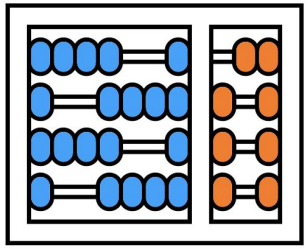


Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

Organização Básica de computadores e linguagem de montagem

Acessando periféricos

Prof. Edson Borin

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

Institute of Computing - UNICAMP

Agenda

- **Periféricos**
- Conexão de periféricos com a CPU
- Leitura e escrita de dados em periféricos
- Técnica de espera ocupada (*Busy waiting*)

Periféricos

Periféricos são dispositivos de entrada e saída (E/S), ou *I/O devices*.

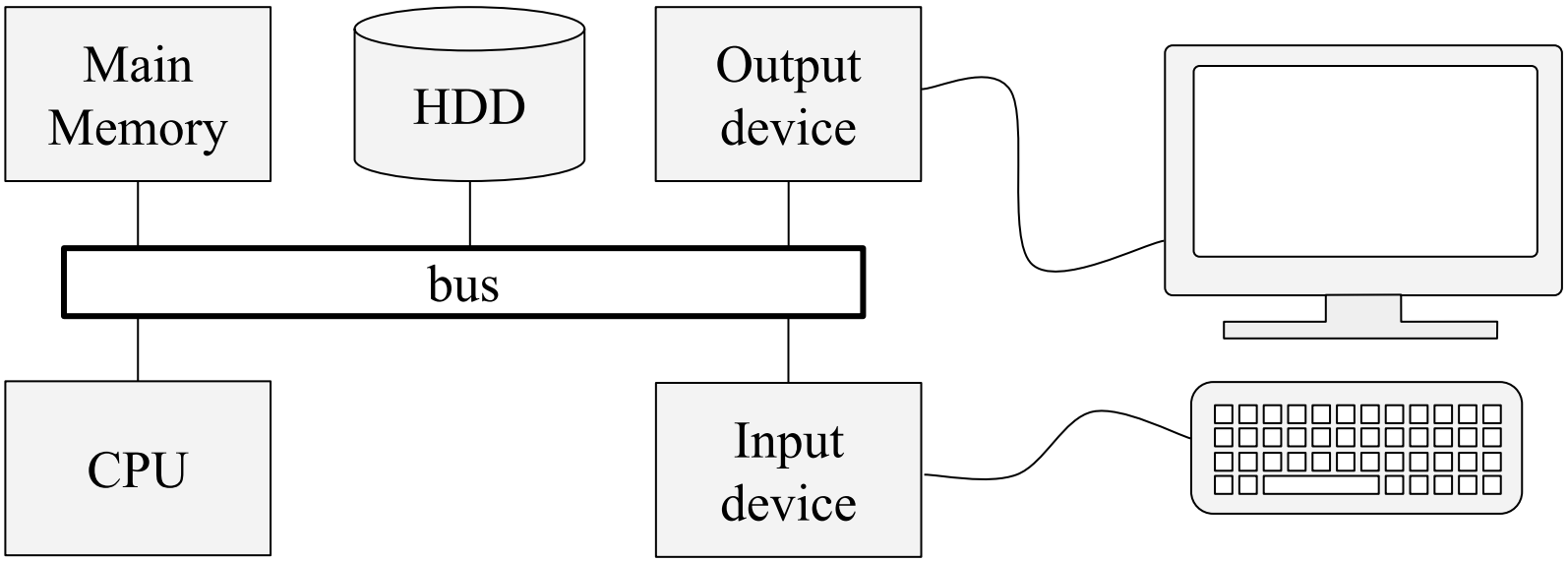
Diversos tipos:

- **Entrada**: mouse, teclado, microfone, câmera web, …
- **Saída**: Monitor, impressora, …
- **E/S**: dispositivos de armazenamento de dados (HDD, USB flash drives, …)

Periféricos

Periféricos são conectados à CPU através de barramentos.

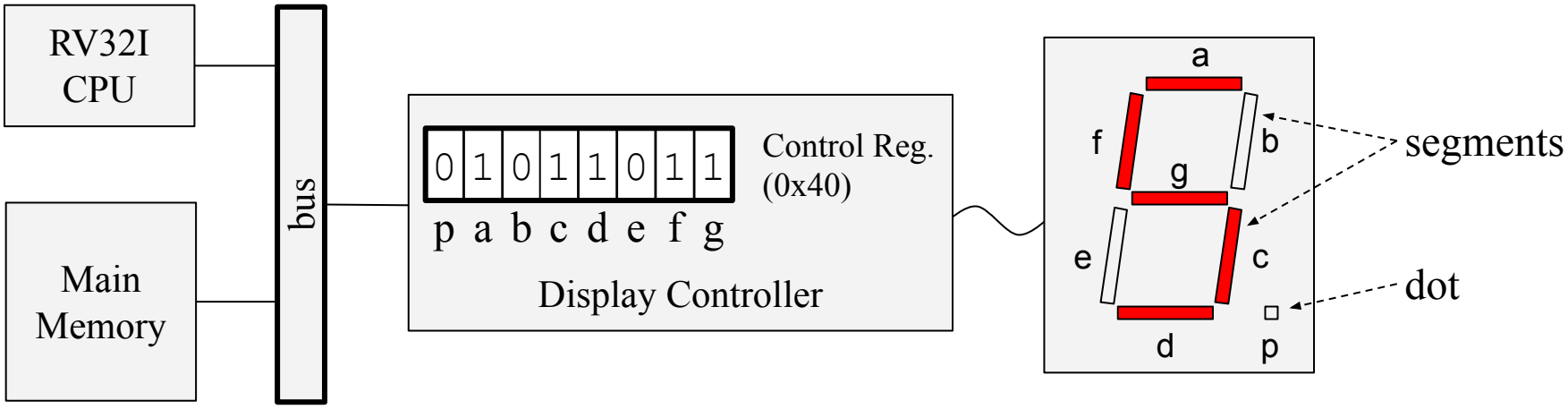
- Exemplo:



Periféricos

Periféricos podem possuir registradores ou memórias internas!

- **Operações de entrada e saída são realizadas através da leitura e escrita nestes registradores e memórias internas**.
- Exemplo: Controlador de *display*

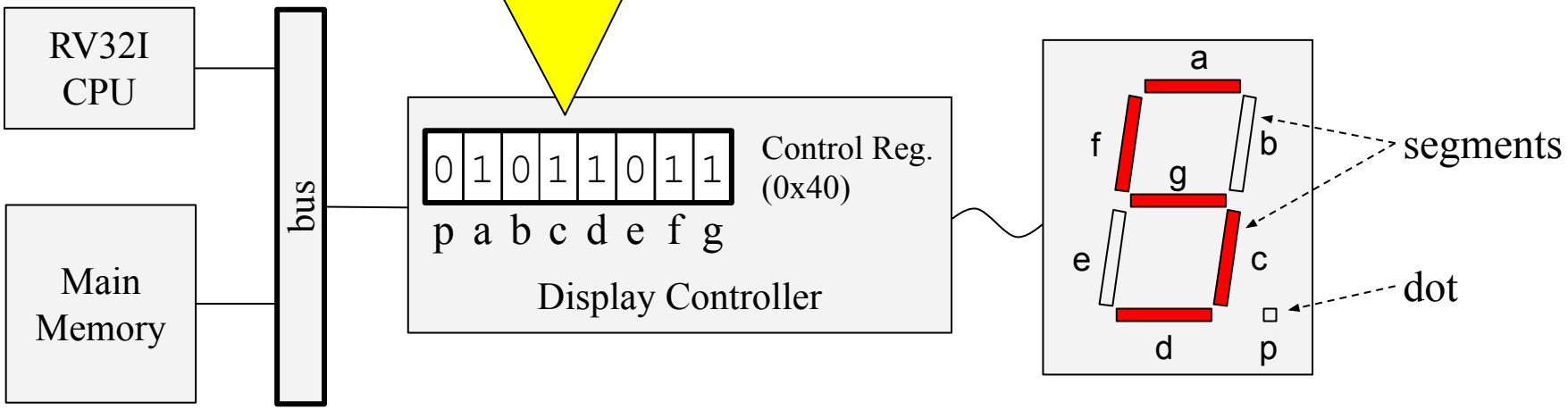


Periféricos

Periféricos podem possuir registradores ou memó

- **Op**
  rea
  nestes re res e memórias internas.
- Exemplo: C dor de *display*

Escrita do valor `0b01011011` no registrador de controle (Control Reg.) faz o display controller ligar os segmentos **a**, **c**, **d**, **f** e **g** e desligar o restante.



