Tópicos sobre Arquitectura de Computadores -- 4

José A. Cardoso e Cunha
DI-FCT/UNL

1. Arquitectura de Von Neumann
2. Programação em linguagem ´´máquina´´
3. Sistema de entradas e saídas
4. Hierarquia das unidades de memória
5. Organização interna do processador

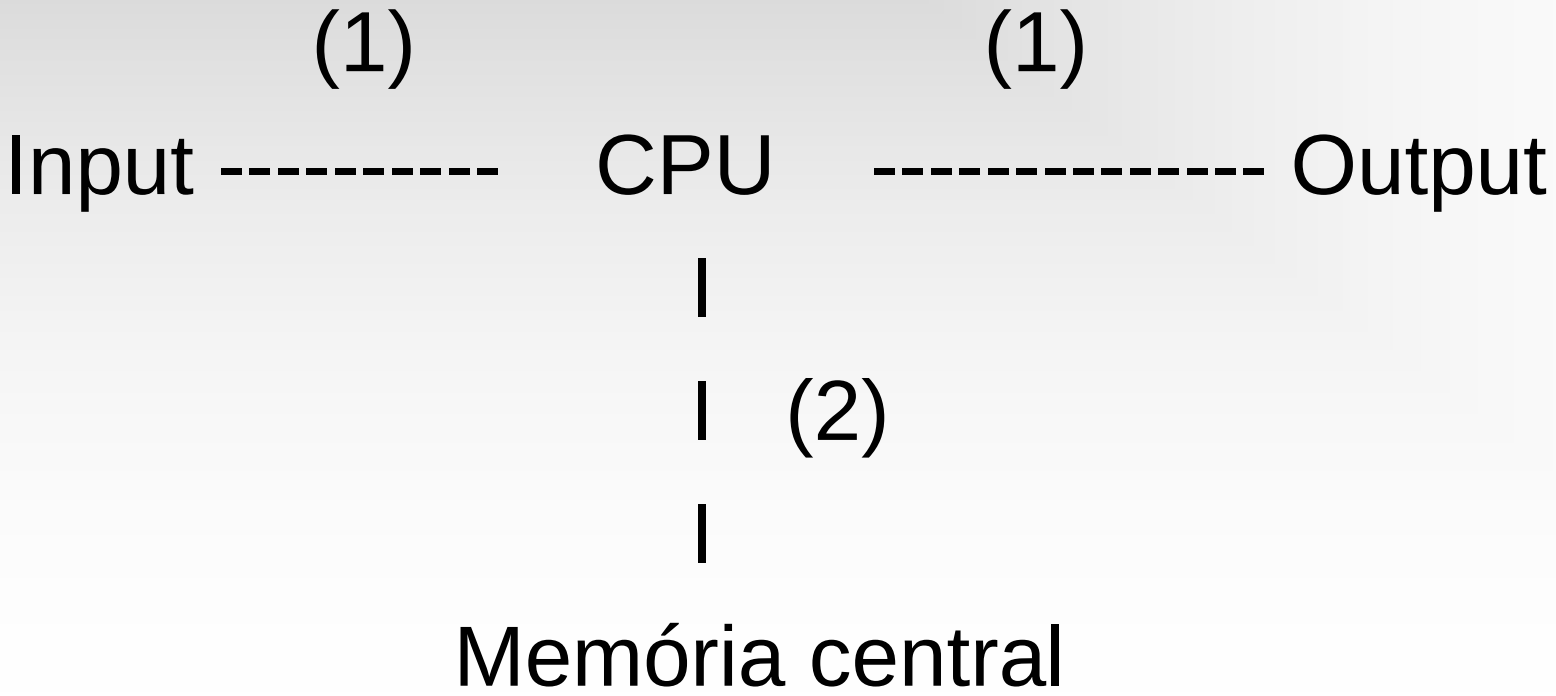
Organização do sistema de entradas e saídas

Características da arquitectura

Operações dentro do CPU: muito rápidas

Interacções CPU – Memória: menos rápidas

Interacções com Periféricos: muito lentas



Dispositivos de input/output

Orientados para caracteres (bytes)

