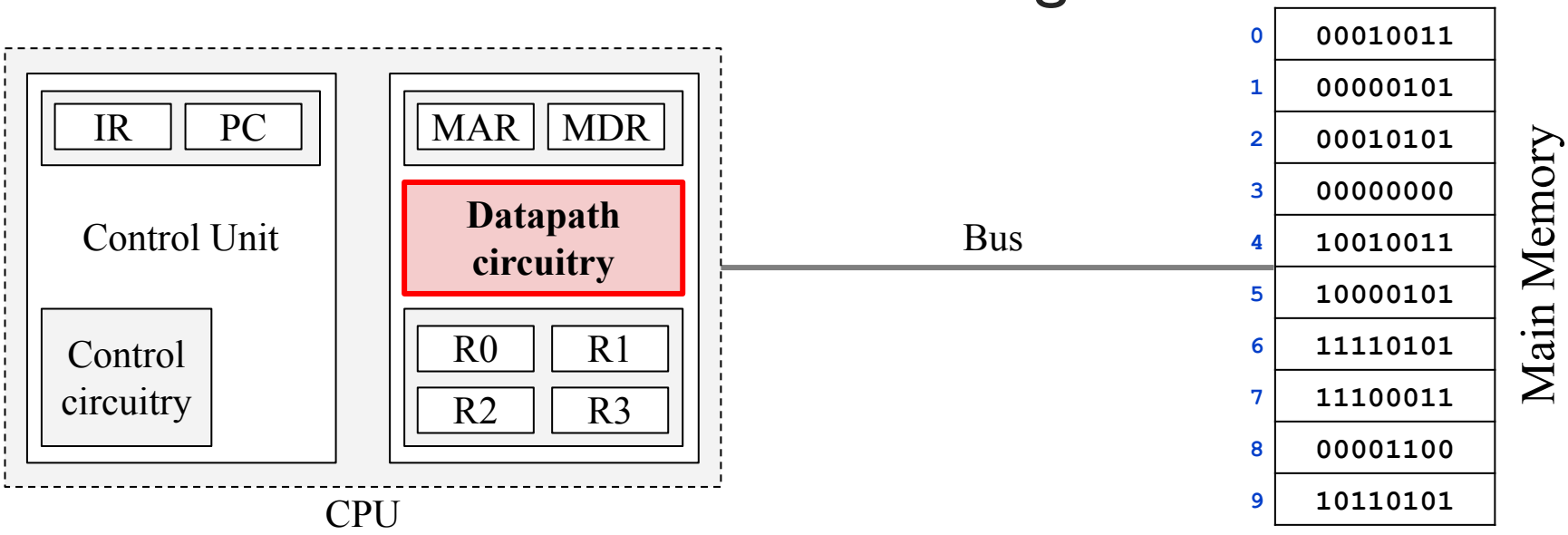
- **Registradores**: dispositivos de armazenamento de dados.
  - Propósito específico - Ex. RISC-V: PC (Program Counter)
  - Propósito geral - Ex. RISC-V: x1, x2, …, x31



A Unidade Central de Processamento (CPU)

A CPU contém:

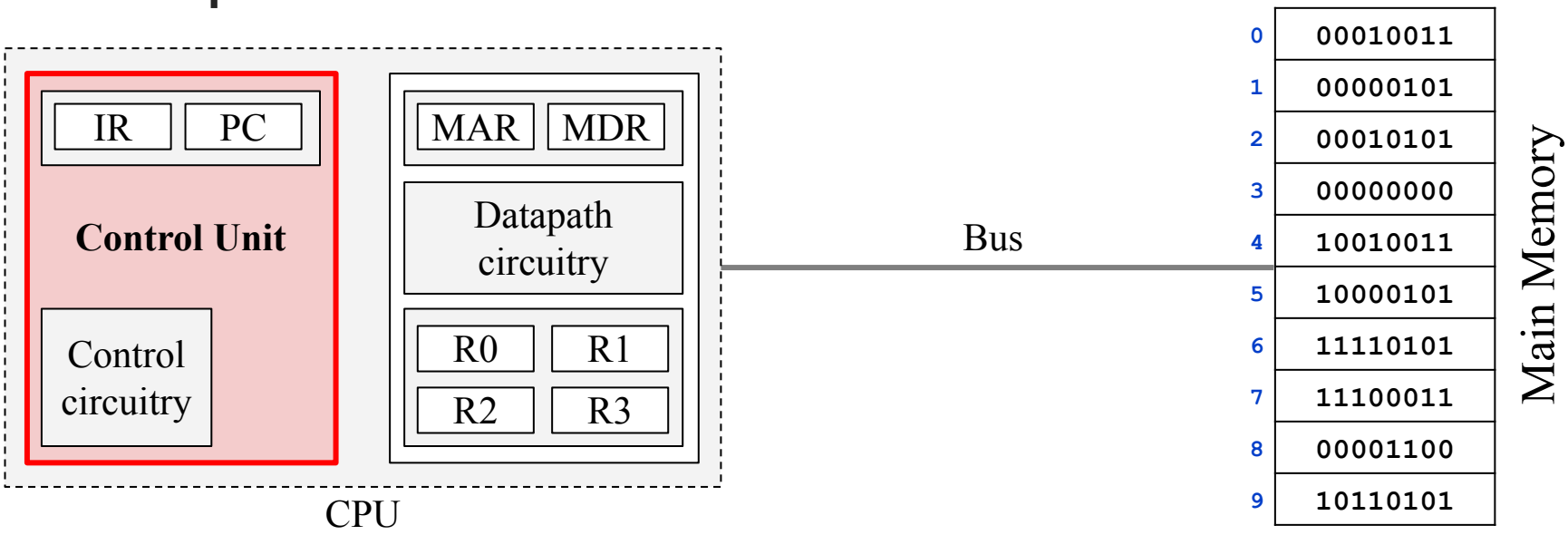
- **Uma via de dados (Datapath)**: Realiza operações aritméticas e lógicas. As operações são geralmente realizadas com dados de registradores e o resultado armazenado em registradores.



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.

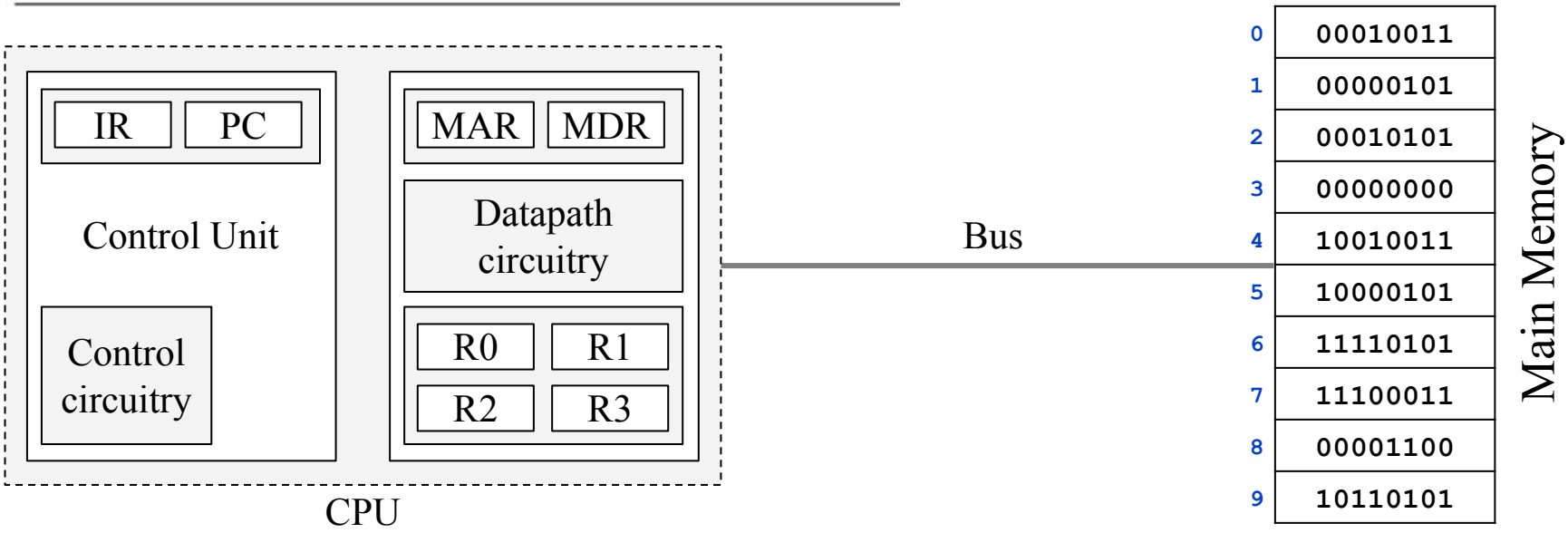


A Unidade Central de Processamento (CPU)

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
```

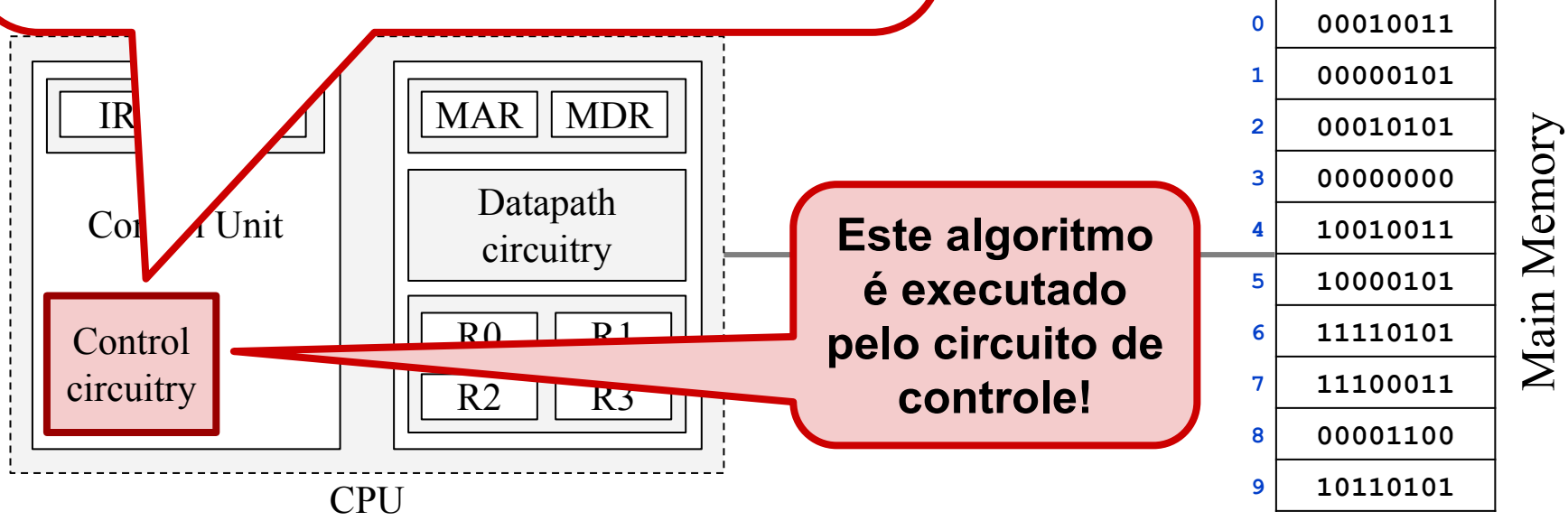


A Unidade Central de Processamento (CPU)

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
```

