SEL-0415 Introdução à Organização de Computadores

SEL/EESC-USP Grupo de Sistemas Digitais

Dispositivos de Entrada e Saída

Aula 8

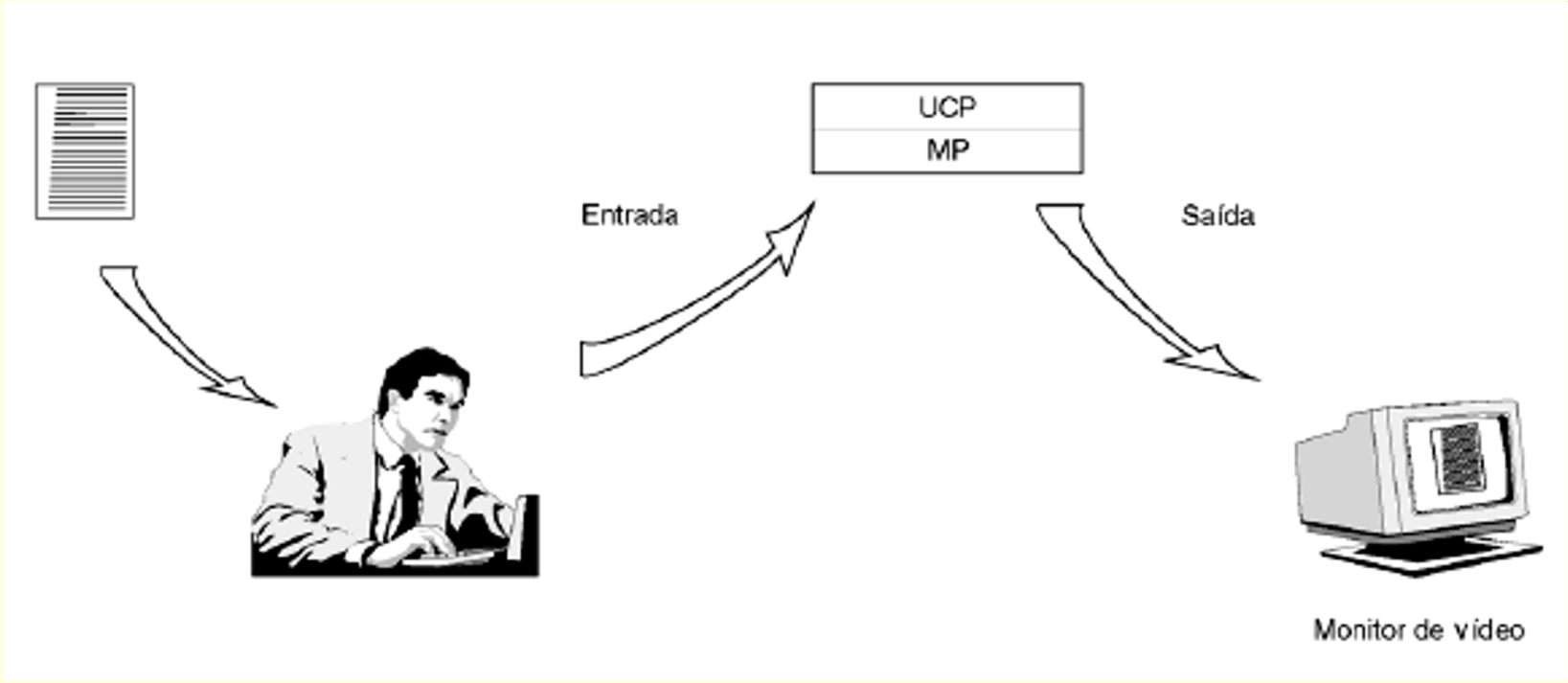
Prof. Dr. Marcelo Andrade da Costa Vieira

ENTRADA e SAÍDA (I/O - *Input/Output*)

- Inserção dos dados
- Apresentação dos resultados
- Comunicação Homem/Máquina

ENTRADA e SAÍDA (E/S)

(I/O - *Input/Output*)



ENTRADA e SAÍDA (E/S)
(I/O - *Input/Output*)

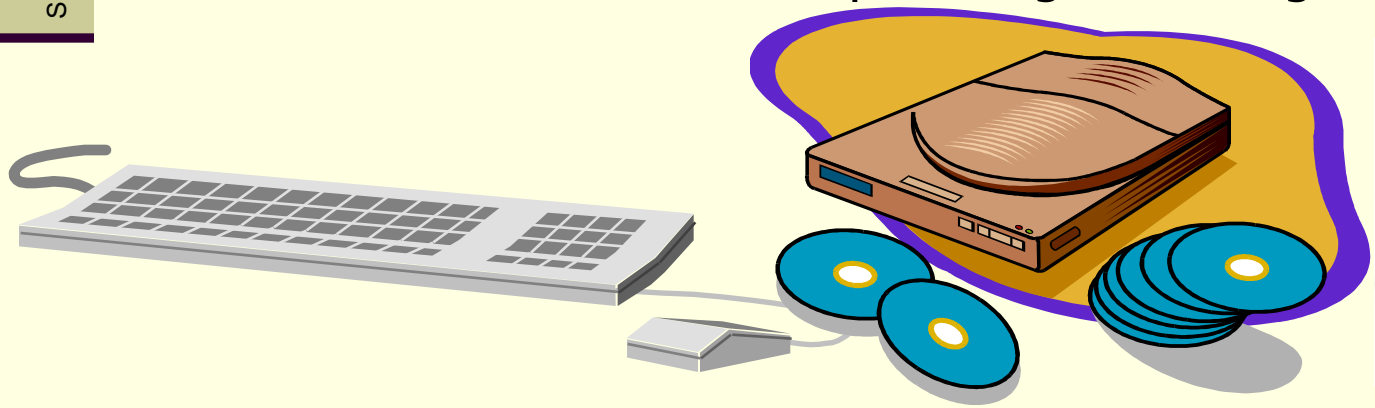
- **Entrada** ➡ Dispositivos (geralmente baseados em chaves) por onde informações entram na memória
  - Ex.: teclados, botões, mouse;

- **Saída** ➡ Dispositivos que mostram o resultado da operação executada
  - Ex: monitores, impressoras, memória secundária;

Dispositivos de Entrada

Periféricos

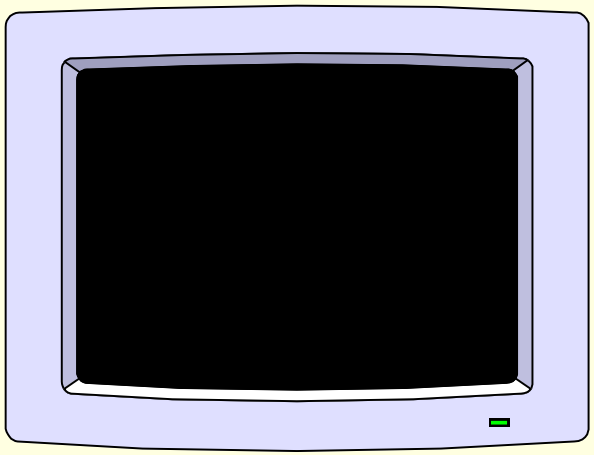
- Existem alguns que são especializados apenas em ENTRADA:
  - Teclado ➡ Lê os caracteres digitados pelo usuário
  - *MOUSE* ➡ Lê os movimentos e toque de botões
  - *Drive* de CD-ROM ➡ Lê dados de discos CD-ROM
  - Microfone ➡ Transmite sons para o computador
  - *Scanner* ➡ Usado para "digitalizar" figuras ou fotos