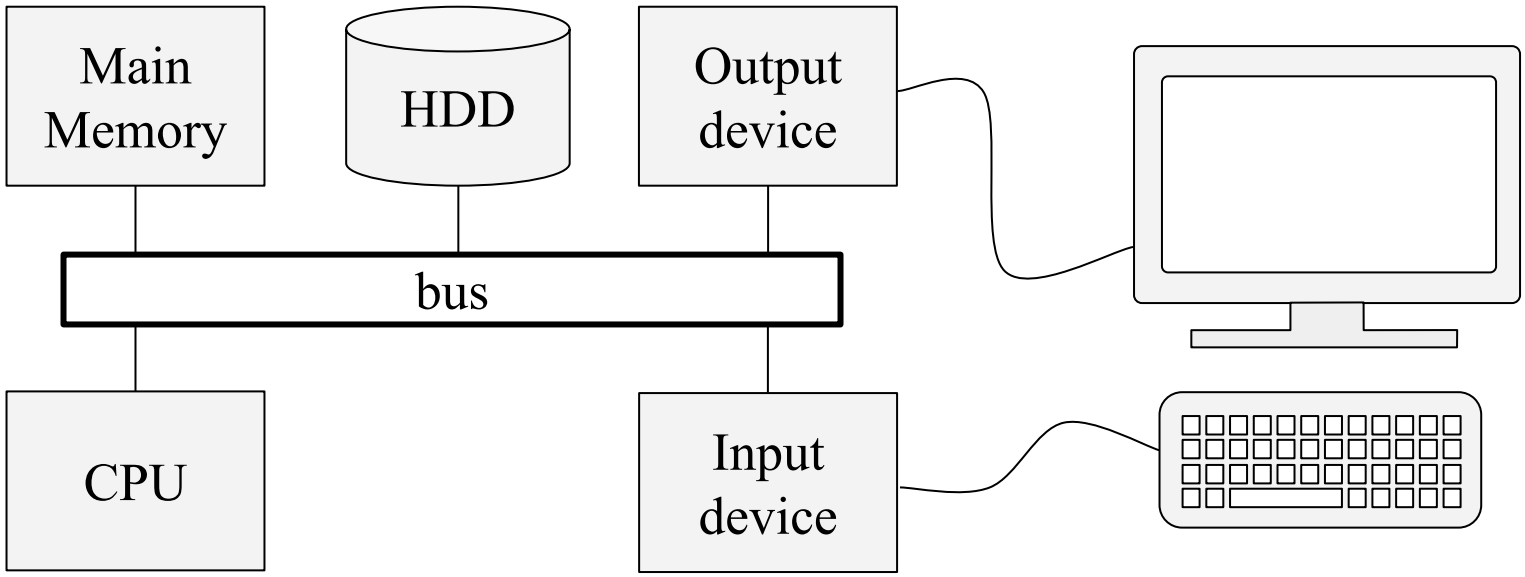
Agenda

- Periféricos
- **Conexão de periféricos com a CPU**
- Leitura e escrita de dados em periféricos
- Técnica de espera ocupada (*Busy waiting*)

Conexão de periféricos com a CPU

Periféricos, a CPU e a memória principal são conectados fisicamente por **barramentos**.

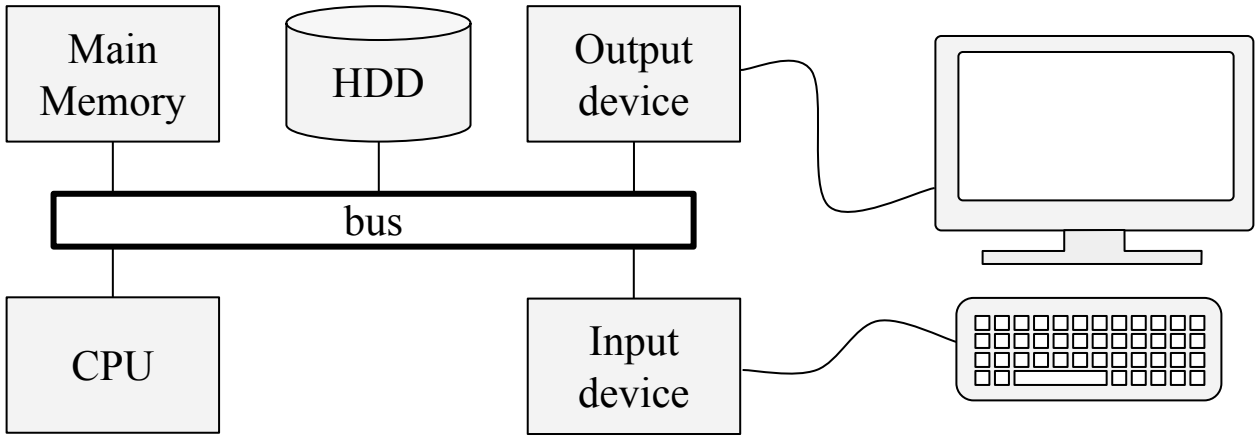
Exemplo:



Conexão de periféricos com a CPU

Barramentos: caminhos de comunicação entre dois ou mais dispositivos.

- PCI: desenvolvido originalmente pela Intel. Atualmente é um padrão público
- AMBA: desenvolvido pela ARM.



Conexão de periféricos com a CPU

Barramentos: caminhos de comunicação entre dois ou mais dispositivos.

- A comunicação é realizada através do envio de endereços, dados e comandos.
  - Exemplo: para escrever o valor 0x42 na palavra de memória associada ao endereço 0x00080000, a CPU envia pelo barramento o endereço 0x00080000, o dado 0x42, e o comando WRITE para a memória principal.