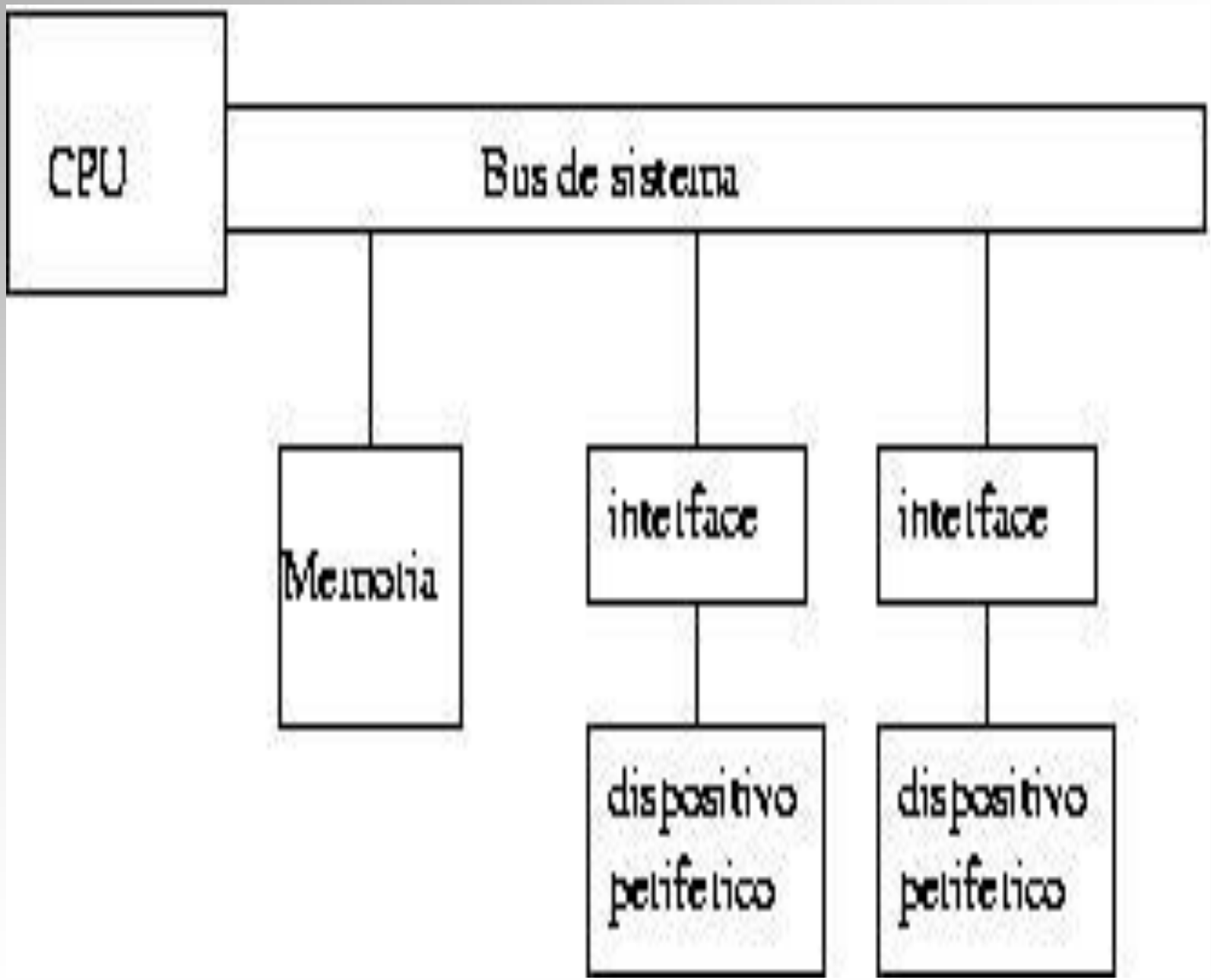
- Fluxos de bytes (streams) sem qualquer estrutura
- Produzem ou consomem fluxos de bytes
  - Teclado
  - Écran
  - leitor/perfurador de fita
  - interface de comunicação