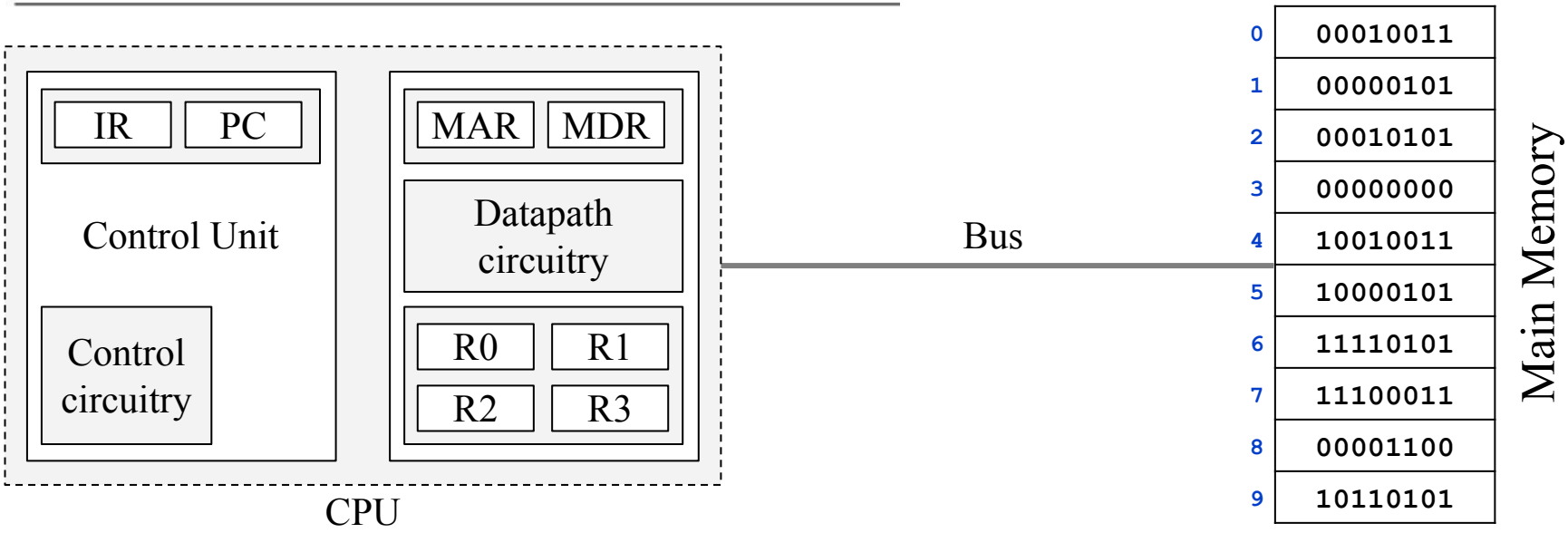


A Unidade Central de Processamento (CPU)