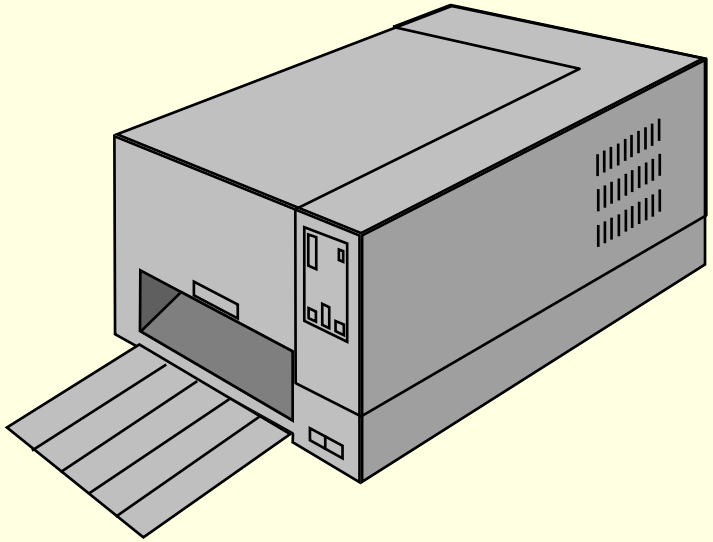


Dispositivos de Saída

Periféricos

- Outros especializados apenas em SAÍDA:
  - Vídeo ➡ Mostra ao usuário, na tela caracteres e gráficos
  - Impressora ➡ Imprime caracteres e gráficos
  - Alto-falante ➡ Realiza comunicação com o usuário através de som





Dispositivos de Entrada e Saída

Periféricos

- Outros em ENTRADA E SAÍDA
  - Disco rígido - Grava e lê dados
  - USB Flash Drive - Grava e lê dados em memória FLASH
  - MODEM - Transmite e recebe dados pela linha telefônica

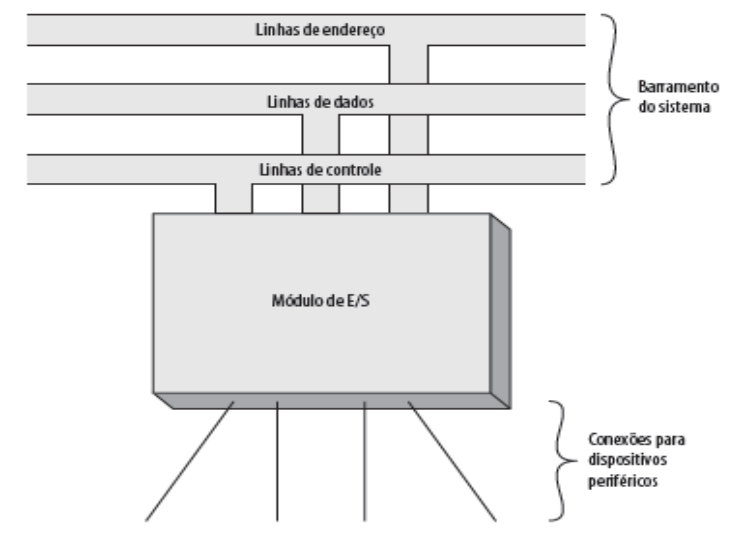
Dispositivos de Entrada e Saída para Controle de Processos

Periféricos

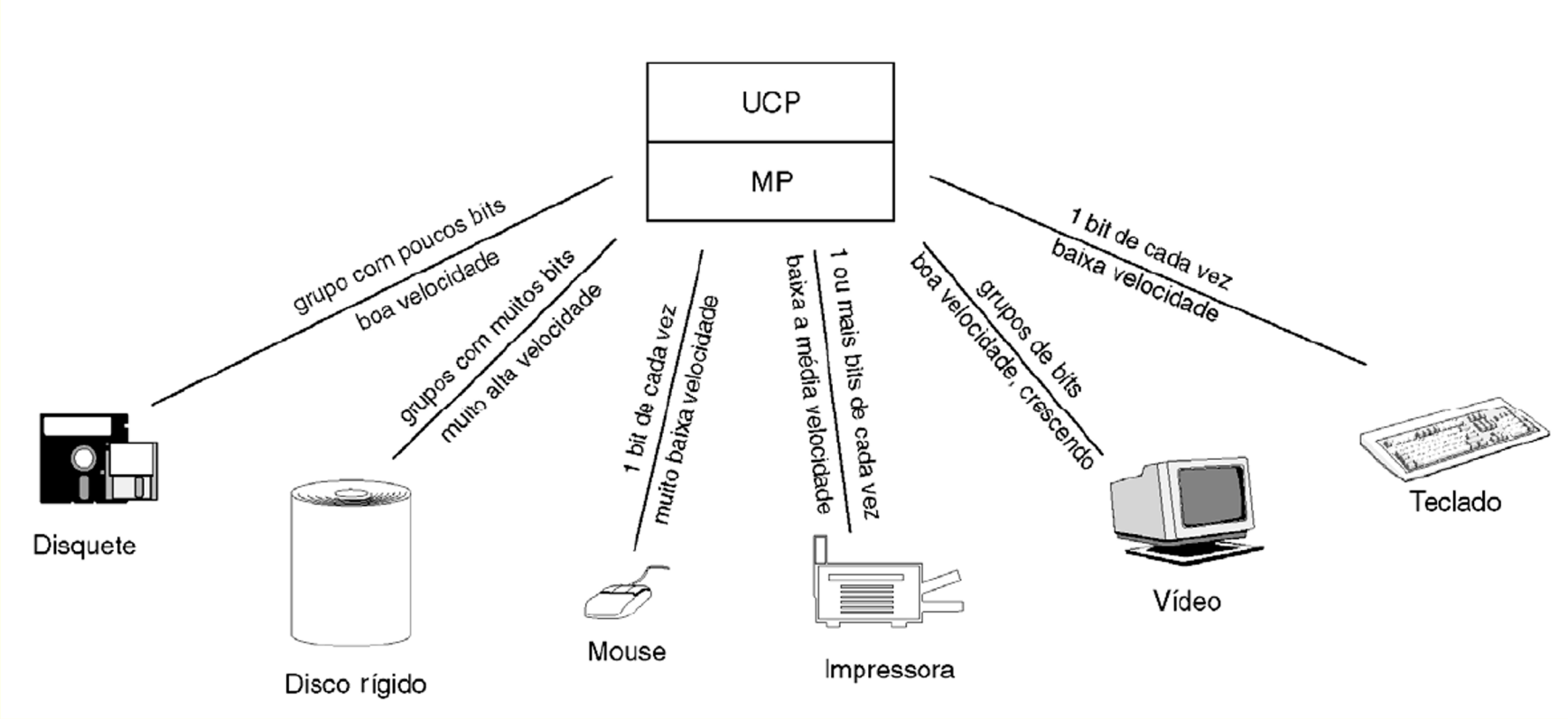
- Para sistemas embarcados
  - Sensores
  - Botões
  - Motores de Passo
  - Servomotores
  - Fotocélulas
  - Termostatos

Dispositivos de Entrada e Saída

- Grande variedade de periféricos:
  - Entregando diferentes quantidades de dados.
  - Em velocidades diferentes.
  - Em formatos diferentes.
- Todos mais lentos que a CPU e Memória RAM.
- Precisa de módulos (interfaces) de I/O.