Conexão de periféricos com a CPU

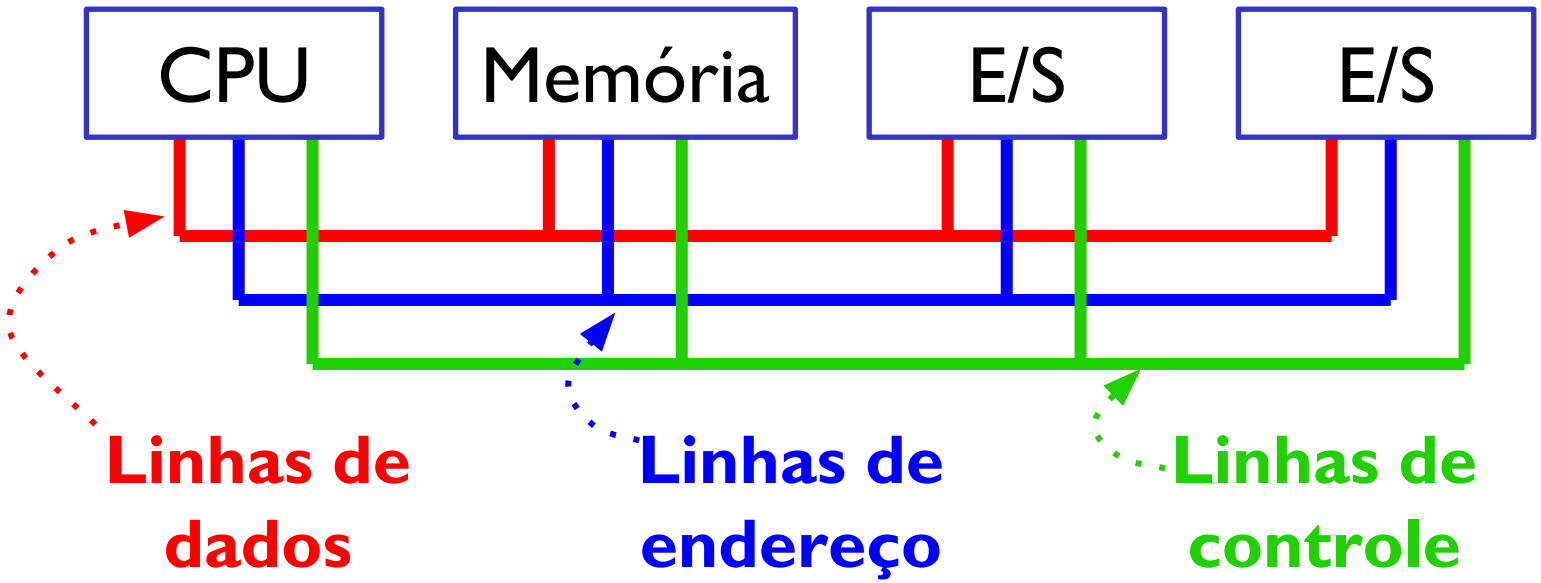
Barramentos: caminhos de comunicação entre dois ou mais dispositivos.

- O barramento contém linhas de comunicação (fios) que transmitem a informação.
- O barramento pode compartilhar as linhas de comunicação para transferir os endereços, dados e comandos, ou pode ter linhas exclusivas para cada tipo de informação. (p.ex: linhas de dados, linhas de endereço e linhas de controle)
    - Depende da implementação!

Conexão de periféricos com a CPU

Barramentos: caminhos de comunicação entre dois ou mais dispositivos.

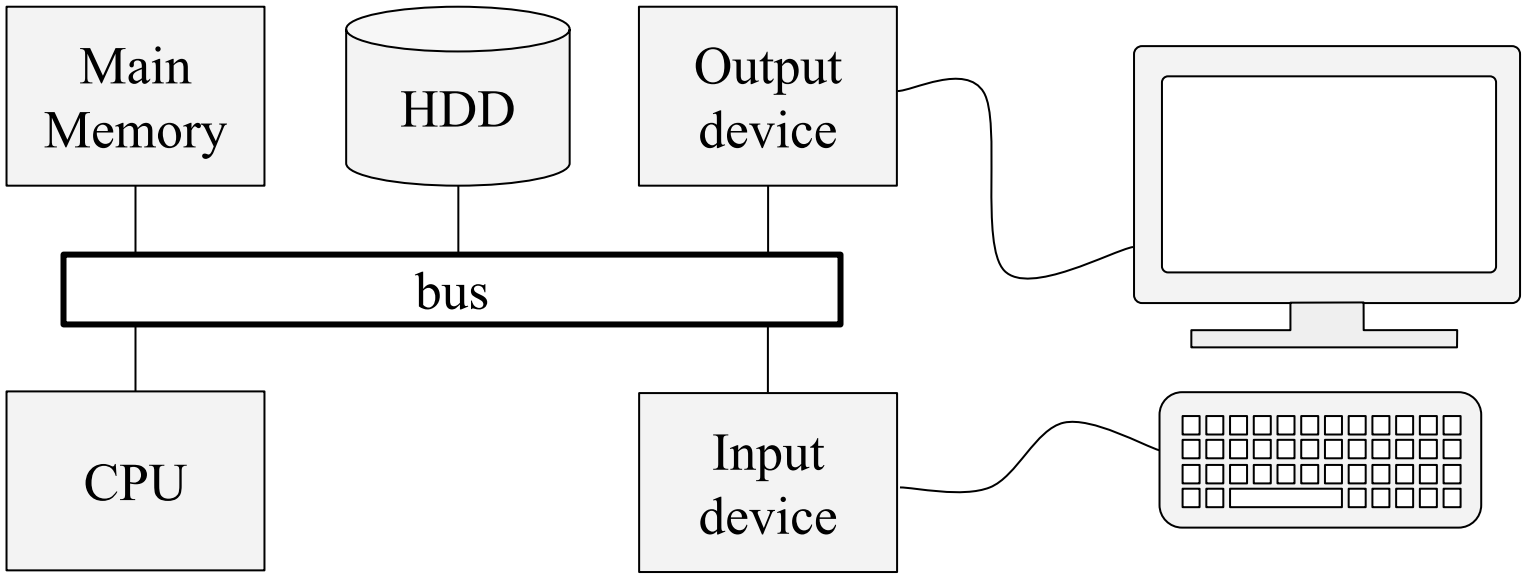
- Exemplo com linhas exclusivas para cada tipo de informação



Conexão de periféricos com a CPU

Barramentos: caminhos de comunicação entre dois ou mais dispositivos.

- Todos os dispositivos (CPU, memória, HDD, …) podem estar ligados a um mesmo barramento.