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
```



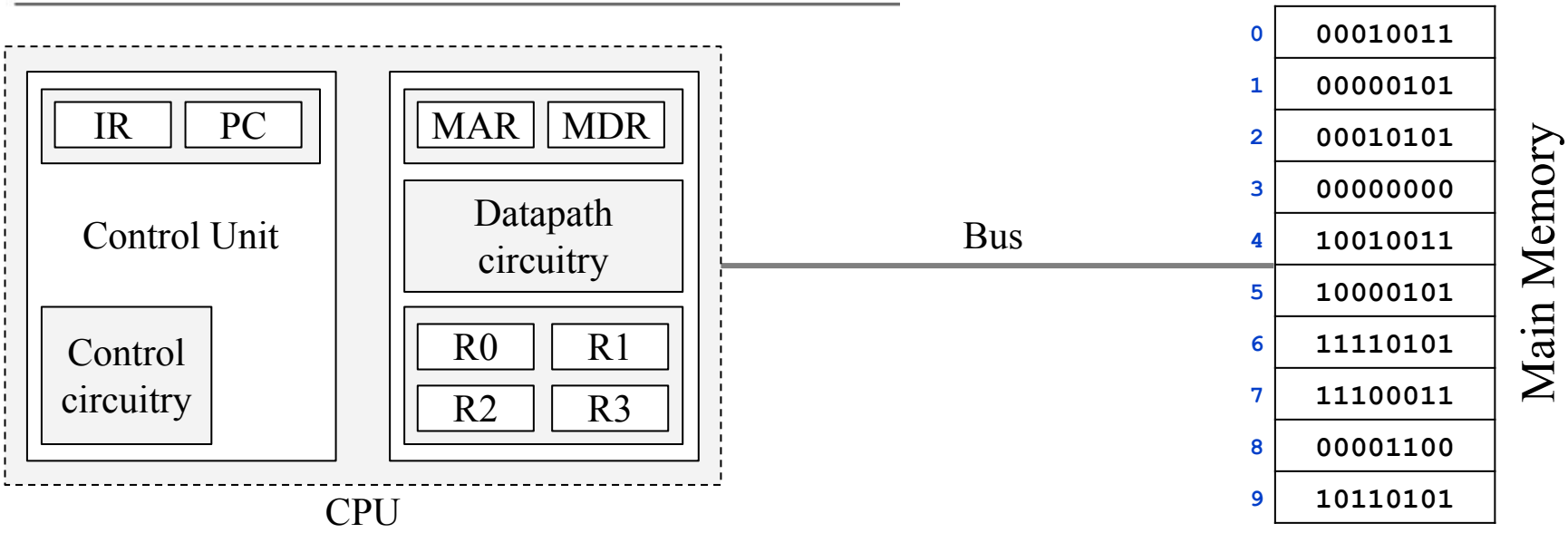
A Unidade Central de Processamento (CPU)

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!



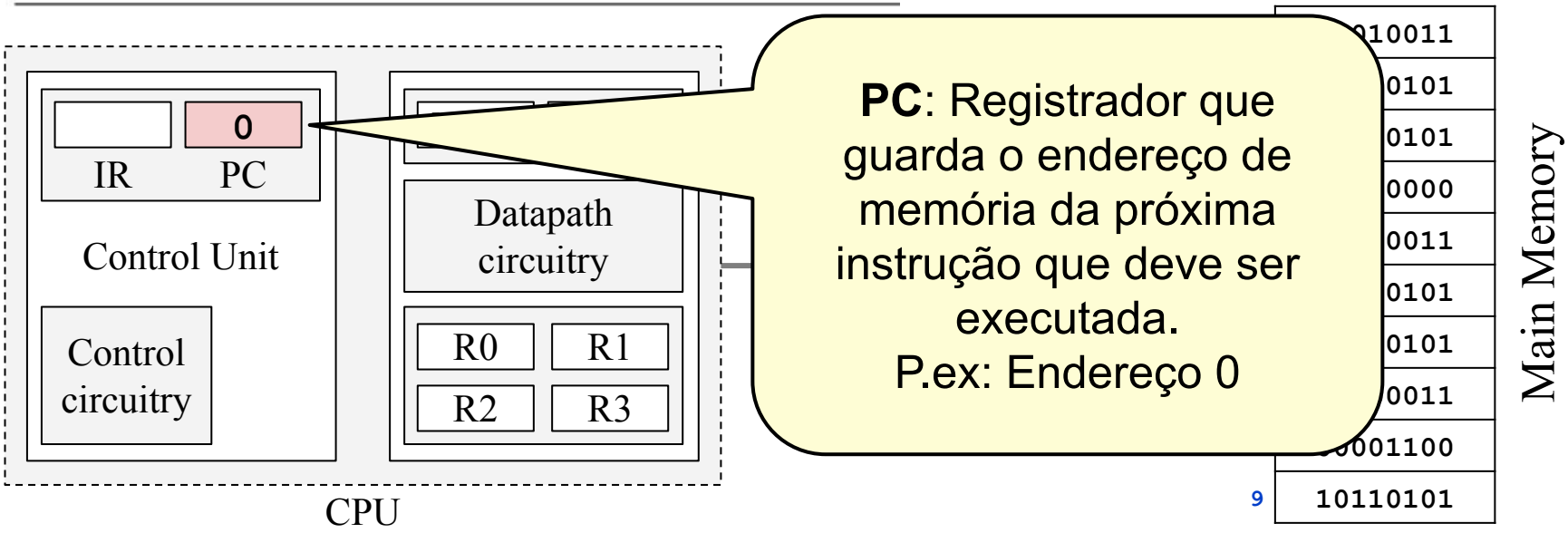
A Unidade Central de Processamento (CPU)

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.
P.ex: Endereço 0

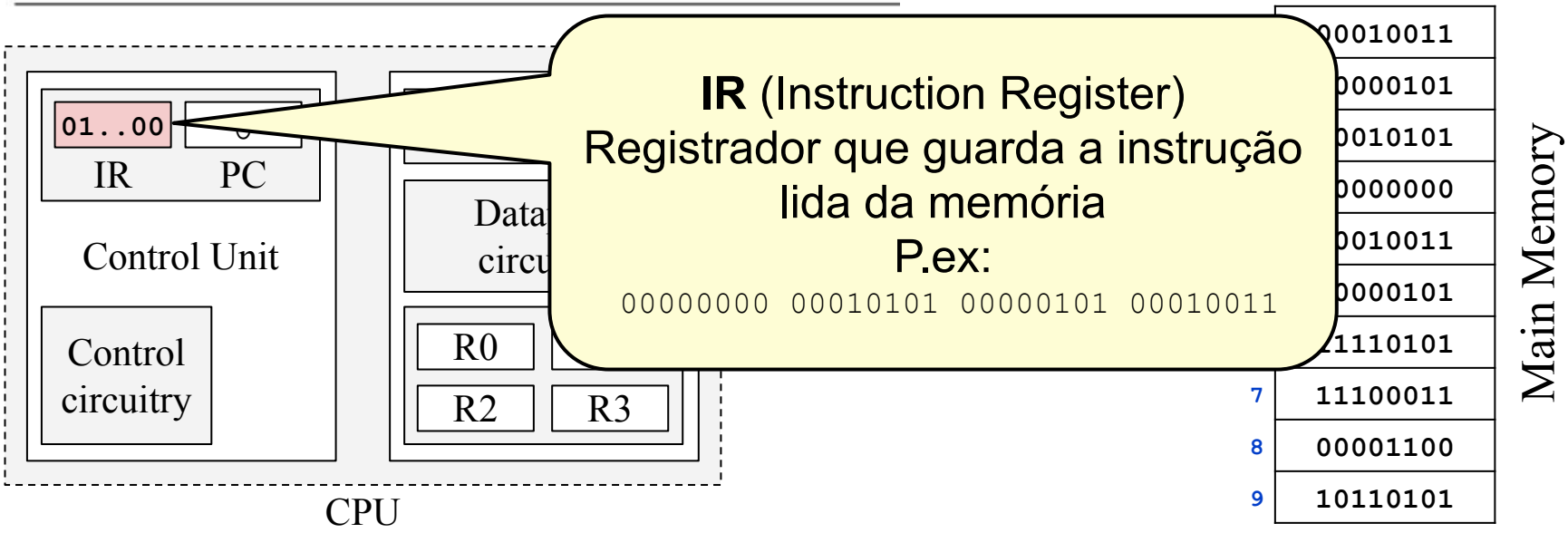
A Unidade Central de Processamento (CPU)

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
P.ex:
`00000000 00010101 00000101 00010011`

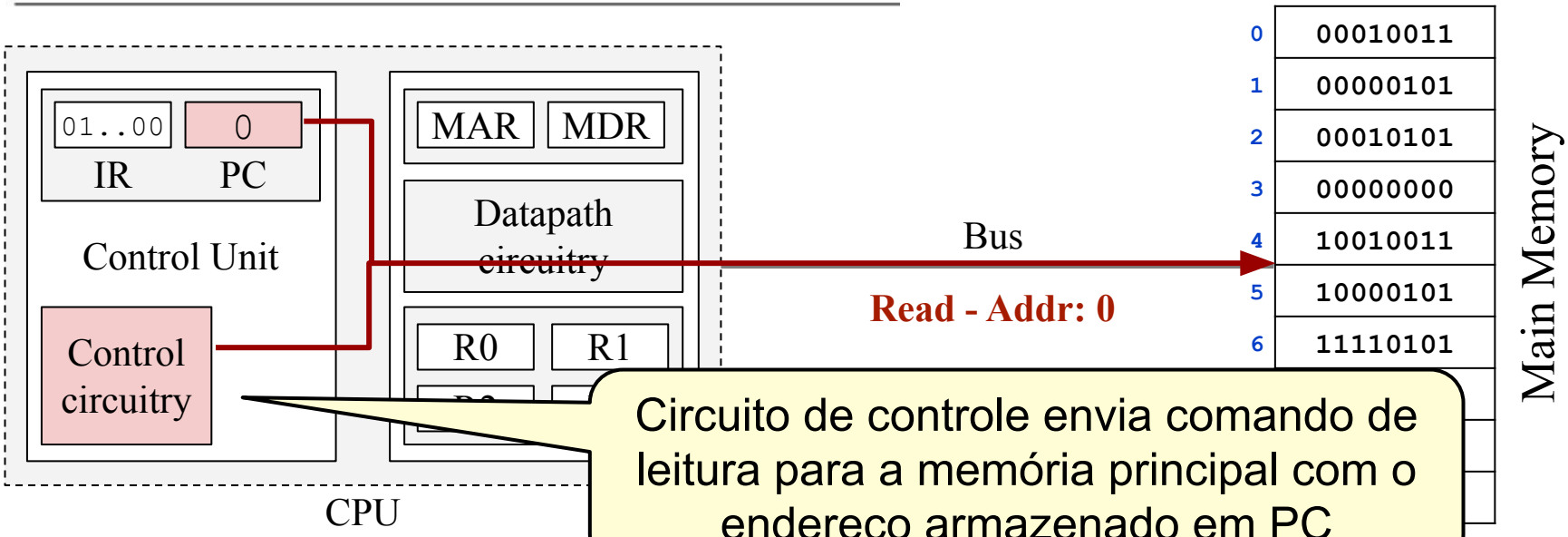
A Unidade Central de Processamento (CPU)

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!



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

A Unidade Central de Processamento (CPU)

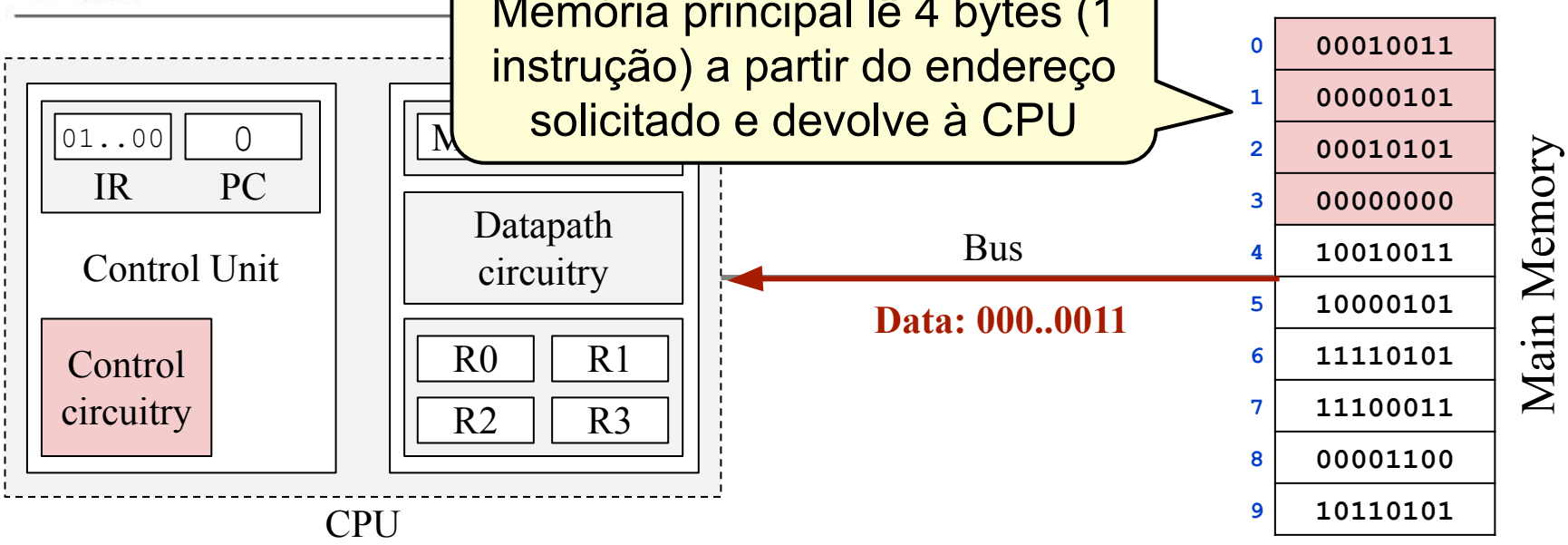
Executando instruções

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



A Unidade Central de Processamento (CPU)

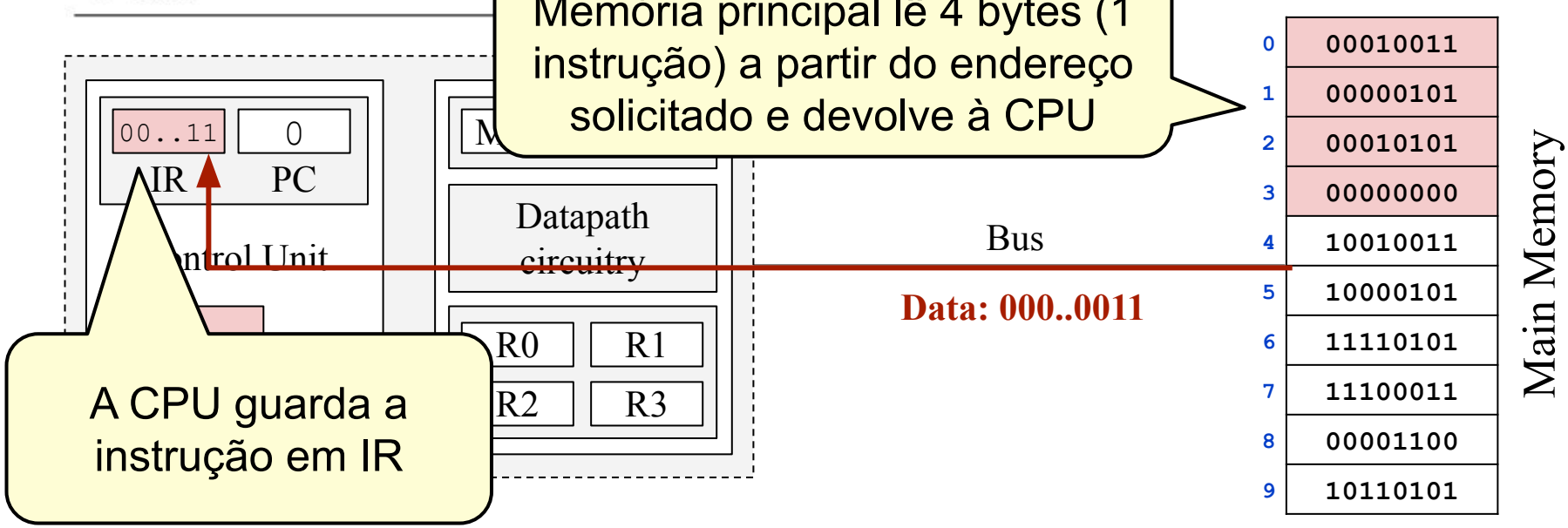
Executando instruções

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



A CPU guarda a instrução em IR

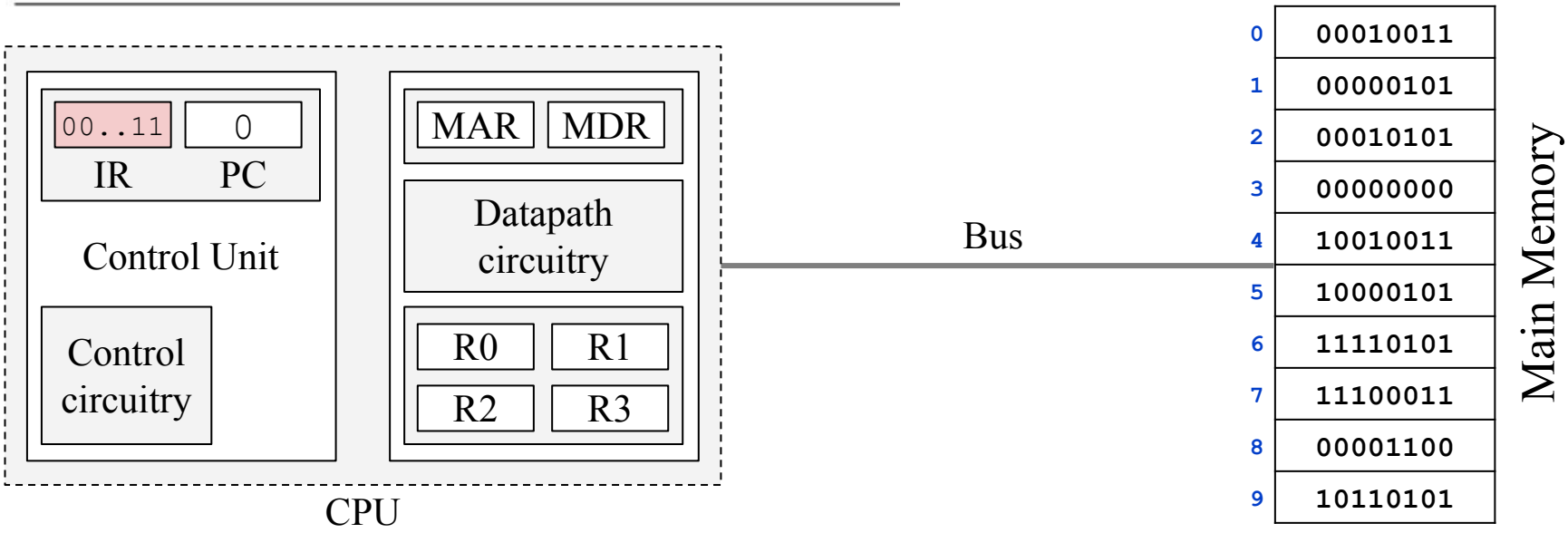