Dispositivos de Entrada e Saída

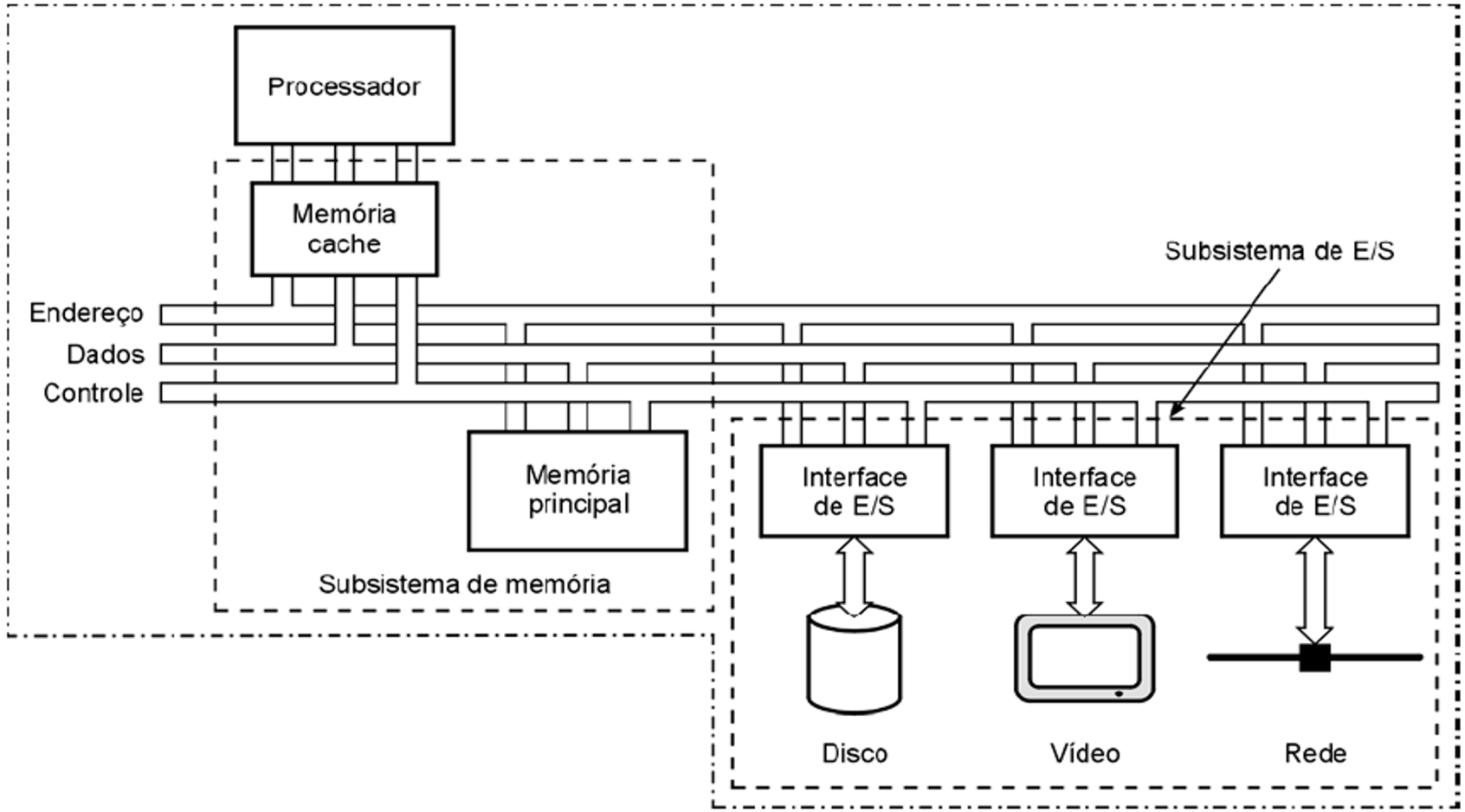


Interfaces de Entrada e Saída

Interfaces de (I/O)

- Geralmente a CPU não pode comunicar-se diretamente com os periféricos ⇨ a comunicação é feita com a ajuda de circuitos chamados de **Interfaces** ou **Módulos** de I/O
- Funções:
  - Presentes entre o barramento e o periférico
  - Compatibilidade entre os dispositivos e o µP
  - Controle da comunicação
  - Ex.: controlador de vídeo, controlador de disco, etc...

Interfaces de Entrada e Saída



Operações de I/O

Operações de I/O

Métodos para realização de operações de I/O

- Três tipos principais:
  - Varredura (*Pooling*)
  - Interrupção
  - Acesso Direto à Memória (DMA)

EXEMPLO FIGURATIVO

O FUNCIONÁRIO ESTÁ TRABALHANDO E TEM COMO FUNÇÃO RECEBER O RECADO DE QUEM LIGAR.

EXEMPLO FIGURATIVO

VARREDURA (telefone SEM campainha): o funcionário de tempos em tempos verifica se há alguém querendo lhe falar ao telefone;

INTERRUPÇÃO (telefone COM campainha): o funcionário apenas para de fazer o trabalho quando o telefone toca, pois há alguém querendo lhe falar ao telefone;

DMA - ACESSO DIRETO À MEMÓRIA (telefone COM campainha e COM secretária eletrônica): o telefone toca, a secretária eletrônica armazena o recado e o funcionário só para de fazer o trabalho quando lhe convier para ouvir o recado.

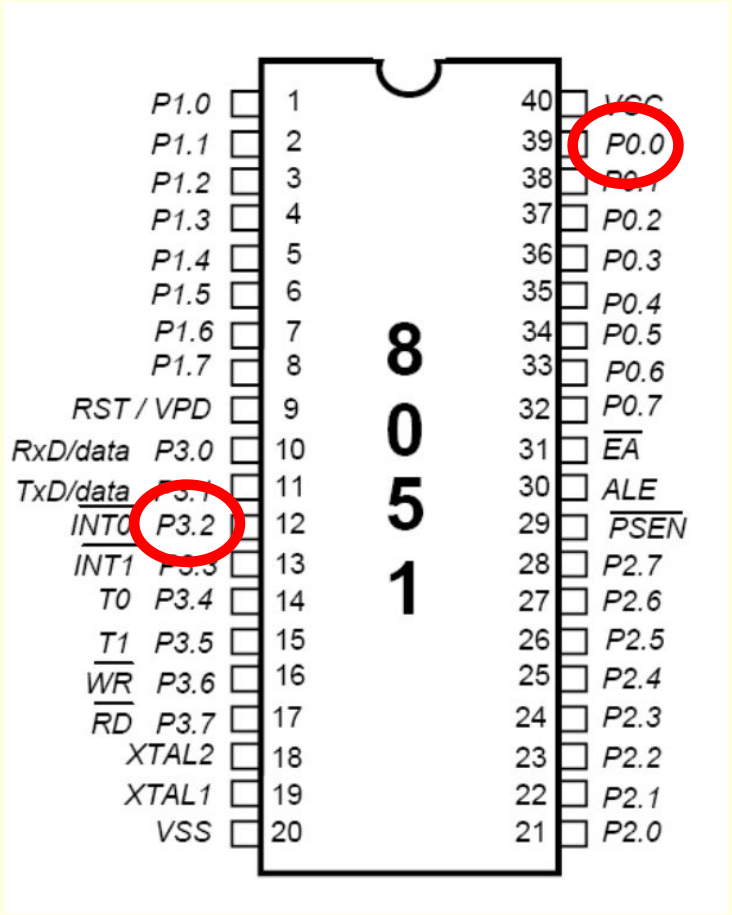
I/O por Varredura (*Pooling*)