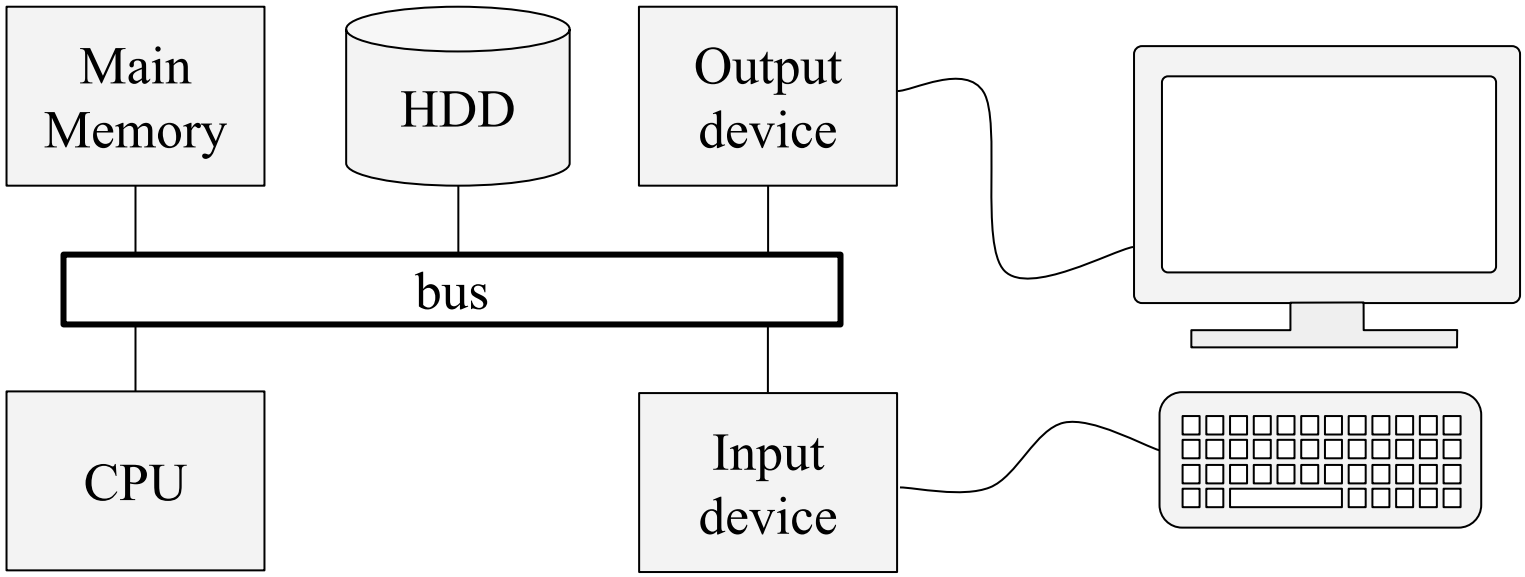


Conexão de periféricos com a CPU

Barramentos: caminhos de comunicação entre dois

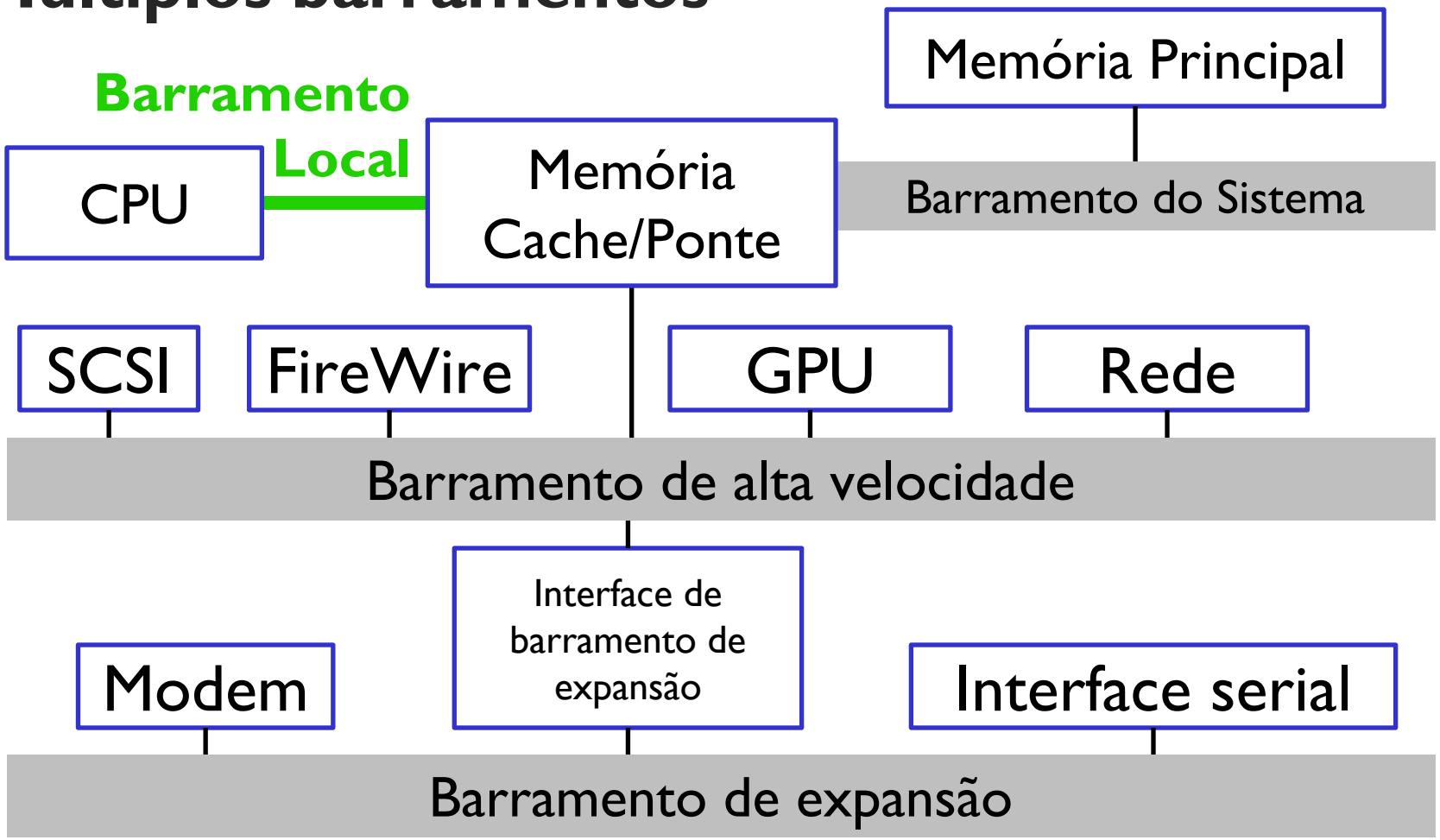
- T a, HDD, …) p rramento.

Problema! Todos têm que operar na mesma velocidade



Conexão de periféricos com a CPU

Múltiplos barramentos



Conexão de periféricos com a CPU

A implementação de barramentos pode variar drasticamente de um sistema para outro.

A CPU abstrai os detalhes do funcionamento do barramento para o programa.

- Do ponto de vista do programador, a comunicação é realizada através da execução de instruções que realizam a leitura/escrita de dados de/em registradores (ou memória interna) dos periféricos!

Agenda

- Periféricos
- Conexão física
- **Leitura e escrita de dados em periféricos**
- Técnica de espera ocupada (*Busy waiting*)

Leitura e escrita de dados em periféricos

Recap: Operações de entrada e saída são realizadas através da leitura e escrita nos registradores e memórias internas dos periféricos.

- CPU ⇔ Periférico:
  - Realizado com instruções do ISA
    - Port-mapped I/O
    - Memory-mapped I/O

- Memória Principal ⇔ Periférico (DMA)

Leitura e escrita de dados em periféricos

Port-mapped I/O: método que emprega instruções especializadas para copiar dados entre a CPU e os Periféricos.

- ***I/O port***: valor numérico que identifica os registradores e posições de memórias internas dos periféricos
- ***I/O instruction***: instrução especializada para copiar dados entre a CPU e os periféricos.
- Exemplo:

Instruções *input from port* (`in`) e *output to port* (`out`) na arquitetura IA-32.

```
in 0x71, %al
out %al, 0x70
```

Leitura e escrita de dados em periféricos

Port-mapped I/O: método que emprega instruções especializadas para copiar dados entre a CPU e os Periféricos.

- ***I/O port***: valor numérico que identifica os registradores e posições de memórias internas dos periféricos
- ***I/O address space***: conjunto de valores válidos de ***I/O ports***.
- Espaço de endereçamento de palavras da memória principal (*memory address space*) e o *I/O address space* são distintos neste método!

Leitura e escrita de dados em periféricos

Mesmo valor numérico (ex: 0x70) pode ser usado para identificar uma palavra da memória principal e um registrador de um periférico.

e emprega instruções
os entre a CPU e os

que identifica os
e memórias internas
d iféricos

- ***I/O address space***: conjunto de valores válidos de ***I/O ports***.
- Espaço de endereçamento de palavras da memória principal (memory address space) e o I/O address space são distintos neste método!

Leitura e escrita de dados em periféricos

Mesmo valor numérico (ex: 0x70) pode ser usado para identificar uma palavra da memória principal e um registrador de um periférico.

e emprega instruções

os entre a CPU e os

que identifica os

e memórias internas

d féricos

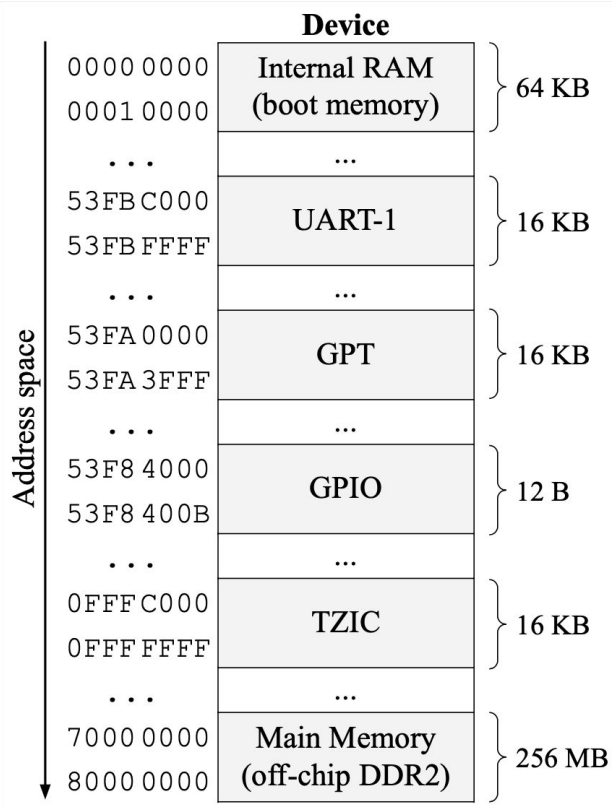
- ***I/O address space***: de ***I/O ports***.
- Espaço de endereçam memória principal (m I/O address space são

A CPU diferencia a leitura e escrita de dados na memória principal e nos periféricos com base na instrução sendo executada!