A Unidade Central de Processamento (CPU)

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!



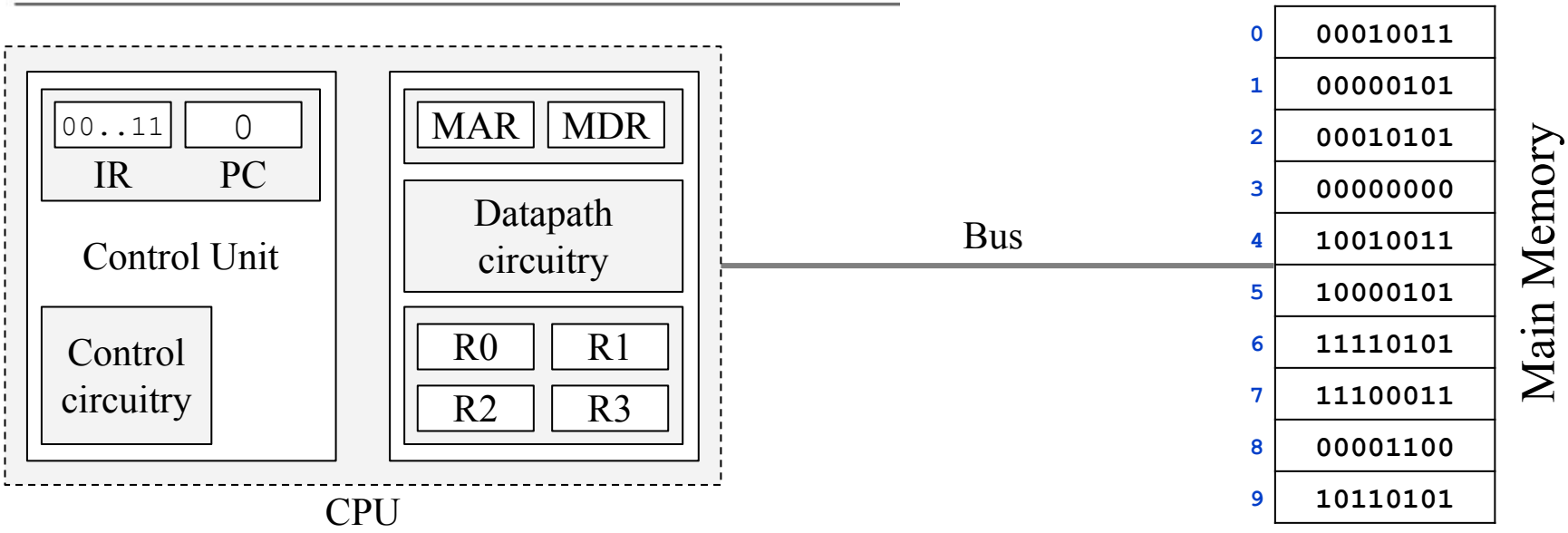
A Unidade Central de Processamento (CPU)

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!



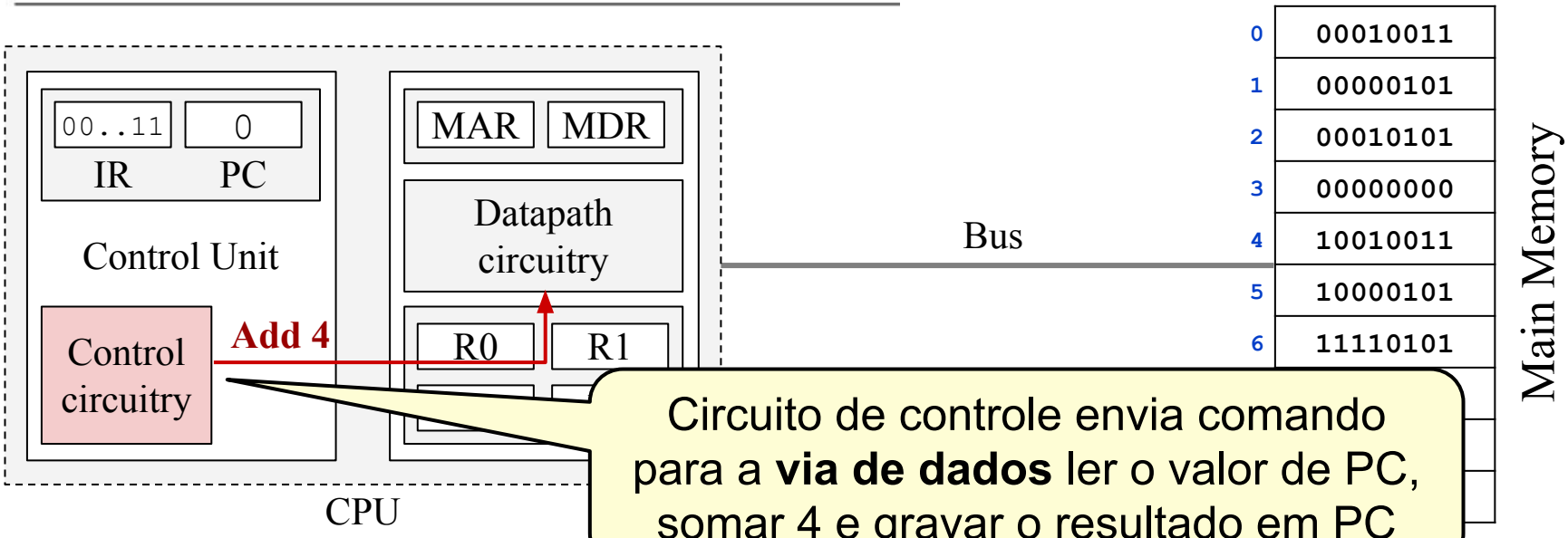
A Unidade Central de Processamento (CPU)

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!



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

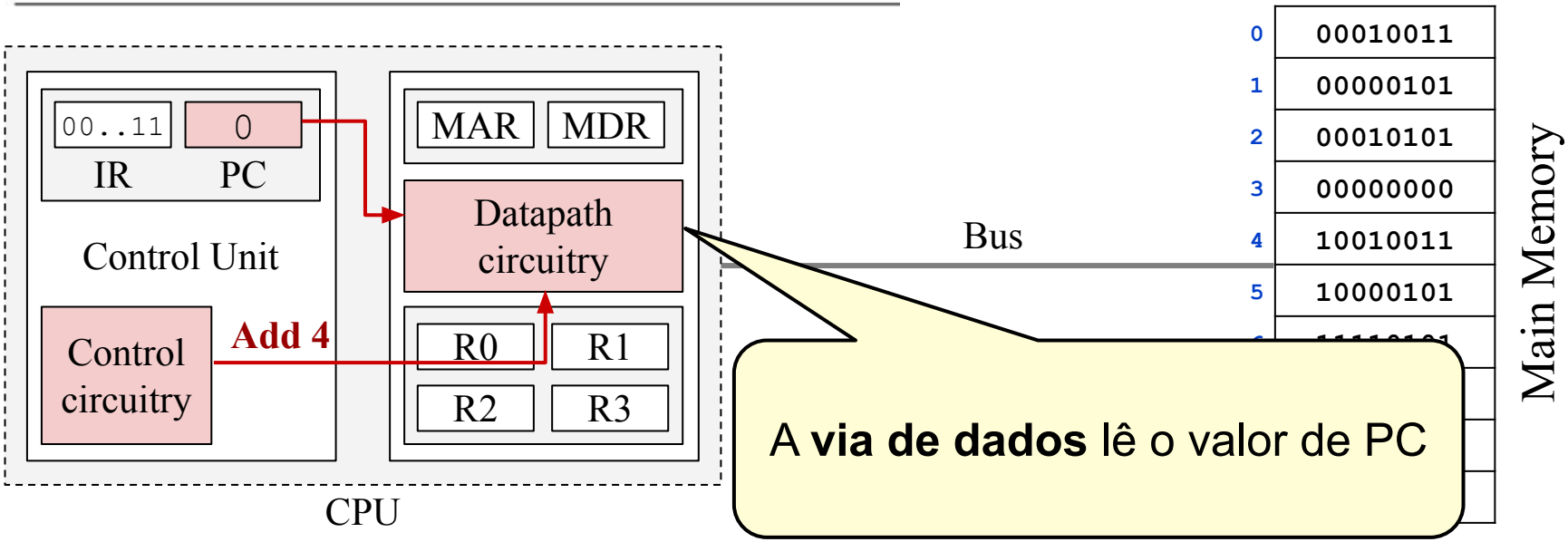
A Unidade Central de Processamento (CPU)

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!



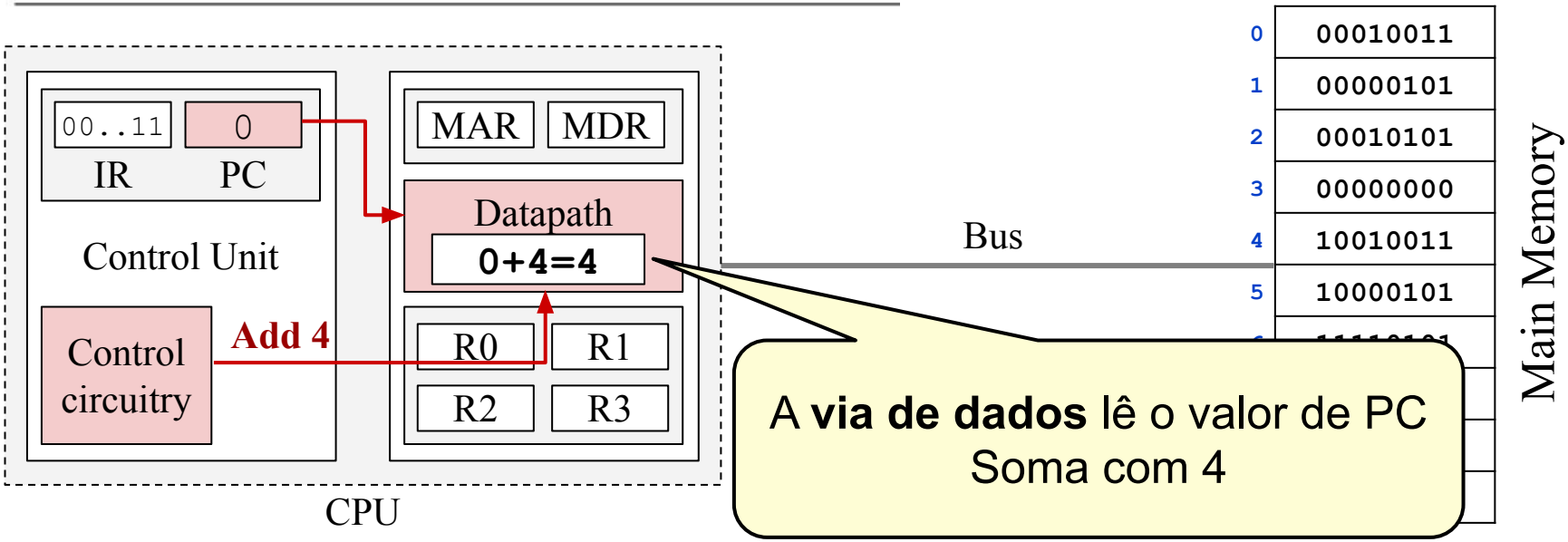
A Unidade Central de Processamento (CPU)

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!



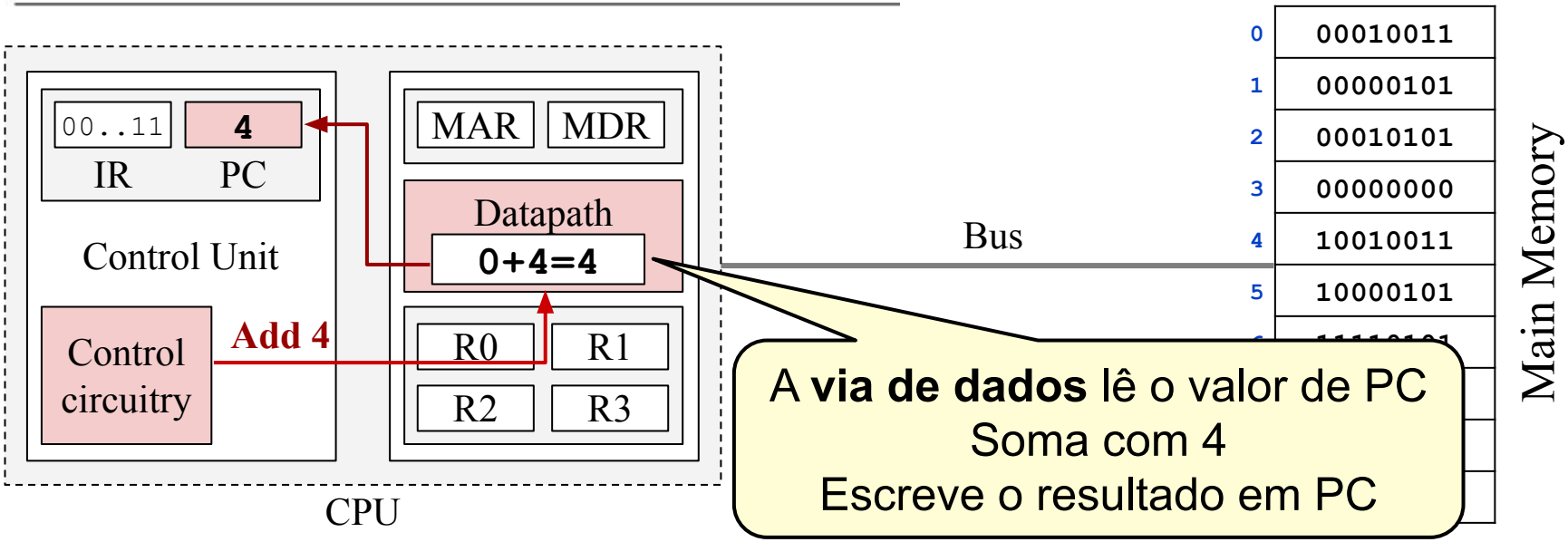
A Unidade Central de Processamento (CPU)

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!



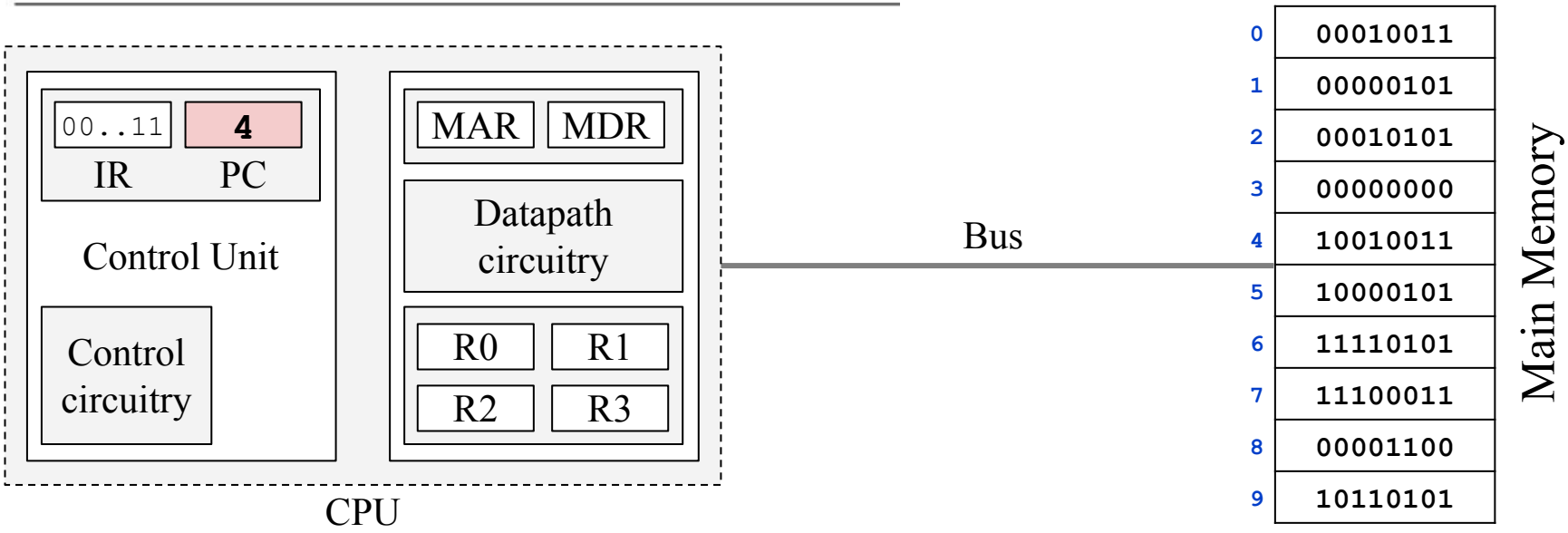
A Unidade Central de Processamento (CPU)

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!



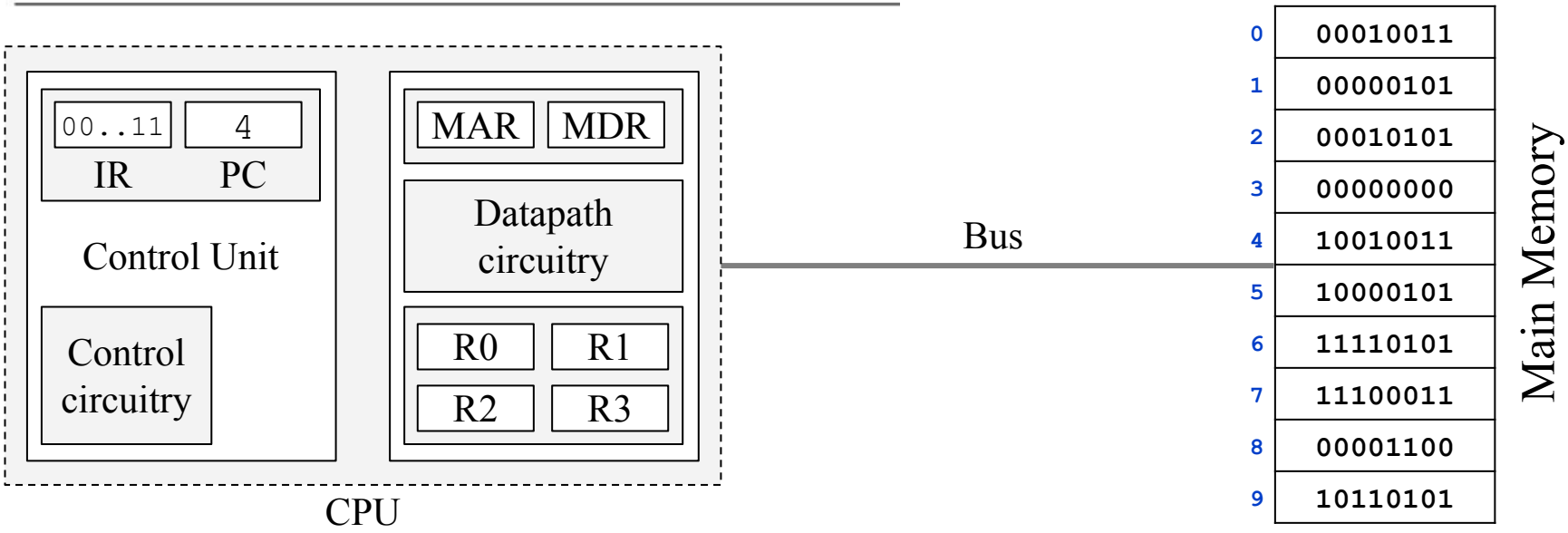
A Unidade Central de Processamento (CPU)

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