- A CPU controla diretamente todas as etapas da comunicação com o dispositivo de I/O;
- O programa deve verificar os dispositivos de I/O de tempos em tempos e parar o processamento principal durante a comunicação;
- Deve-se criar uma sub-rotina para varredura e para atendimento de cada dispositivo de I/O;
- Processo muito pouco eficiente:
  - Gasta-se muito tempo verificando os dispositivos de I/O;
  - O atendimento ao dispositivo de I/O pode não ser imediato;

I/O por Varredura (*Pooling*)

Exemplo de programa para o 8051:

- O computador deve realizar a tarefa de mover dados da memória RAM interna;
- Ao mesmo tempo, deve verificar o estado de um botão conectado na porta P3.2 (varredura);
- Se o botão for apertado, deve-se ligar um LED conectado na porta P0.0;



I/O por Varredura (*Pooling*)

Exemplo de programa para o 8051:

Sub-rotina de varredura e de atendimento

```
                ORG 0000h
                SJMP PROG
VERIFICA:       JNB P3.2,VOLTA
                SETB P0.0
VOLTA:          RET
PROG:           MOV 10h,30h
                MOV 11h,31h
                ACALL VERIFICA
```

Verificação →

```
                MOV 12h,32h
                MOV 13h,33h
                ACALL VERIFICA
```

Verificação →

```
                MOV 14h,34h
                MOV 15h,35h
                ...
```

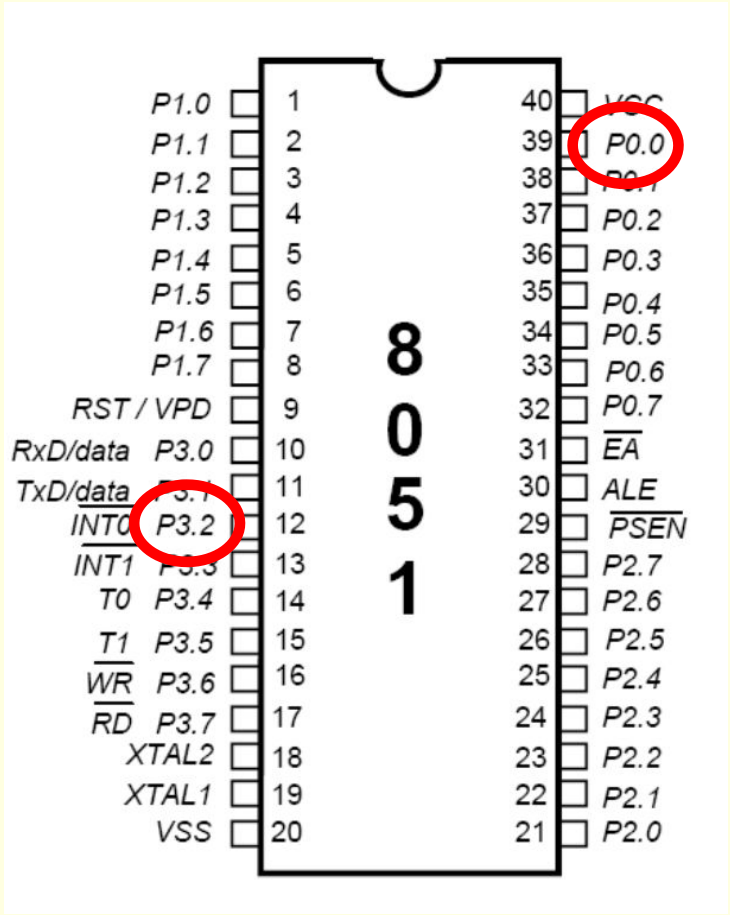
I/O por Interrupção

- A CPU aguarda a interface de I/O requisitar uma comunicação;
- O programa não precisa verificar os dispositivos de I/O de tempos em tempos;
- Não precisa criar uma sub-rotina de varredura para os dispositivos de I/O, já que a verificação é realizada automaticamente pelo *hardware*;
- Apenas deve-se criar uma sub-rotina para atendimento à interrupção de cada um dos dispositivos de I/O;
- Essa sub-rotina deve ser escrita em um endereço pré-definido na memória de programa.

I/O por Interrupção

Exemplo de programa para o 8051:

- O computador deve realizar a tarefa de mover dados da memória RAM interna;
- Ao mesmo tempo, deve verificar, por interrupção, o estado de um botão conectado na porta P3.2
- Se o botão for apertado, deve-se ligar um LED conectado na porta P0.0;



I/O por Interrupção

Exemplo de programa para o 8051:

```
        ORG 0000h
        SJMP PROG
        ORG 0003h
        SETB P0.0
        RETI
PROG:   MOV 10h,30h
        MOV 11h,31h
        MOV 12h,32h
        MOV 13h,33h
        MOV 14h,34h
        MOV 15h,35h
        ...
```

Sub-rotina de atendimento à interrupção

Não há a necessidade de verificação de I/O por *software*

Varredura x Interrupção

```
                ORG 0000h
                SJMP PROG
VERIFICA:       JNB P3.2,VOLTA
                SETB P0.0
VOLTA:          RET
PROG:           MOV 10h,30h
                MOV 11h,31h
                ACALL VERIFICA
                MOV 12h,32h
                MOV 13h,33h
                ACALL VERIFICA
                MOV 14h,34h
                MOV 15h,35h
                ...
```

```
                ORG 0000h
                SJMP PROG
                ORG 0003h
                SETB P0.0
                RETI
PROG:           MOV 10h,30h
                MOV 11h,31h
                MOV 12h,32h
                MOV 13h,33h
                MOV 14h,34h
                MOV 15h,35h
                ...
```

I/O por Interrupção

- Processo mais eficiente do que a varredura:
  - Enquanto a interrupção não ocorre, o µP pode realizar outras tarefas;
  - O µP só interrompe a tarefa atual quando ocorre a requisição de interrupção;
  - Não gasta-se tempo verificando os dispositivos de I/O;
  - O atendimento ao dispositivo de I/O é imediato;
- Pode ser externa ou interna:
  - Interna: divisão por zero, *overflow* de timer, etc.
  - Externa: interfaces de I/0, disparo de timer, etc.