Leitura e escrita de dados em periféricos

Memory-mapped I/O: método que faz uso das mesmas instruções para copiar dados entre a CPU e os periféricos e a CPU e a memória principal!

- Há apenas um espaço de endereçamento.
- Partes distintas do espaço (faixas de endereço) são mapeados na memória principal e em periféricos.



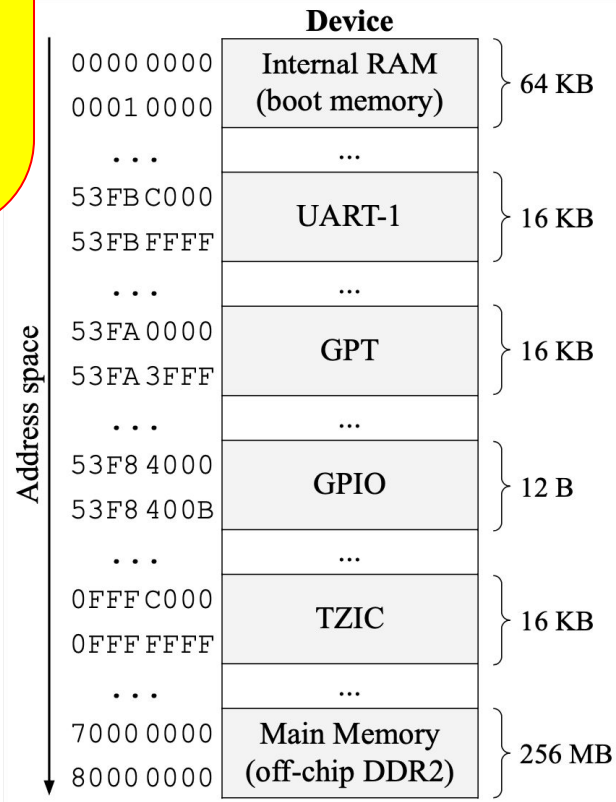
Leitura e escrita de dados em periféricos

Instrução *load word* (`lw`) sendo usada para realizar entrada: ler um valor de um registrador do periférico GPIO e gravar o valor no registrador `a1`.

o que faz uso das
lados entre a CPU
mória principal!

- [illegible] (fa[illegible] endereço) são ma[illegible]os na memória prin[illegible]al e em periféricos.

```
li a0, 0x53F84000
lw a1, (a0)
```



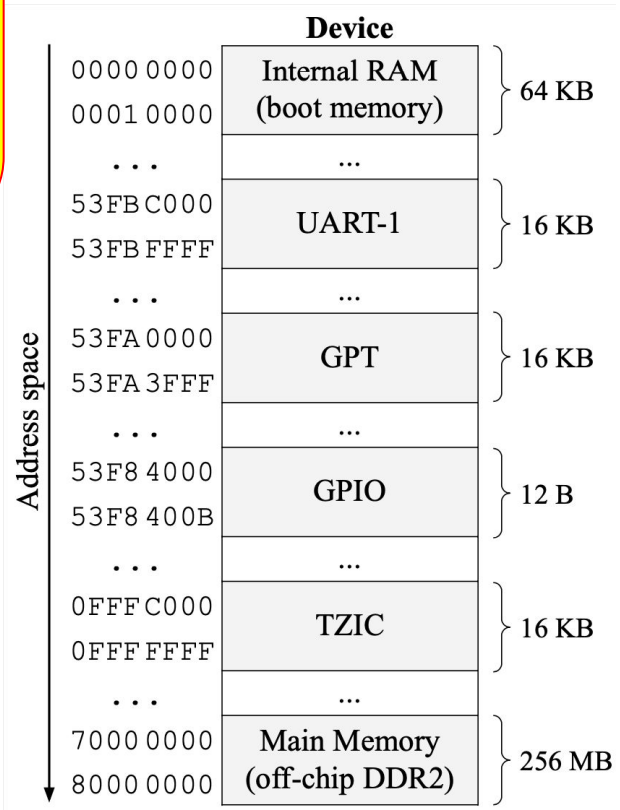
Leitura e escrita de dados em periféricos

Memory mapped I/O: método que faz uso das dados entre a CPU emória principal!

- [illegible] endereço) são mapeados na memória principal e em periféricos.

Instrução *store byte* (`sb`) sendo usada para realizar saída: escrever o valor em `a1` em um registrador do periférico GPT.

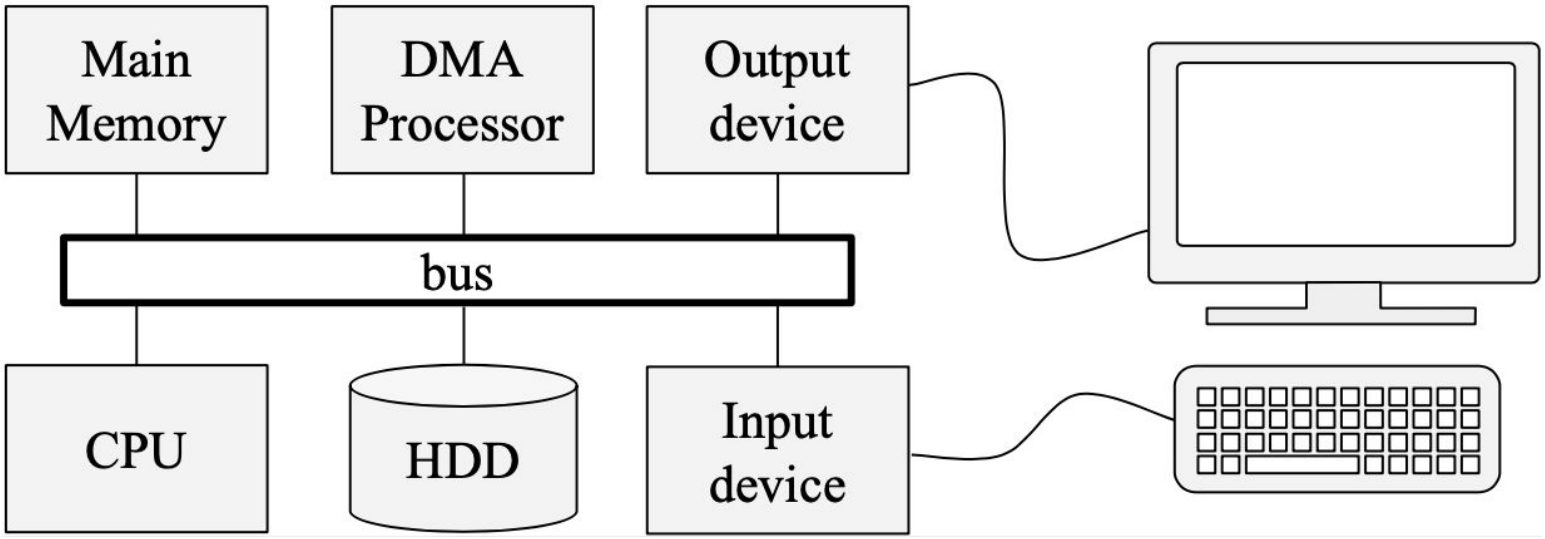
```
li a0, 0x53FA0000
sb a1, (a0)
```



Leitura e escrita de dados em periféricos

Memória Principal ⇔ Periférico (DMA):

- DMA Processor: Co-processador especializado
  - Periférico que é programado pela CPU usando port-mapped I/O ou memory-mapped I/O
- Exemplo:



Leitura e escrita de dados em periféricos

Entrada e saída na arquitetura RV32I

- Memory-mapped I/O!
- Instruções de transferência de dados entre a memória e os registradores da CPU (*load e store*) são usadas.