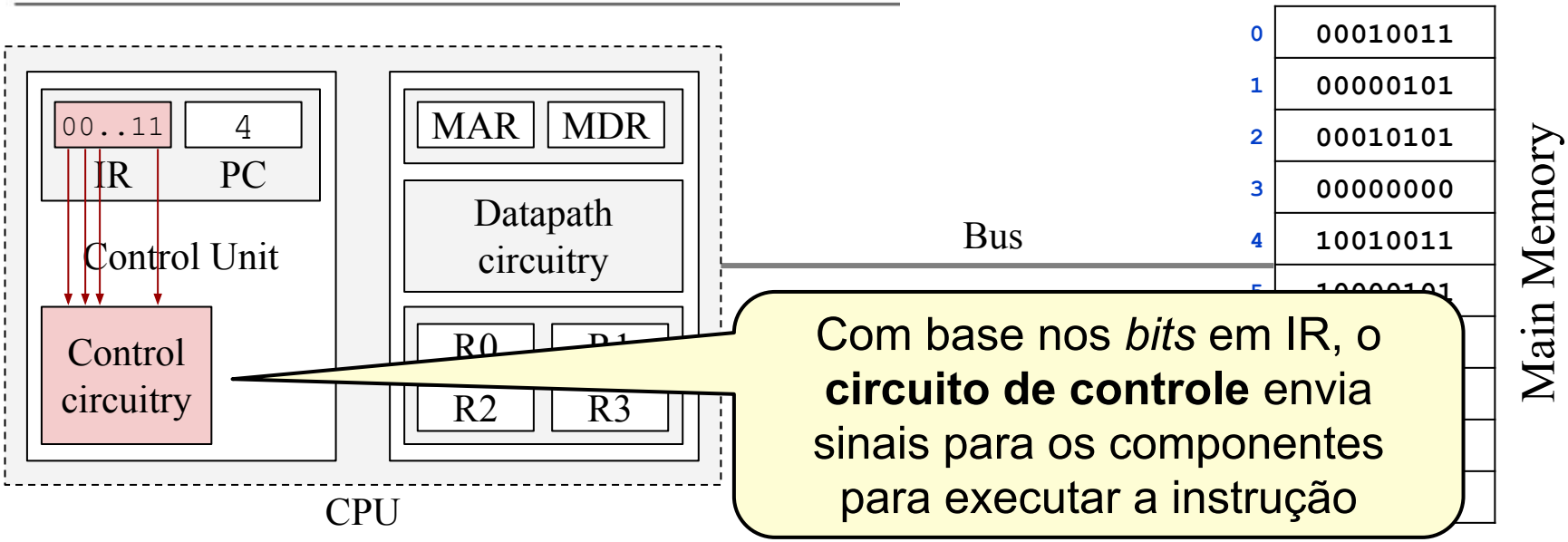
A Unidade Central de Processamento (CPU)

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



Com base nos *bits* em IR, o **circuito de controle** envia sinais para os componentes para executar a instrução

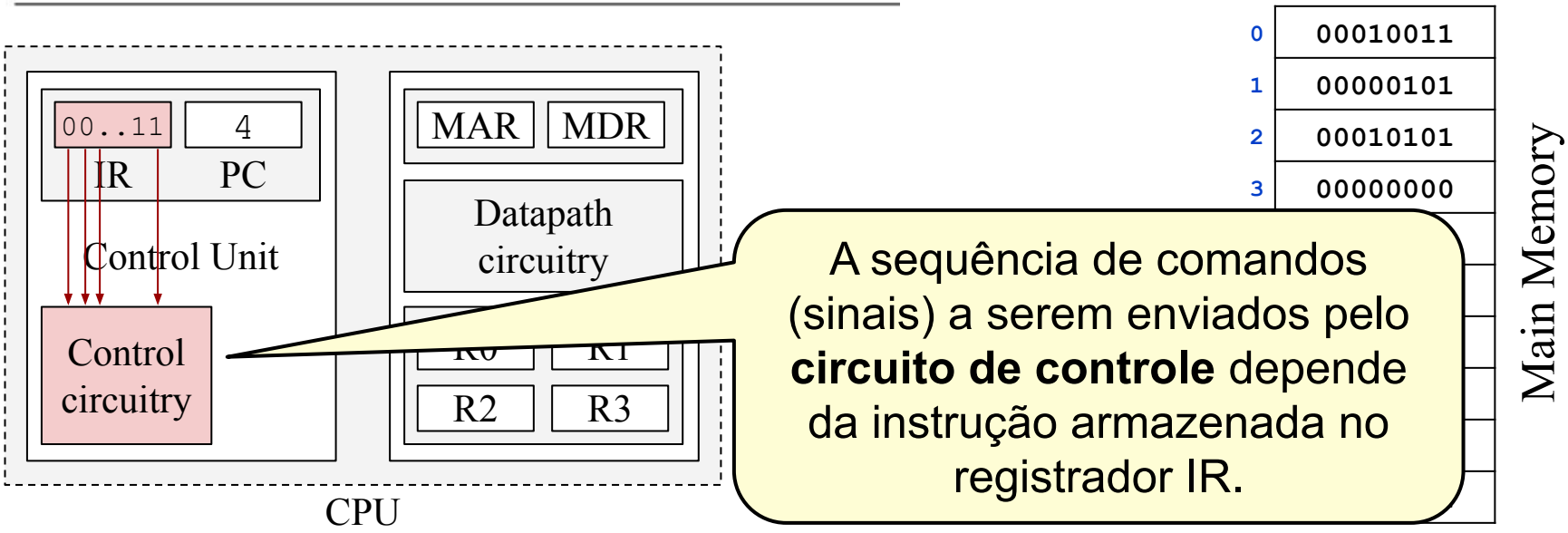
A Unidade Central de Processamento (CPU)

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.

A Unidade Central de Processamento (CPU)

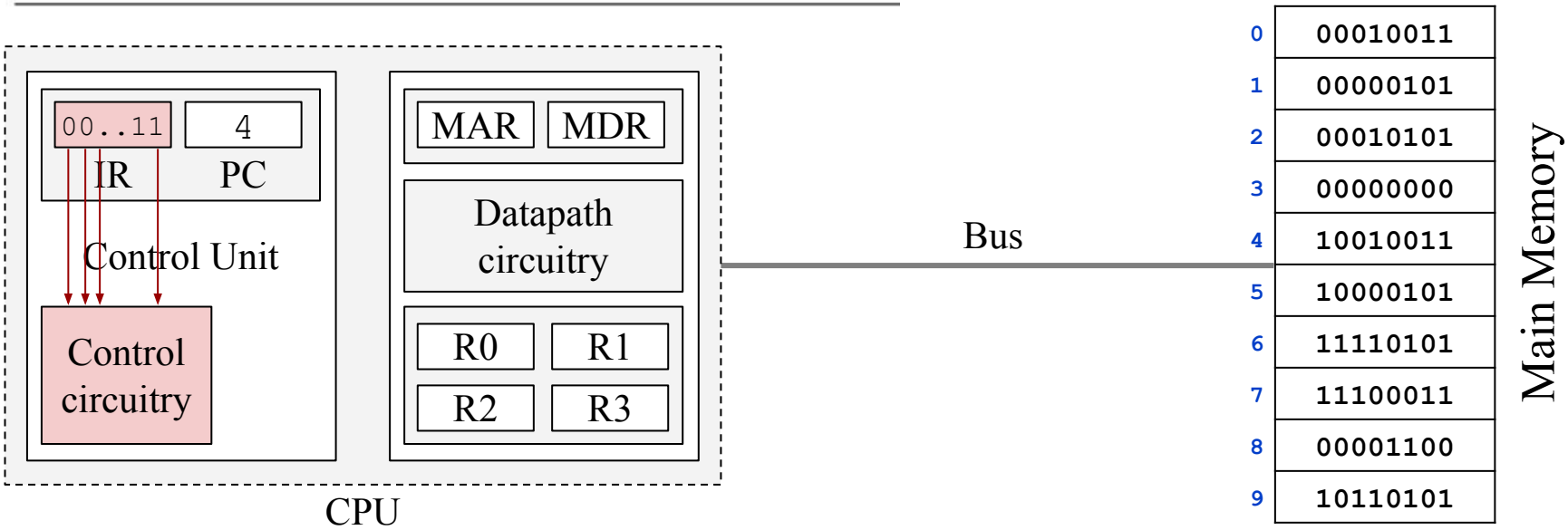
Executando instruções

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

Ex: `add R3,R1,R2`



A Unidade Central de Processamento (CPU)

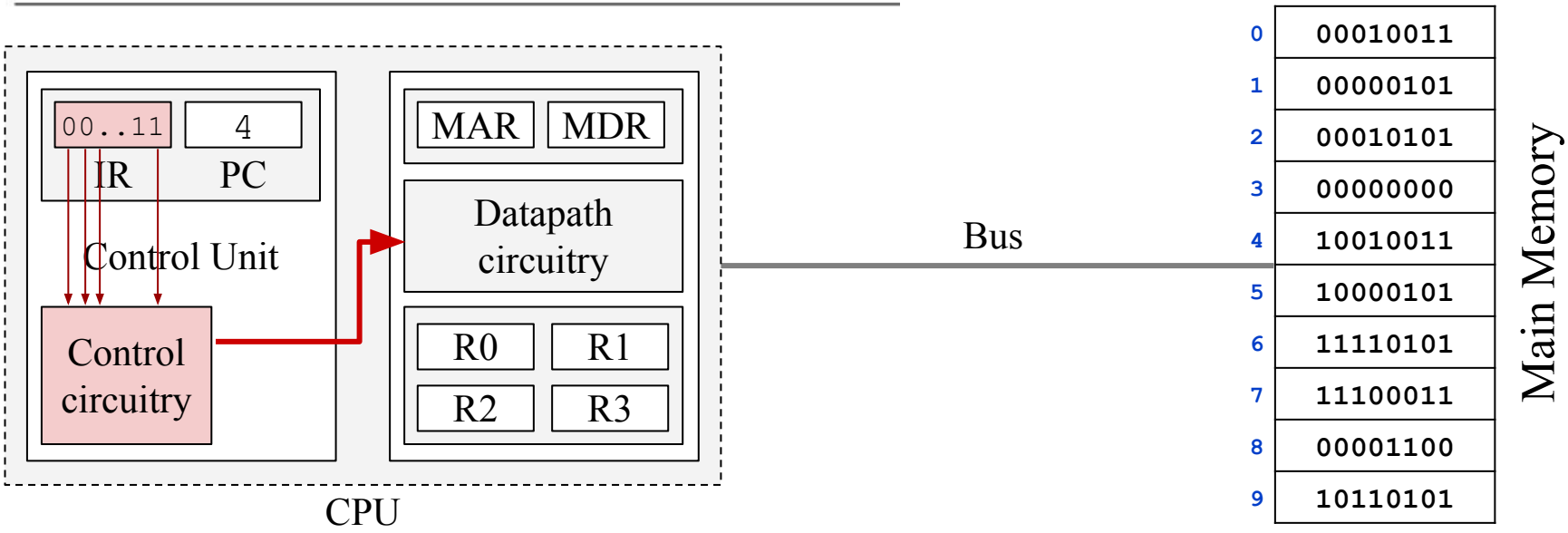
Executando instruções

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

Ex: `add R3,R1,R2`



A Unidade Central de Processamento (CPU)

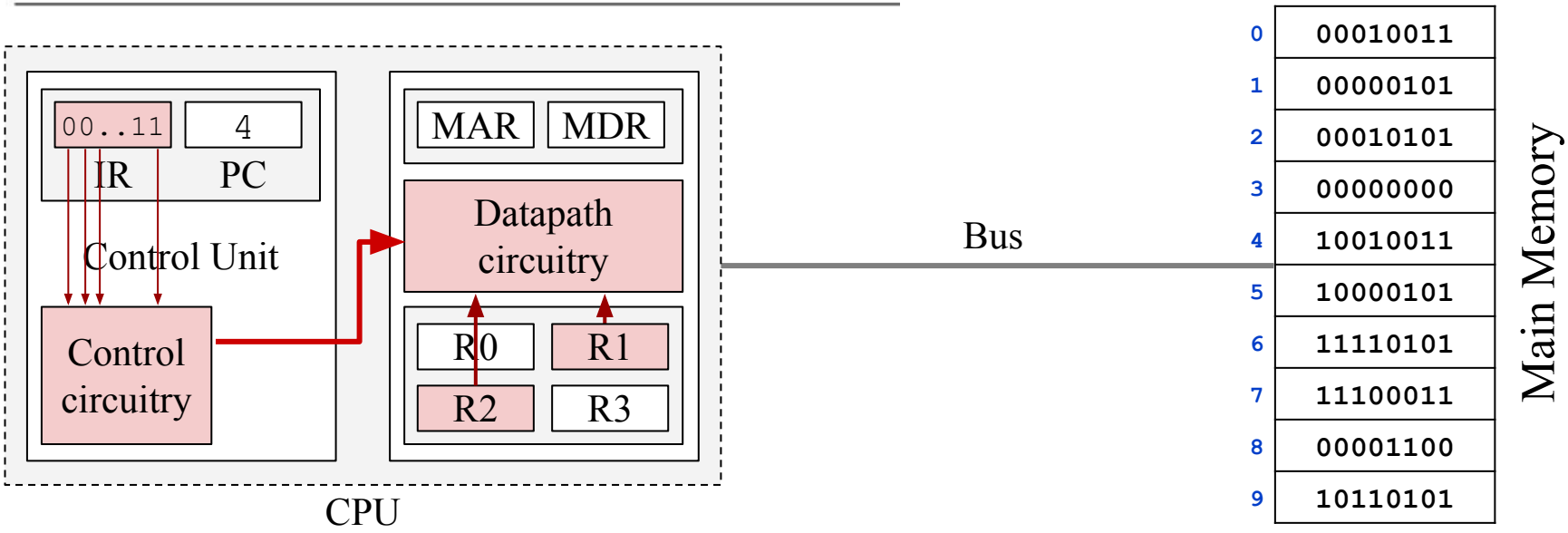
Executando instruções

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

Ex: `add R3,R1,R2`



A Unidade Central de Processamento (CPU)

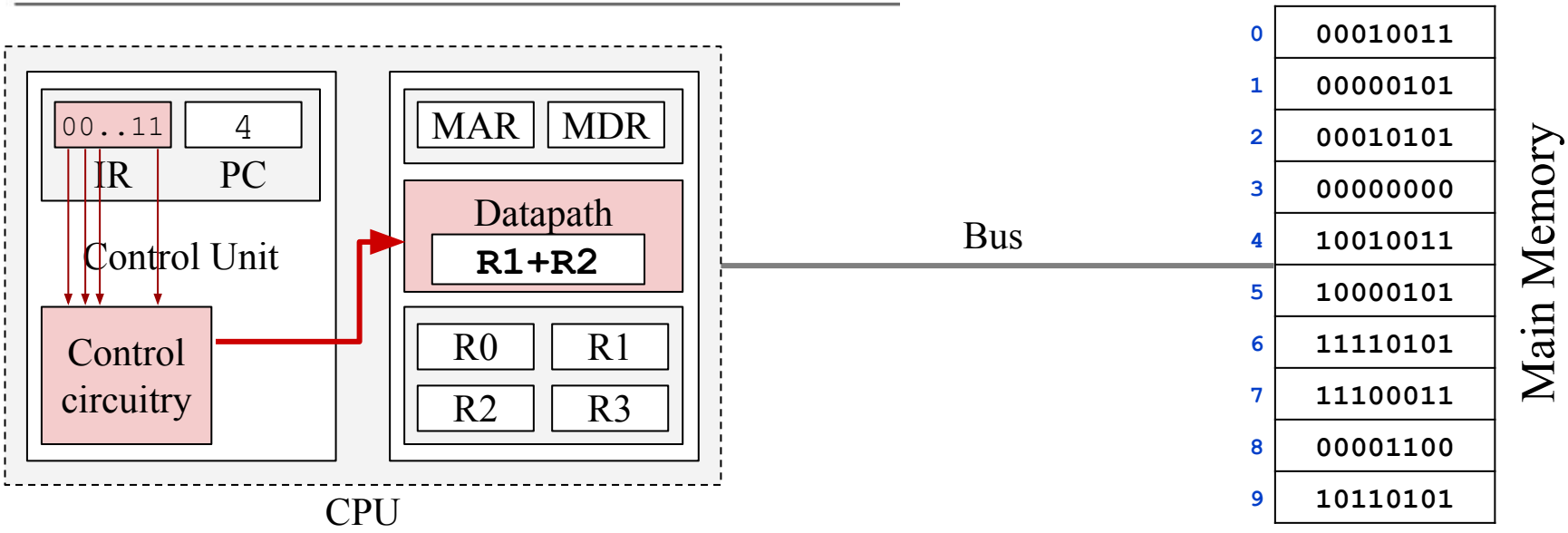
Executando instruções

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

Ex: `add R3,R1,R2`