I/O por Interrupção

Passo-a-passo de uma Interrupção:

1. Quando o evento ocorre, o µP altera um FLAG correspondente para sinalizar que existe uma requisição de interrupção;
2. Se o µP aceitar o pedido de interrupção (que pode ser configurado via *software*), o µP termina de executar a instrução atual e grava o endereço de retorno (**PC+1**)* em uma memória sequencial chamada PILHA (*stack*);
3. Em seguida, o µP desvia o programa (valor do registrador **PC**) para o endereço pré-definido na memória de programa para executar a rotina de atendimento à interrupção;

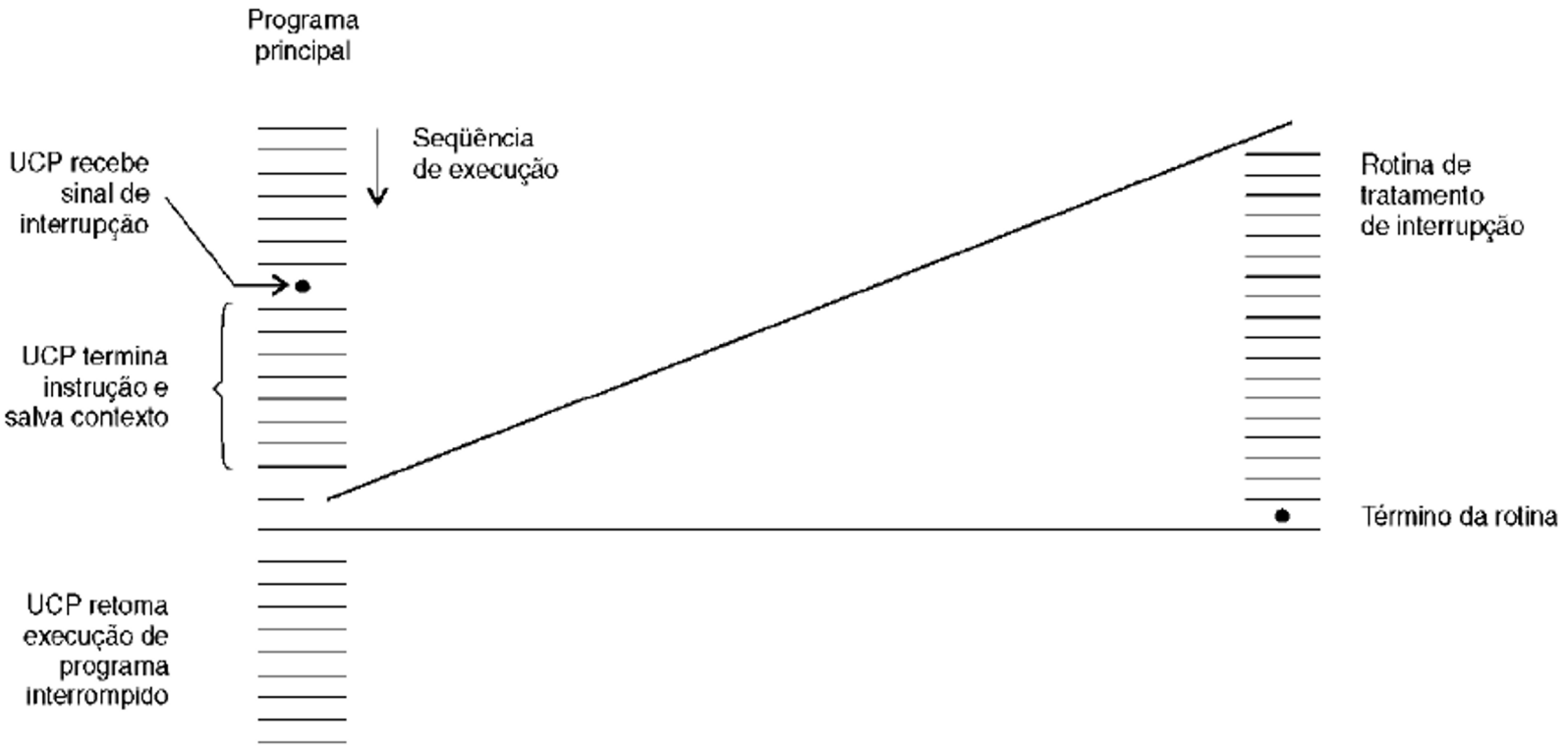
* PC = Registrador *Program Counter*

I/O por Interrupção

Passo-a-passo de uma Interrupção:

4. Após o término da execução da rotina de interrupção (instrução de retorno: RET, RETI, etc.), o µP volta ao programa principal no ponto onde parou, ou seja, o registrador **PC** recebe de volta o endereço que havia sido armazenado na PILHA;
5. Nem sempre é possível prever o local exato no programa onde ocorrerá o desvio para a sub-rotina de interrupção (evento assíncrono);
6. A varredura é um evento síncrono, pois o local exato do desvio para verificação está definido no programa.

I/O por Interrupção



I/O por DMA

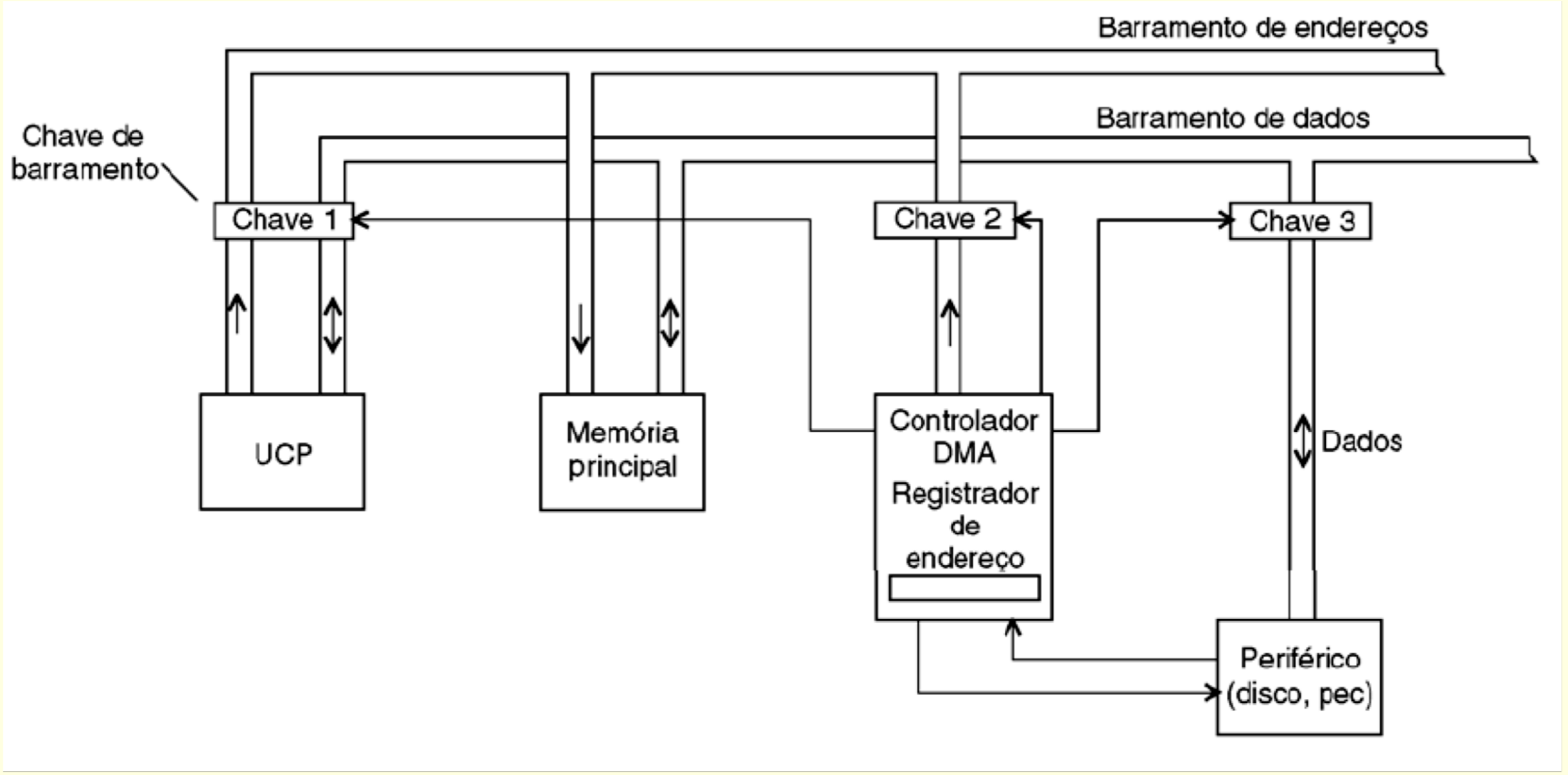
DMA (*Direct Memory Access*)

- Permite a movimentação de dados entre os dispositivos de I/O e a memória do computador sem envolver o microprocessador na transferência;
- Processo mais eficiente do que todos os outros, pois não utiliza o µP e não sobrecarrega o barramento.