Leitura e escrita de dados em periféricos

Entrada e saída na arquitetura RV32I

- Memory-mapped I/O!
- Instruções de transferência de dados entre a memória e os registradores da CPU (*load e store*) são usadas.
- Exemplos:

```
li a0, 0x53FA0000
sb a1, (a0)
```

Operação de saída.

```
li a0, 0x53F84000
lw a1, (a0)
```

Operação de entrada.

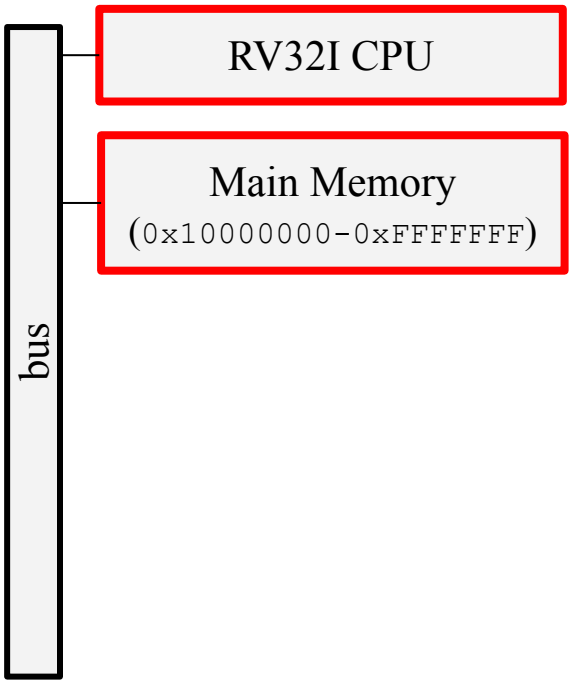
Leitura e escrita de dados em periféricos

Entrada e saída na arquitetura RV32I

Exemplo: Elevador

O sistema possui:

- Um processador RV32I e uma memória principal.



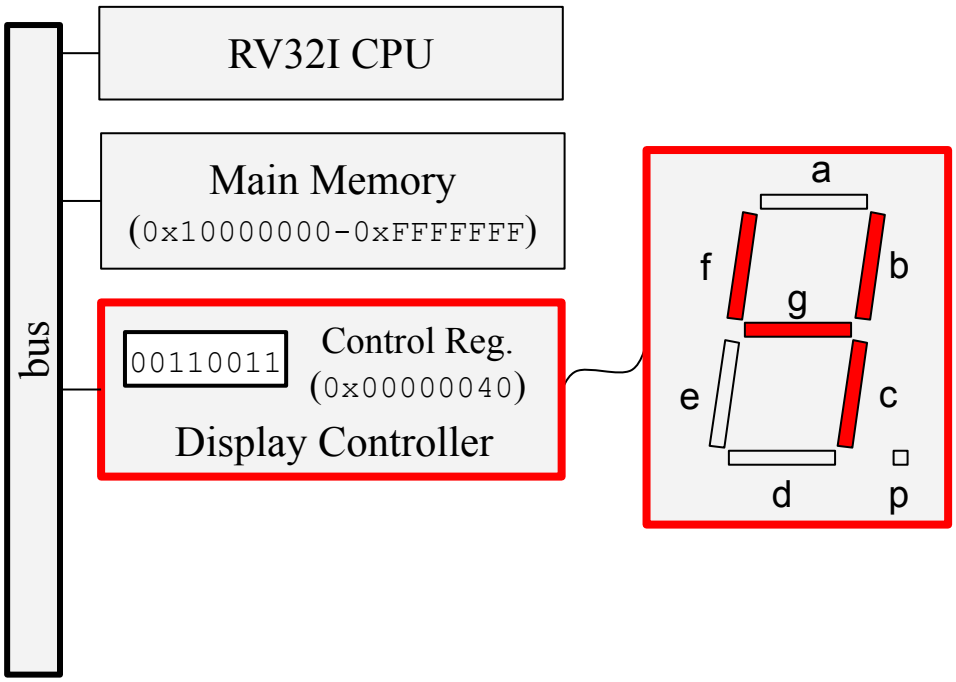
Leitura e escrita de dados em periféricos

Entrada e saída na arquitetura RV32I

Exemplo: Elevador

O sistema possui:

- Um processador RV32I e uma memória principal.
- Um *display* que mostra o andar para os(as) passageiro(as).



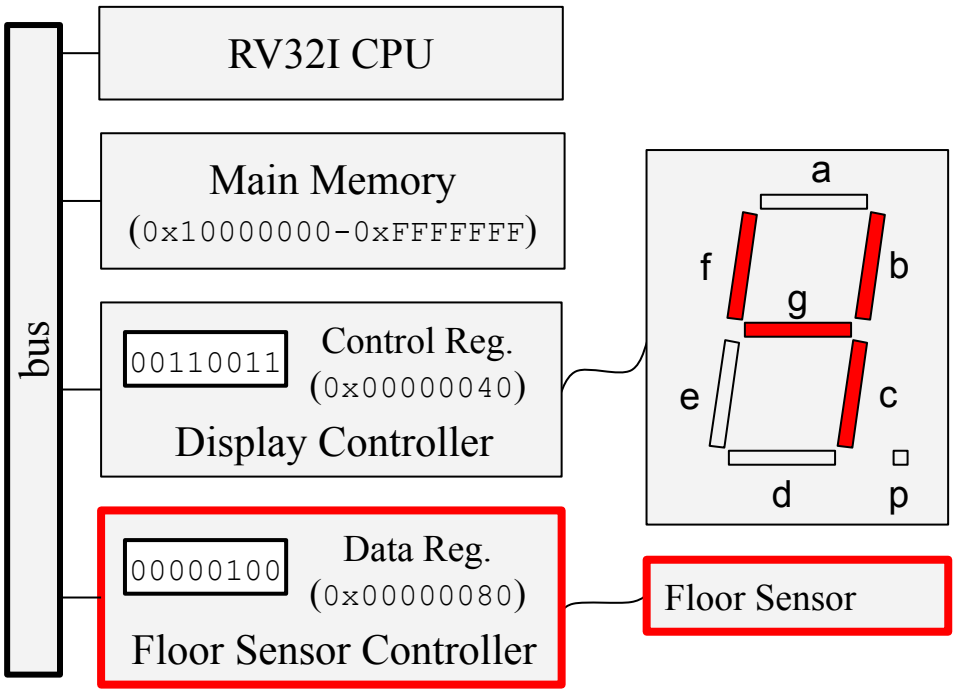
Leitura e escrita de dados em periféricos

Entrada e saída na arquitetura RV32I

Exemplo: Elevador

O sistema possui:

- Um processador RV32I e uma memória principal.
- Um *display* que mostra o andar para os(as) passageiro(as).
- Um sensor de andar (*Floor Sensor*) que detecta o andar onde o elevador se encontra.