A Unidade Central de Processamento (CPU)

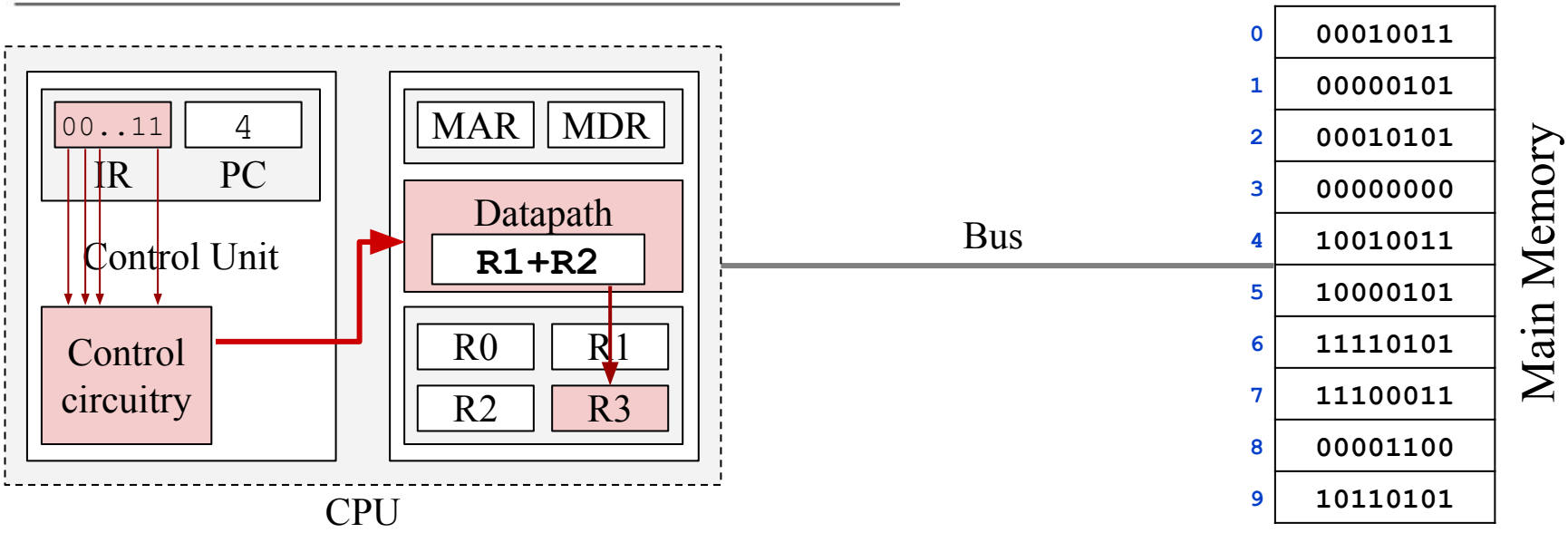
Executando instruções

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

Ex: `add R3,R1,R2`



A Unidade Central de Processamento (CPU)

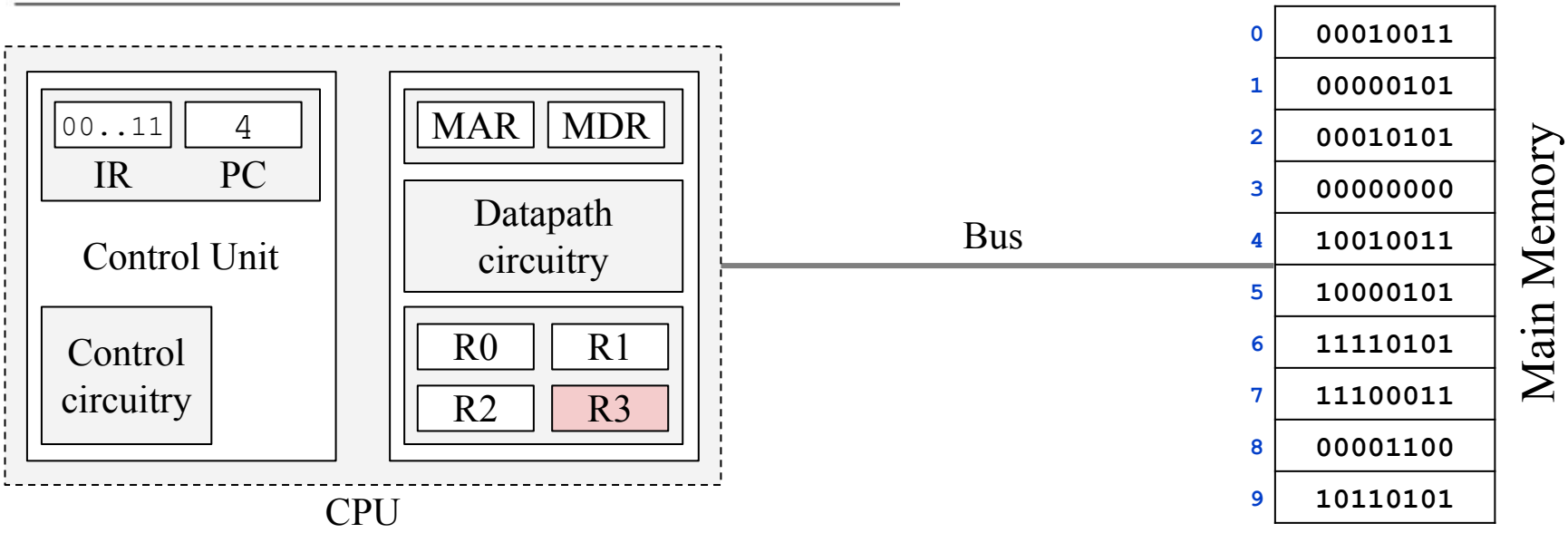
Executando instruções

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

Ex: `add R3,R1,R2`



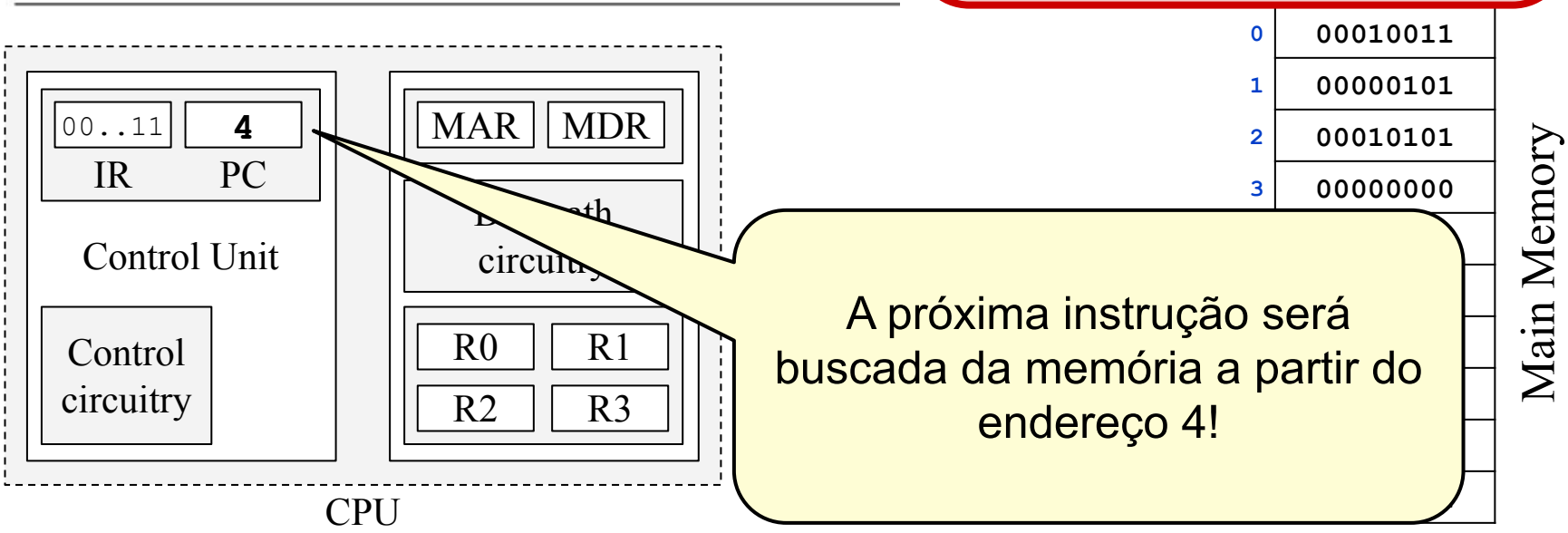
A Unidade Central de Processamento (CPU)

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



A Unidade Central de Processamento (CPU)

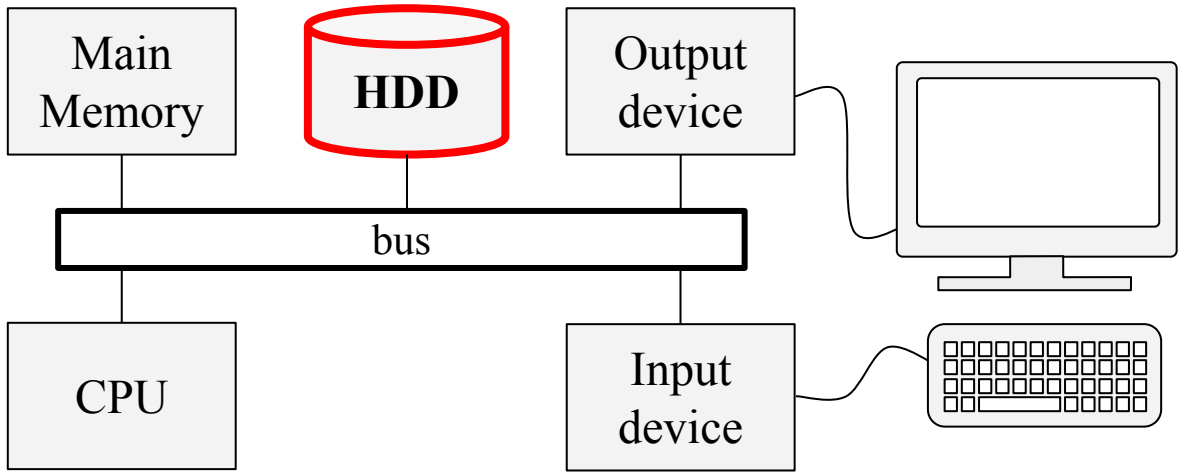
Resumo:

- 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 Central de Processamento
- **Memória secundária (persistente)**
- Barramento
- Periféricos



Memória secundária (persistente)

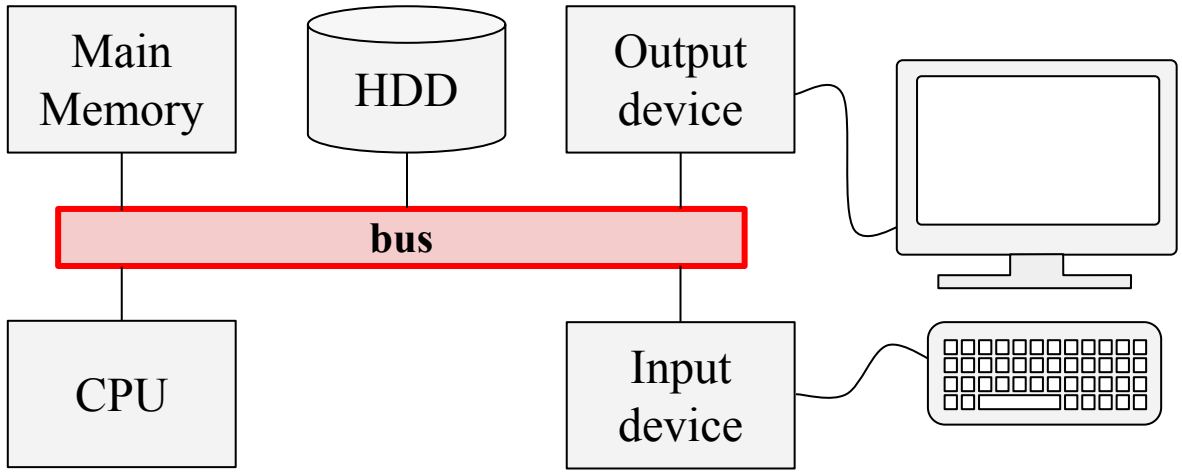
A memória principal é geralmente volátil, *i.e.*, seu conteúdo é perdido quando o sistema é desligado. A memória secundária é persistente e serve para armazenar os dados e programas de forma persistente.

- 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 Central de Processamento
- Memória secundária (persistente)