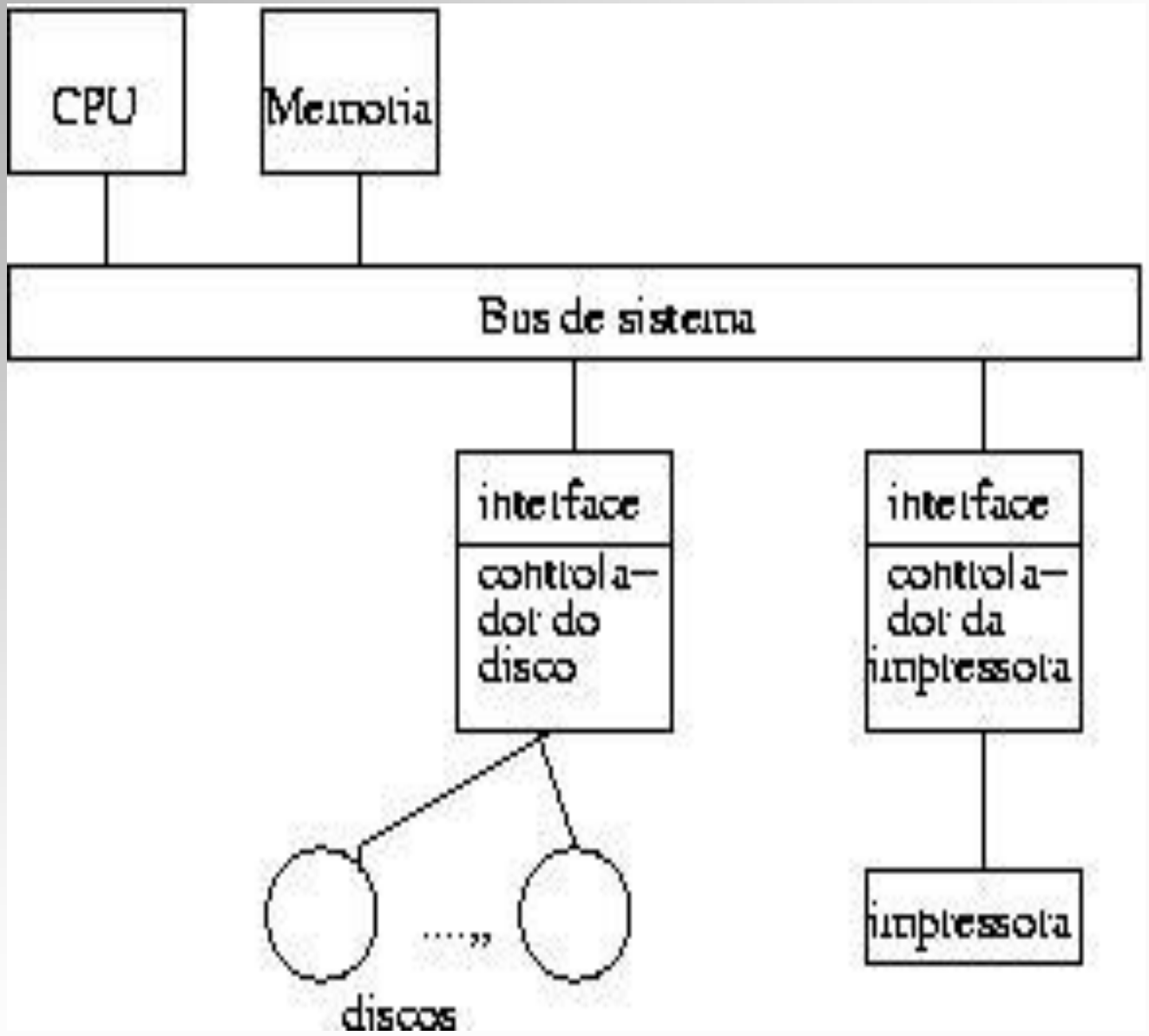
Orientados para blocos de bytes

- Guardam blocos de dimensão fixa (1K-2K bytes)
- Cada bloco é individualmente endereçado
- Cada bloco pode ser lido/escrito independentemente dos outros (acesso directo)
  - Discos
  - bandas mangnéticas

Controladores dos periféricos

- Circuitos electrónicos que controlam a operação do periférico
- Definem a interface de I/0 ´´´vista´´ pelo CPU e acessível aos programas ´´assembly´´


CPU
Bus de sistema
Memoria
inteiface
inteiface
dispositivo
periferico
dispositivo
periferico


CPU
Memoria
Bus de sistema
interface
controla-
dor do
disco
interface
controla-
dor da
impressora
discos
impressora

Os periféricos são controlados pelo CPU via unidades funcionais dedicadas: interfaces

A interface do periférico é vista pelo CPU como um conjunto de registadores

Exemplos:

Teclado: ligado via o controlador do teclado

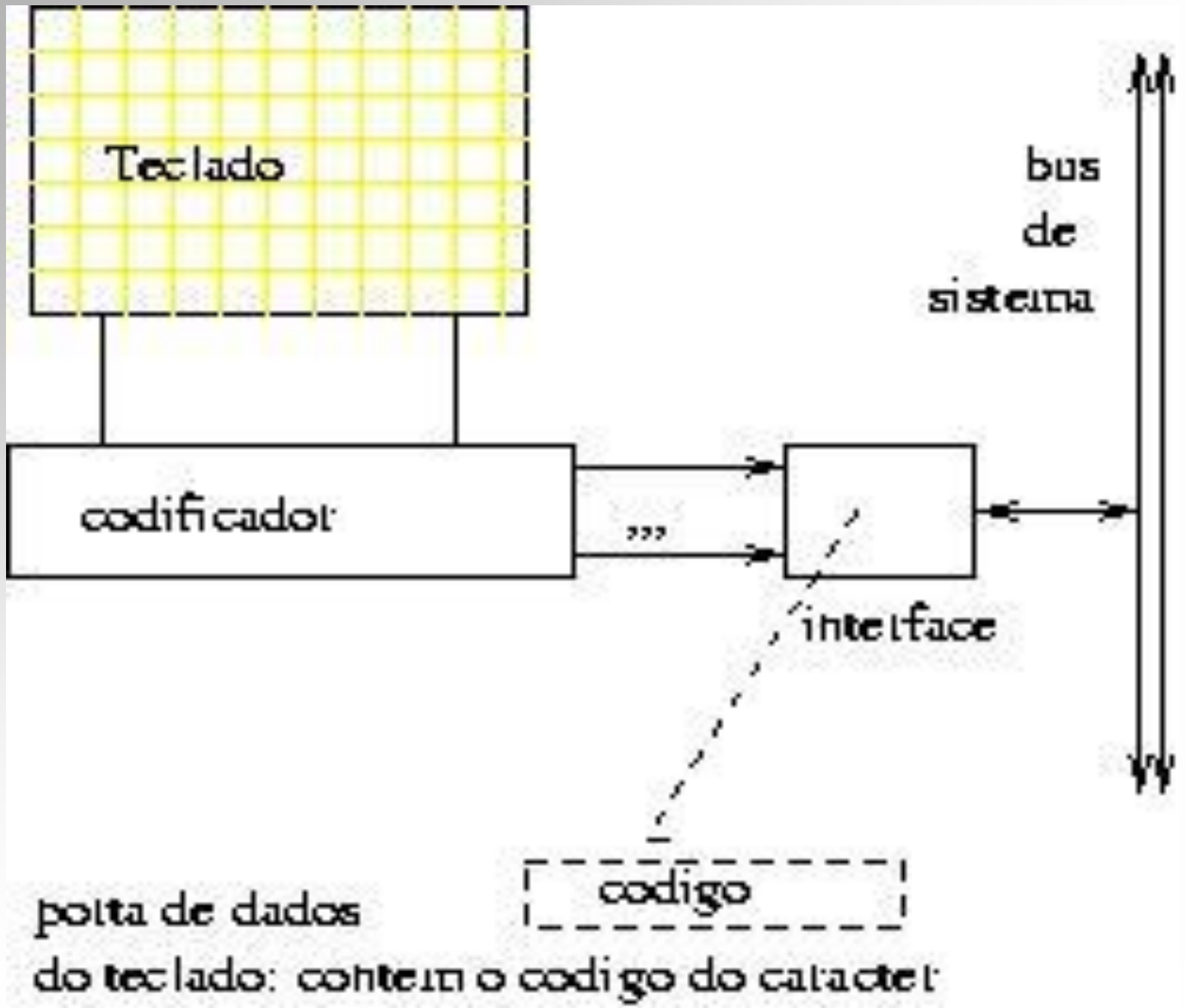
Écran: ligado via o adaptador de écran:

O CPU envia bytes; o controlador converte-os em sinais para o écran, que é visto como parametrizado por:

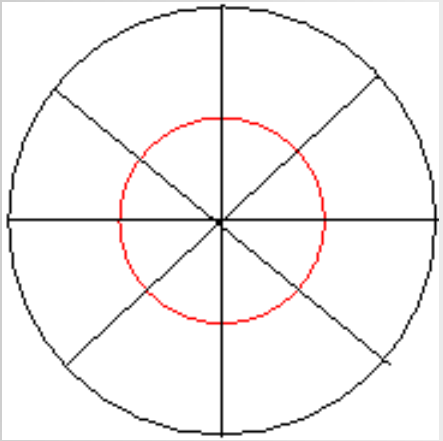
nº de caracteres / linha

nº de linhas / écran

nº de pixels / nº de pixels


Teclado
bus
de
sistema
codificador
interface
código
porta de dados
do teclado: contem o codigo do caracter

Disco: a unidade física manipula sequências de bits em série



Quando formatado, passa a indicar informação sobre o seu conteúdo e formato

- nº de cilindros ou pistas → nº de bloco
- nº de sector

O controlador converte a sequência de bits em blocos de bytes tal que o CPU vê um formato de n sectores, cada com x bytes por pista

Aceita comandos para:

read, write, seek, .. Endereçando blocos

O CPU envia comandos para a interface e fica a aguardar a confirmação de que o comando foi completado, para poder obter resultados da interface.