DMA: ACESSO DIRETO À MEMÓRIA

- Dispositivo de *hardware* dedicado à operação de transferência de dados entre um dispositivo de I/O e a memória principal;
- Coloca a saída do microprocessador em estado de alta impedância (tri-state) para permitir a um dispositivo externo o Acesso Direto à Memória – *Bus Request*;
- Acesso direto à memória (DMA) permite uma forma mais rápida de mover dados entre as portas de I/O e a memória.

DMA: ACESSO DIRETO À MEMÓRIA

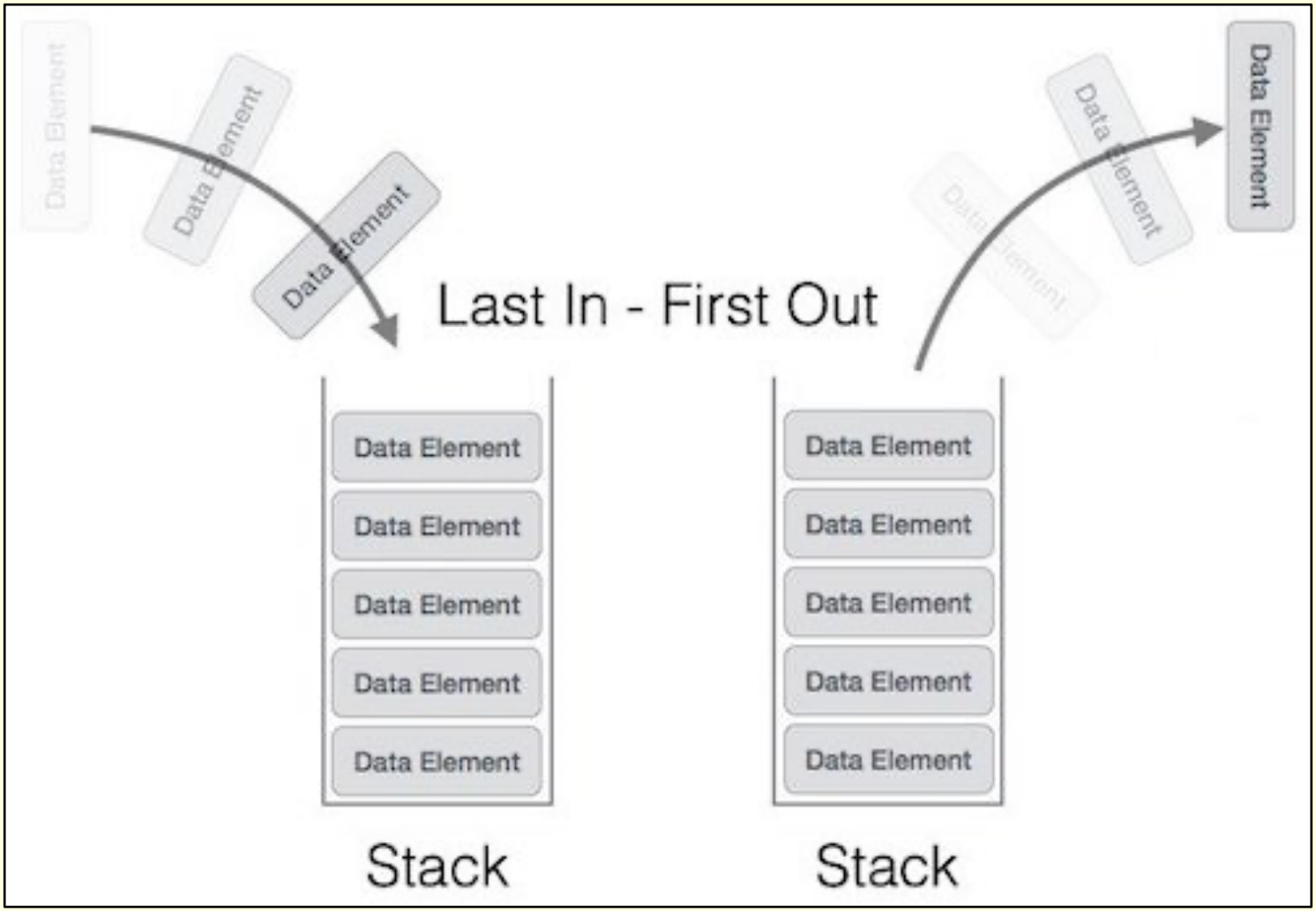


PILHA (*STACK*)

Pilha (*Stack*)

- Memória de escrita e leitura (RWM);
- Sequencial;
- Tipo LIFO ⇨ *Last in First Out*;
- Utilizada principalmente para armazenamento de endereço de retorno de uma sub-rotina de varredura ou de interrupção;
- A pilha também pode ser usada para armazenamento de dados temporários, utilizando as instruções `PUSH` e `POP`;
- Cada posição da pilha possui ***m*** bits ⇨ tamanho necessário para armazenar endereços da memória de programa (registrador *Program Counter* – PC).

Pilha (*Stack*)


Data Element
Data Element
Data Element
Last In - First Out
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Stack
Stack

Pilha (*Stack*)

- ✓ Uso mais importante ➡ armazenar endereços de retorno de sub-rotinas de varredura ou interrupção:
- ✓ Instrução de varredura: ACALL, LCALL ou uma Interrupção ➡ O programa principal é desviado para o endereço de início da sub-rotina;
- ✓ Instrução de retorno: RET, RETI ➡ última instrução da sub-rotina;

Guarda automaticamente o endereço de retorno na pilha (PC+1) antes de desviar para a sub-rotina