- **Barramento**
- Periféricos



Barramento

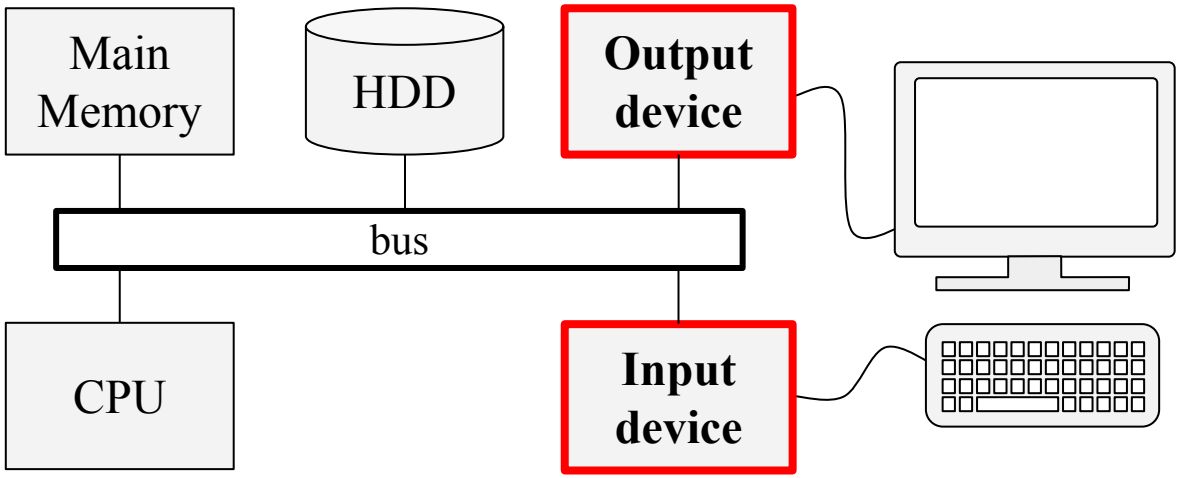
O barramento é um sistema de comunicação que transfere informação entre os componentes do computador (p.ex.: entre a memória e a CPU).

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

Componentes de um computador

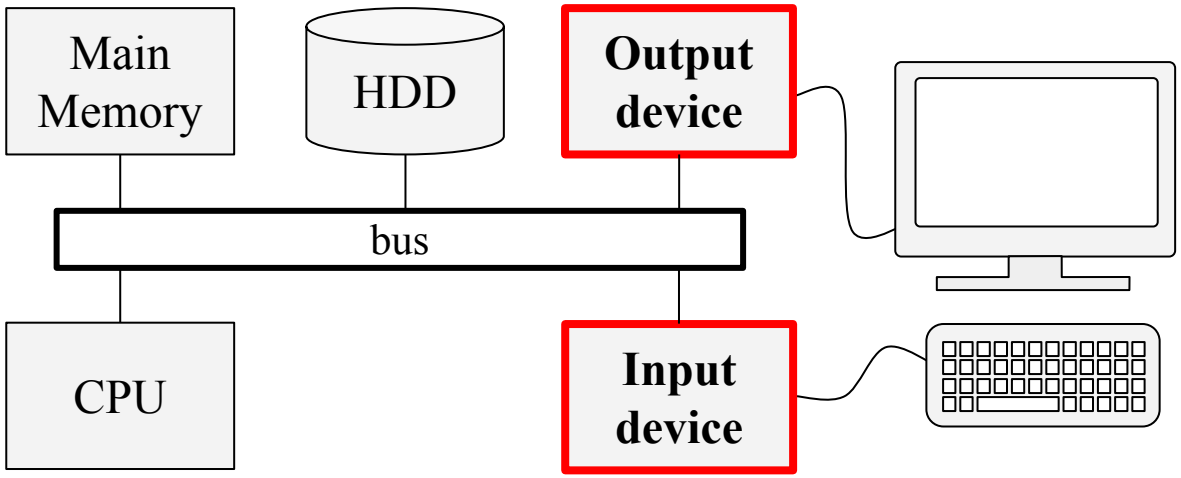
Um computador é geralmente composto pelos seguintes componentes:

- Memória principal
- CPU - Unidade Central de Processamento
- Memória secundária (persistente)
- Barramento
- **Periféricos**



Periféricos

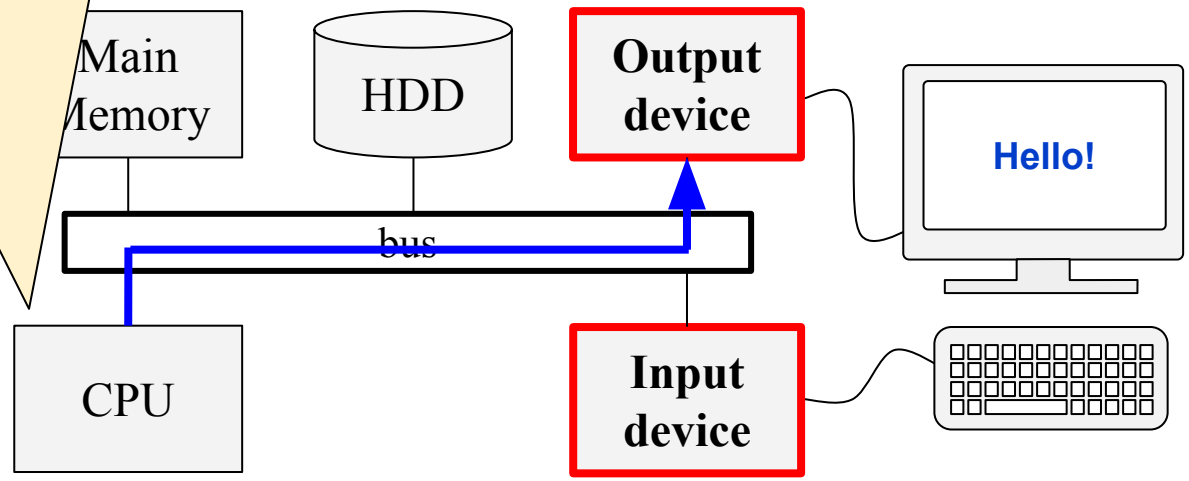
Periféricos são dispositivos de entrada e saída (E/S) de dados e são conectados ao restante dos componentes através do barramento.



Periféricos

Instruções do programa podem fazer com que a CPU leia ou escreva dados em periféricos.

os de entrada e saída (E/S) de
o restante dos
barramento.



Agenda

- Componentes de um computador
- **Codificação de programas de computador**
- Geração de programas nativos
- Execução de programas nativos

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. P.ex: Compilado para formato executável.