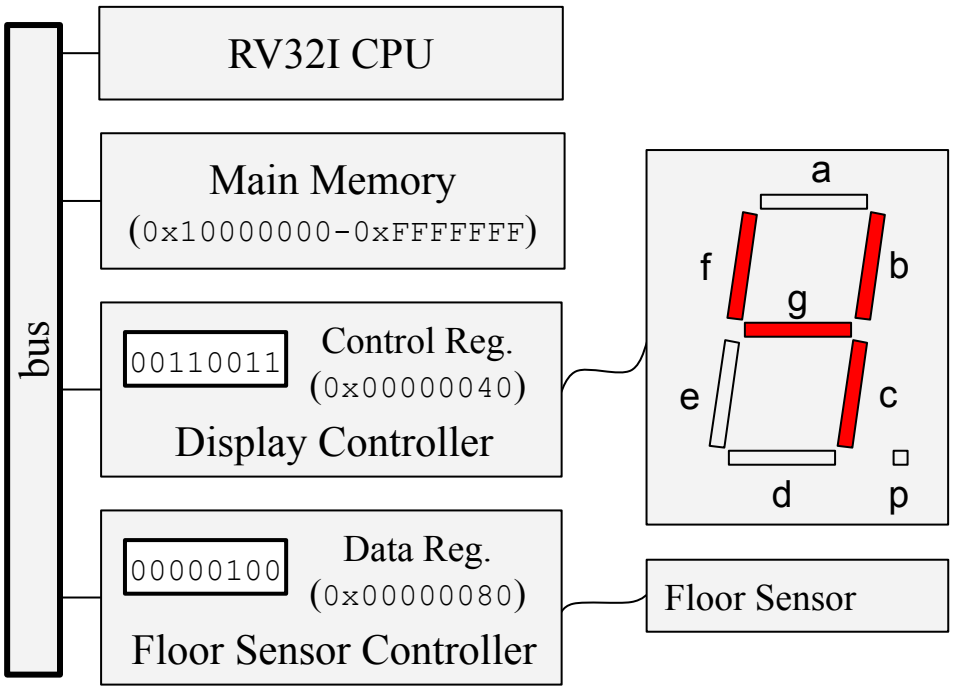
Leitura e escrita de dados em periféricos

Entrada e saída na arquitetura RV32I

Exemplo: Elevador

A rotina `update_display`

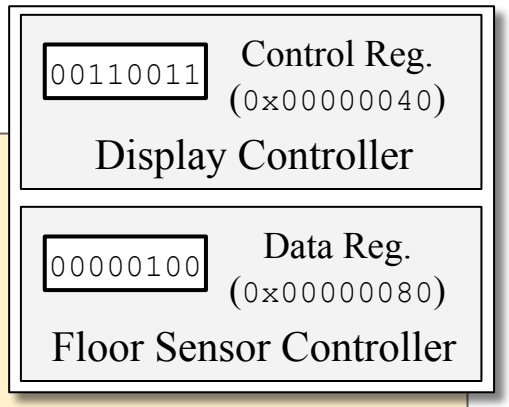
- Lê andar do *Floor Sensor* e atualiza o *display* com um padrão de segmentos que representa o número do andar.



Leitura e escrita de dados em periféricos

Entrada e saída na arquitetura RV32I

Exemplo: Elevador




```
.section .text
.set DISPLAY_CONTROL_REG_PORT, 0x00000040
.set FLOOR_DATA_REG_PORT, 0x00000080

update_display:
  li  a0, FLOOR_DATA_REG_PORT      # Reads the floor number and
  lb  a1, (a0)                     #  store into a1
  la  a0, floor_to_pattern_table   # Converts the floor number
  add t0, a0, a1                   #  into a configuration
  lb  a1, (t0)                     #  byte
  li  a0, DISPLAY_CONTROL_REG_PORT # Sets the display controller
  sb  a1, (a0)                     #  with the configuration byte
  ret                              # Returns

floor_to_pattern_table:
  .byte 0x7e,0x30,0x6d,0x79,0x33,0x5b,0x5f,0x70,0x7f,0x7b
```

Agenda

- Periféricos
- Conexão física
- Leitura e escrita de dados em periféricos
- **Técnica de espera ocupada (*Busy waiting*)**

Técnica de espera ocupada (Busy waiting)

***Busy waiting*, ou espera ocupada, é uma técnica de sincronização na qual o programa aguarda uma condição se tornar verdadeira verificando a condição repetidamente até que ela se torne verdadeira.**

- É implementada através de um laço que checa a condição a cada iteração. Caso ela seja falsa, o código salta para o início do laço e tenta novamente, caso contrário, ele sai do laço e segue a execução com o código após o laço.

Técnica de espera ocupada (Busy waiting)

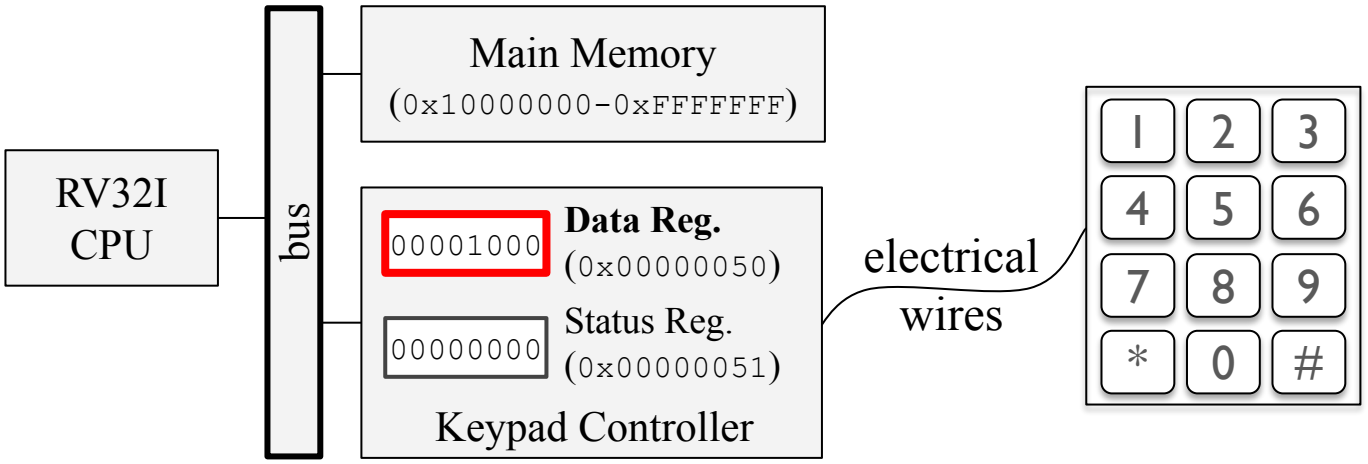
Em algumas situações, o programa deve esperar o periférico entrar em um determinado estado antes de realizar a operação de entrada ou saída.

- Nestes casos, o programador pode usar a técnica de **espera ocupada** para aguardar o periférico atingir este estado.

Técnica de espera ocupada (Busy waiting)

Exemplo: Teclado numérico (*keypad*)

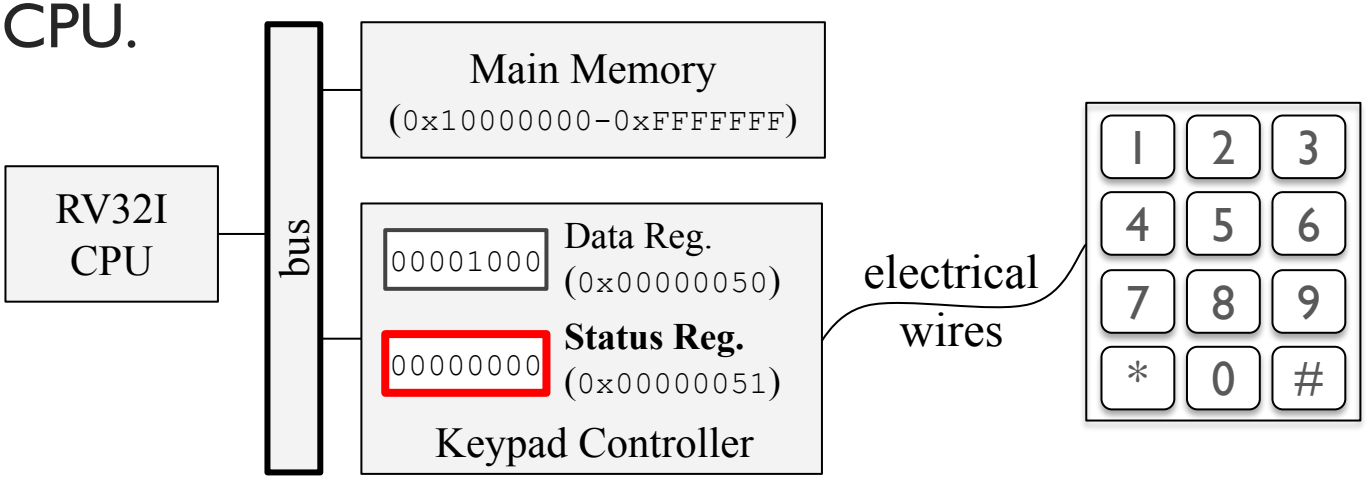
- **Data Register**: Registra a última tecla pressionada.