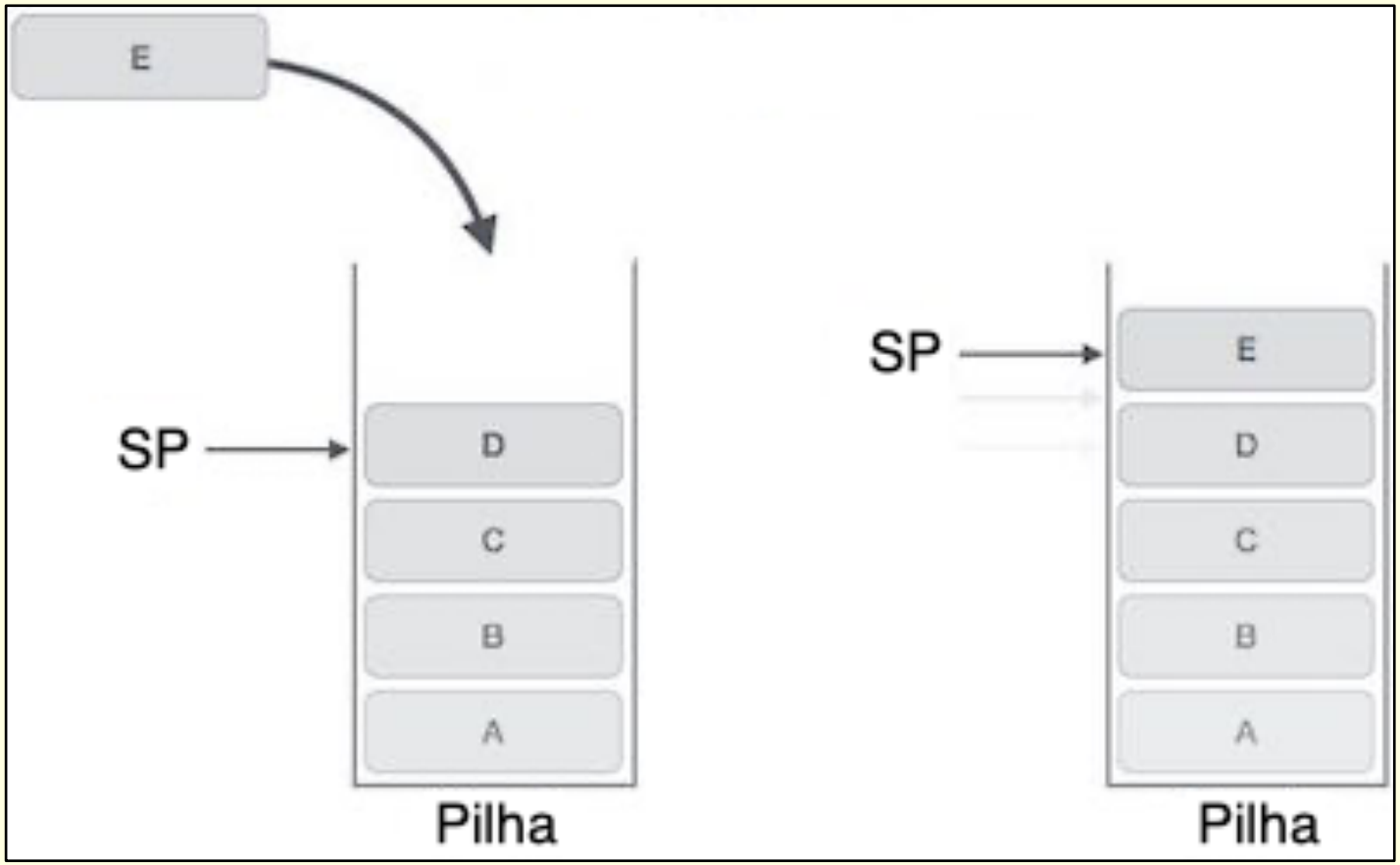
Resgata da pilha o endereço de retorno e salva no registrador PC (*program counter*)

Stack Pointer (SP)

- Registrador *Stack Pointer* (Ponteiro de Pilha):
  - O **SP** aponta para o último endereço da pilha (topo da pilha) e é incrementado cada vez que um endereço ou dado é armazenado na pilha;
  - O **SP** é decrementado cada vez que um endereço ou dado é lido na pilha;
  - O **SP** garante que os dados sejam escritos ou lidos sequencialmente na pilha;
  - O **SP** tem largura de ***n*** bits ⇨ o qual define o tamanho máximo da pilha (número máximo de endereços ou dados que ela consegue armazenar).

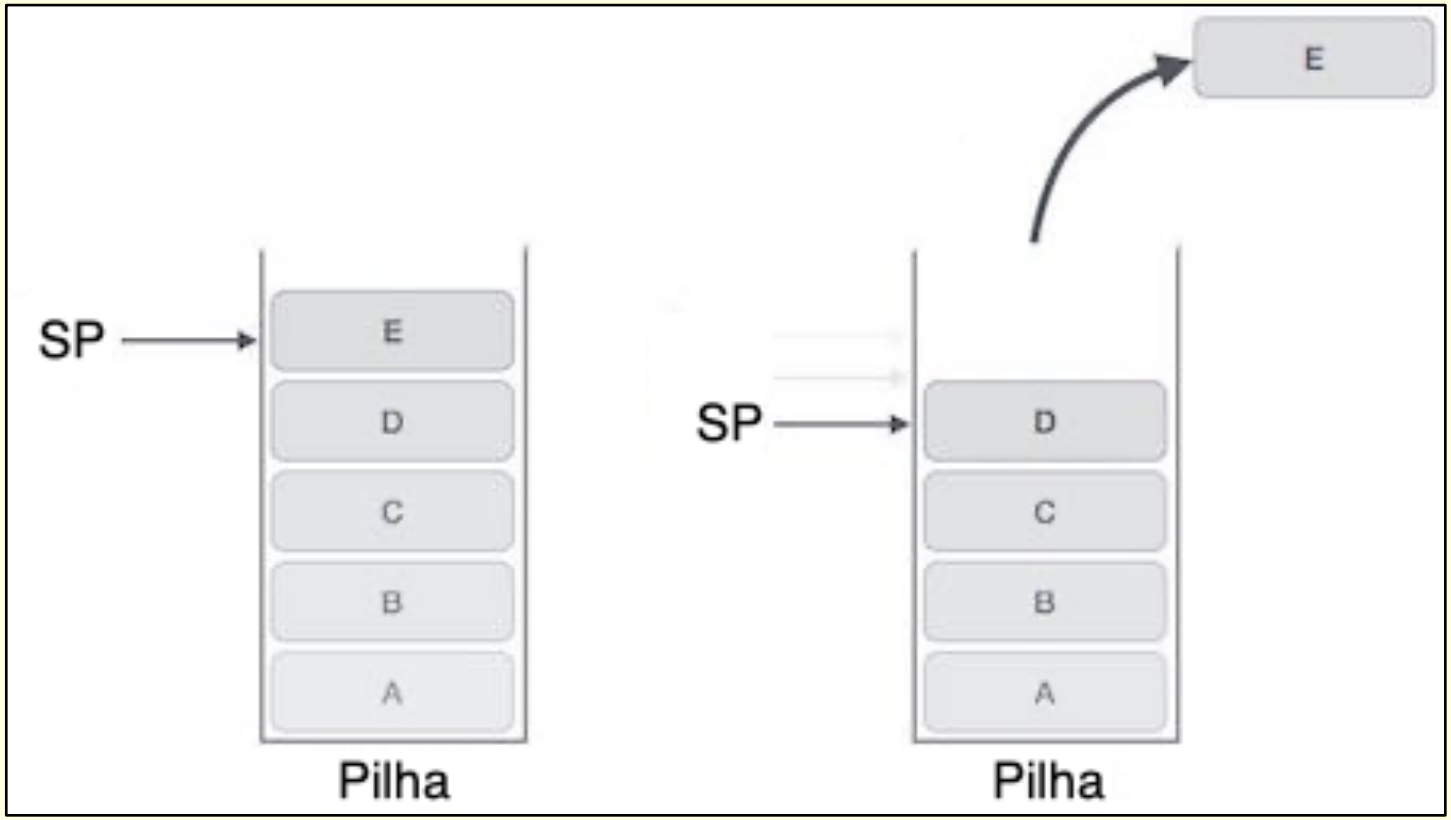
Pilha (*Stack*)