```c
#include<stdio.h>

int main()
{
    printf("Hello! I'm a C program!\n");
    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
  - P.ex: Com o interpretador do Bash e/ou Python.

```
1
2   mkdir "new-folder"
3   echo "Hello! I'm a bash script" > new-folder/new-file.txt
4
```

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!

Agenda

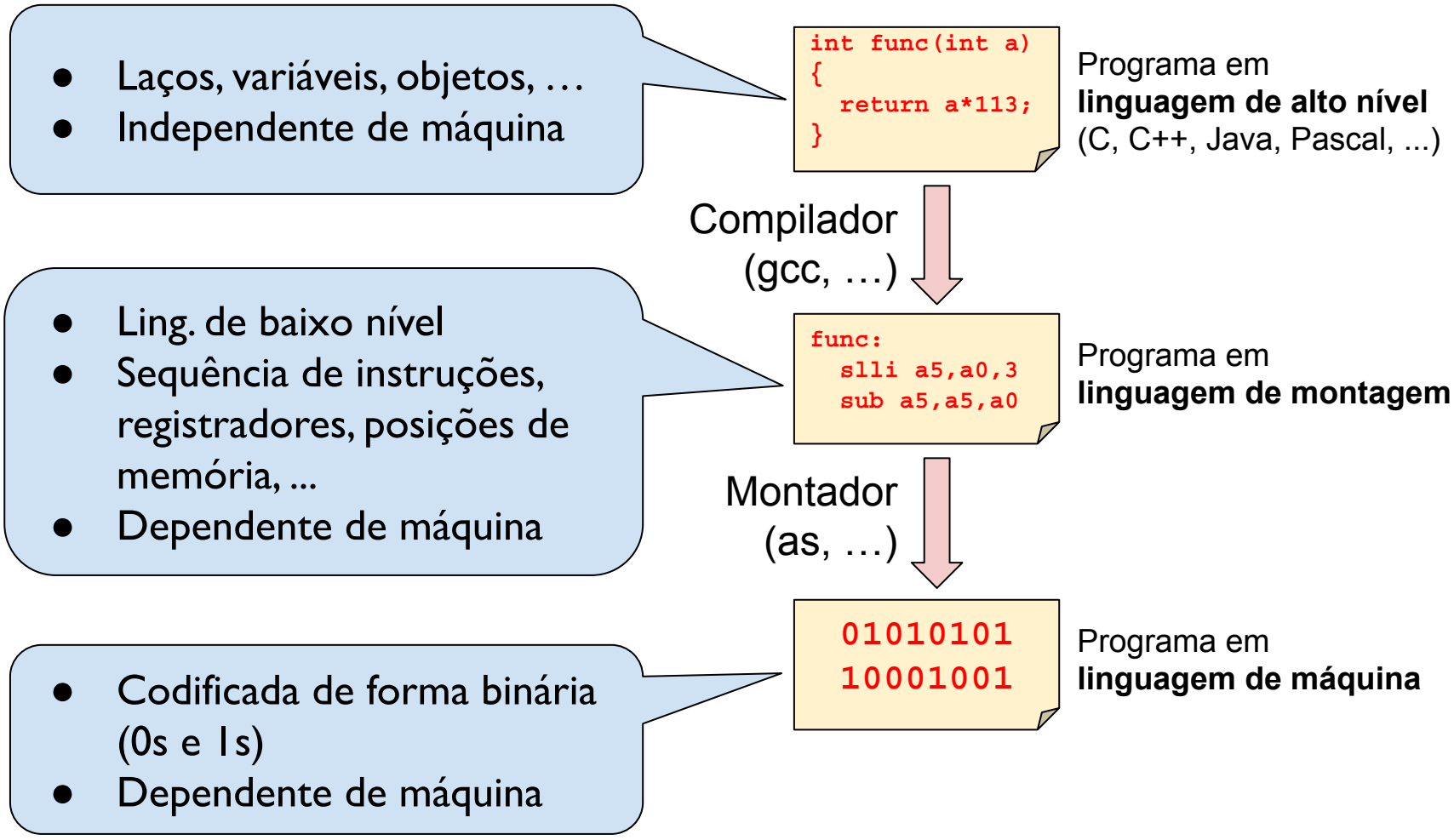
- Componentes de um computador
- Codificação de programas de computador
- **Geração de programas nativos**
- Execução de programas nativos

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



Agenda

- Componentes de um computador
- Codificação de programas de computador
- Geração de programas nativos
- **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!