Técnica de espera ocupada (Busy waiting)

Exemplo: Teclado numérico (*keypad*)

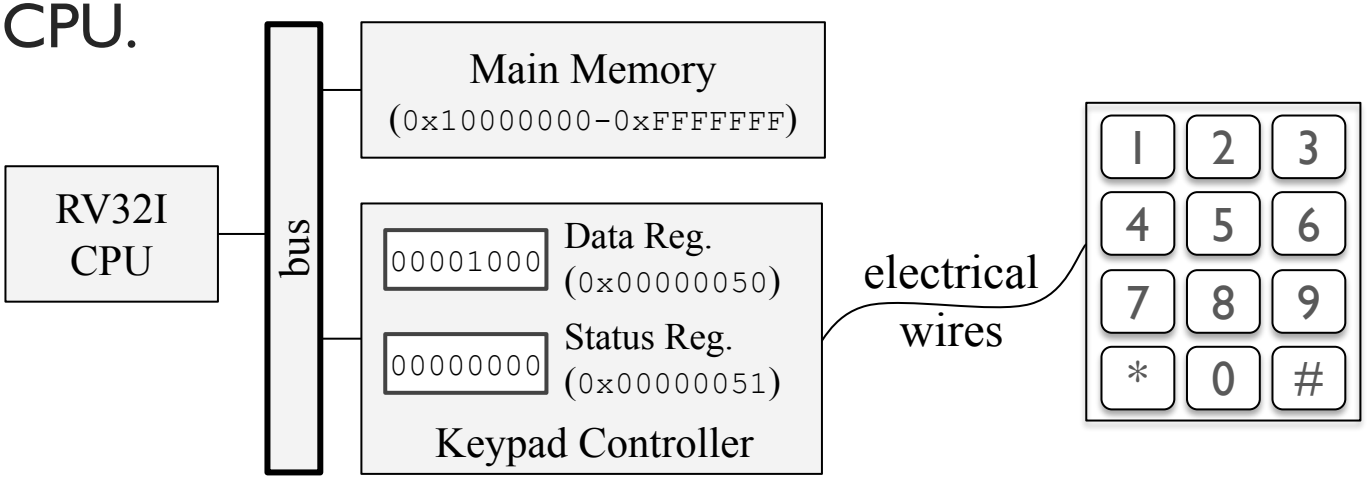
- Data Register: Registra a última tecla pressionada.
- **Status Register:**
  - Bit 0 (READY): Indica se alguma tecla foi pressionada desde a última vez que o Data Reg. foi lido pela CPU
  - Bit 1 (OVRN): Indica se o teclado foi pressionado mais de uma vez desde a última vez que o Data Reg. foi lido pela CPU.



Técnica de espera ocupada (Busy waiting)

Exemplo: Teclado numérico (*keypad*)

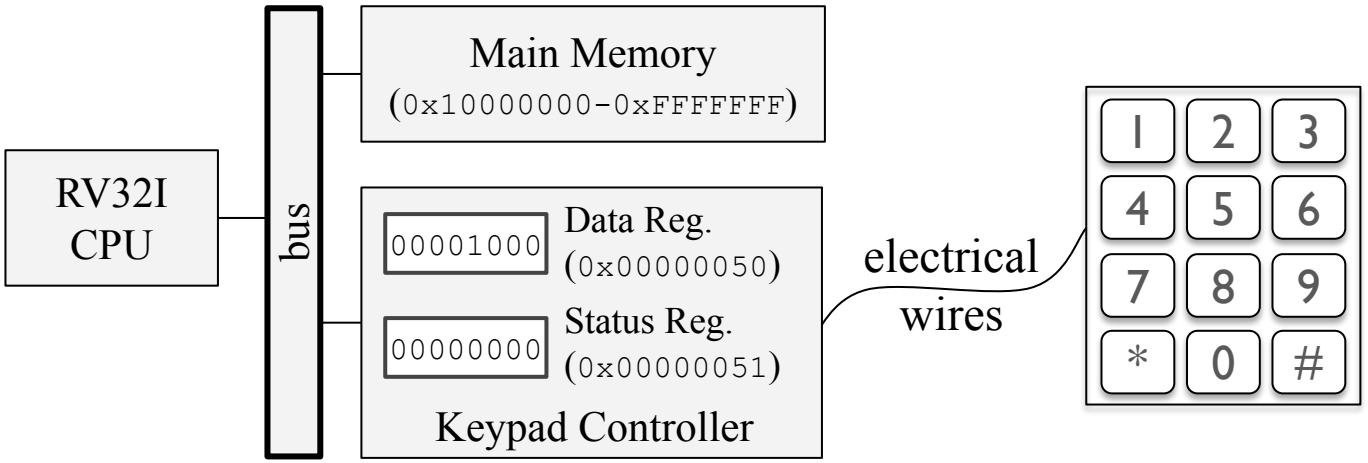
- Data Register: Registra a última tecla pressionada.
- Status Register:
  - Bit 0 (READY): Indica se alguma tecla foi pressionada desde a última vez que o Data Reg. foi lido pela CPU
  - Bit 1 (OVRN): Indica se o teclado foi pressionado mais de uma vez desde a última vez que o Data Reg. foi lido pela CPU.



Técnica de espera ocupada (Busy waiting)

Exemplo: Teclado numérico (*keypad*)

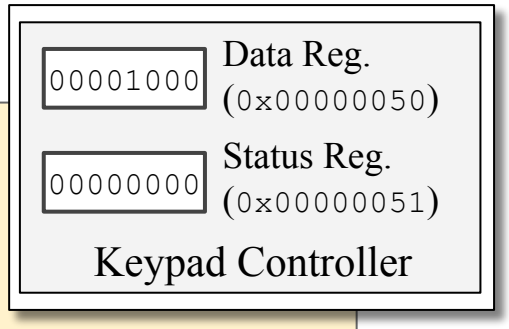
- Rotina `read_keypad`:
  - Retorna o valor da tecla que foi pressionada
    - Caso nenhuma tecla tenha sido pressionada desde a última leitura, deve esperar até que uma nova tecla seja pressionada.
    - Caso o teclado tenha sido pressionado mais de uma vez desde a última leitura, a rotina deve retornar o valor -1.



Técnica de espera ocupada (Busy waiting)

Exemplo: Teclado numérico (*keypad*)

- Rotina `read_keypad`:




```
.text
.set  DATA_REG_PORT, 0x00000050
.set  STAT_REG_PORT, 0x00000051
.set  READY_MASK,    0b00000001
.set  OVRN_MASK,     0b00000010
read_keypad:
  li   a0, STAT_REG_PORT  # Reads the keypad
  lb   a0, 0(a0)          #  status into a0
  andi t0, a0, READY_MASK # Check the READY bit and
  beqz t0, read_keypad    #  until it is equal to 1
  andi t0, a0, OVRN_MASK  # Check if OVRN bit and jump
  bnez t0, ovrn_occured   #  to ovrn_occured if equals to 1
  la   a0, DATA_REG_PORT  # Reads the key from the
  lb   a0, 0(a0)          #  data register into a0
  ret                     # Return
ovrn_occured:
  li   a0, -1             # Returns -1
  ret                     # Return
```