Após uma chamada de sub-rotina de varredura (instrução `ACALL, LCALL`) ou uma interrupção ou uma instrução `PUSH`:

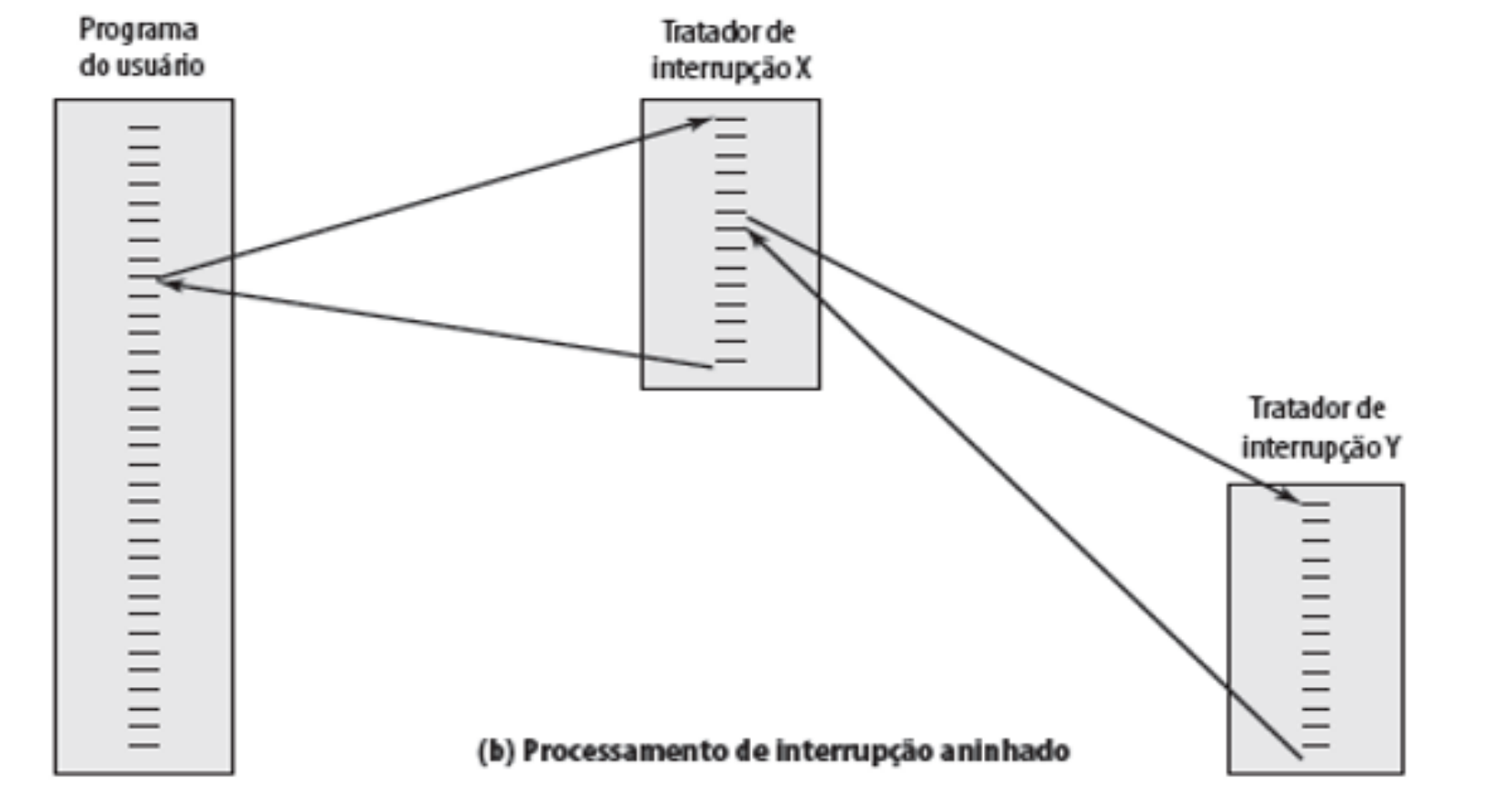


Pilha (*Stack*)

Término de execução de uma sub-rotina de varredura ou interrupção (instrução de retorno: `RET, RETI`) ou após uma instrução `POP`:



Pilha (*Stack*)


Programa
do usuário
Tratador de
interrupção X
Tratador de
interrupção Y
(b) Processamento de interrupção aninhado

Exemplo de Funcionamento da Pilha

Memória ROM

	End.	INSTRUÇÃO
PC →	*End. de Reset*	00h	Salta para o Programa Principal
		01h	
		02h	
PC →	*End. da Sub-rotina de interrupção*	03h	Inicio da Sub-rotina de interrupção
PC →		04h	
PC →		05h	
PC →		06h	
PC →		07h	Fim da Sub-rotina de interrupção
PC →	*End. da Sub-rotina de varredura*	08h	Inicio da Sub-rotina de varredura
PC →		09h	
PC →		0Ah	
PC →		0Bh	
PC →		0Ch	
PC →		0Dh	
PC →		0Eh	
PC →		0Fh	Fim da Sub-rotina de varredura
PC →	*End. do Programa Principal*	10h	Inicio do Programa Principal
PC →		11h	
PC →		12h	ACALL Varredura
PC →		13h	
PC →		14h	
PC →		15h	Fim do Programa Principal

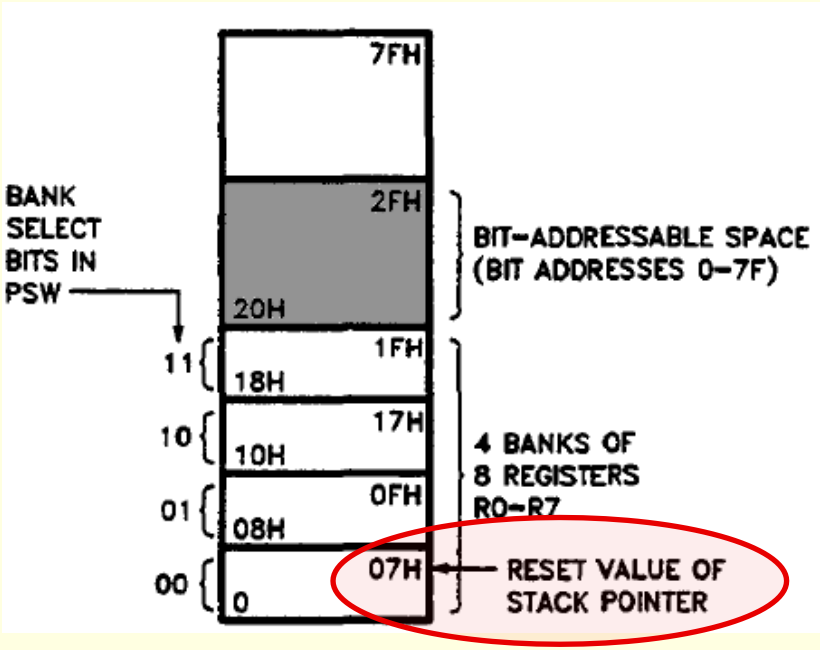
INTERRUPÇÃO

Pilha

SP →	0Ch
SP →	13h
SP →	

Pilha (*Stack*)

- Alguns computadores usam uma área da memória de dados (RAM) para servir como pilha;
- Nesse caso, a pilha pode ser alocada para qualquer área na RAM interna, carregando-se o endereço adequado no registrador **SP**;
- Para o μC 8051:



FIM