Acesso ao disco: -- meio de memória secundária (não volátil)

Interfaces:

-- de baixo nível, definida pelas instruções máquina do CPU que acedem directamente aos registadores da interface (portas de I/O) para:

posicionar a cabeça de leitura/escrita na pista desejada

transferir dados

inspeccionar o estado da interface

-- de alto nível, definida pelos programas do Sistema de Operação:

a) ao nível de rotinas (gravadas em memória permanente, só acessível para leitura (ROM – read-only memory))

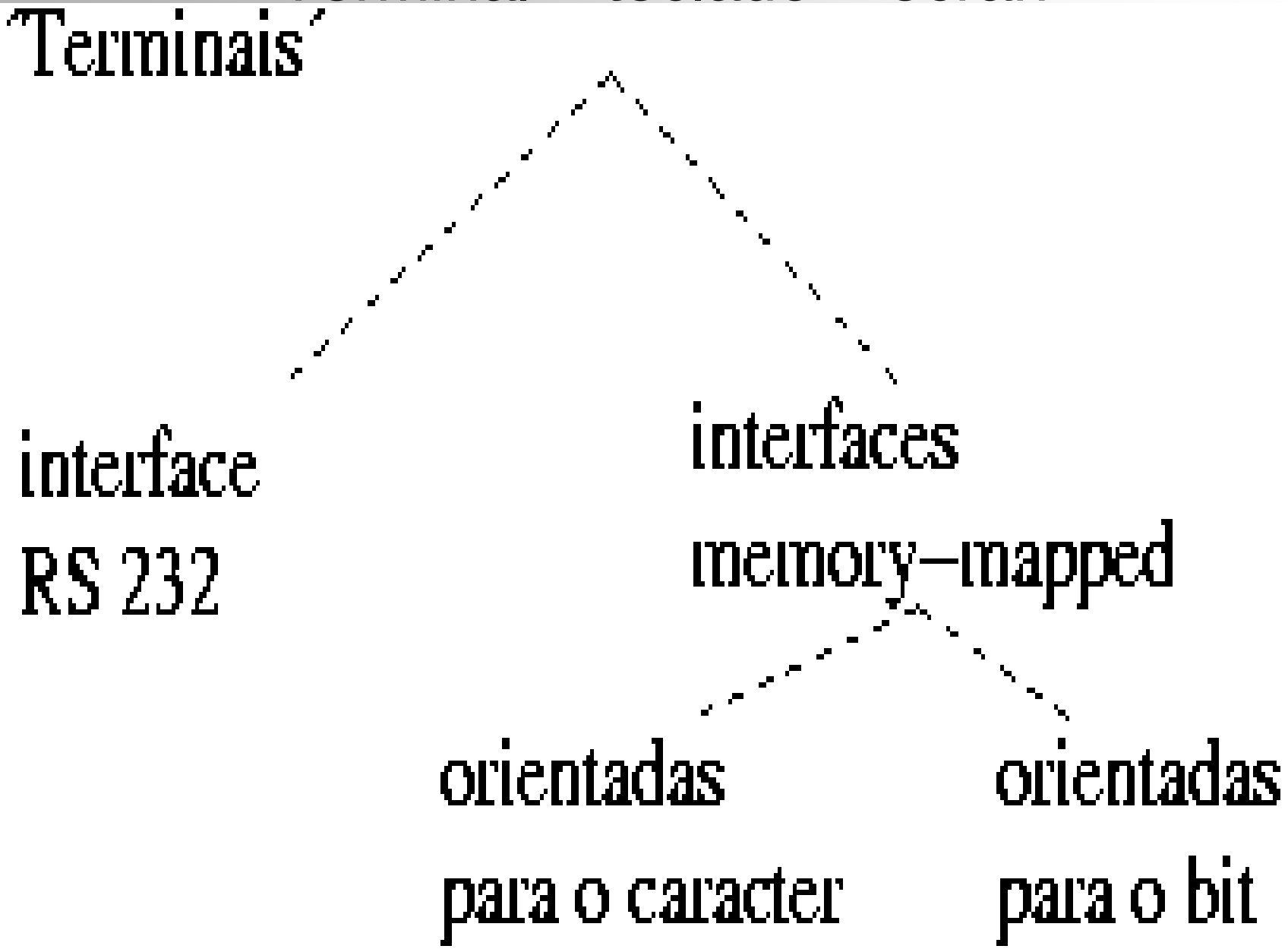
e reunidas no módulo Basic Input-Output System (BIOS)

dão acesso a blocos de bytes

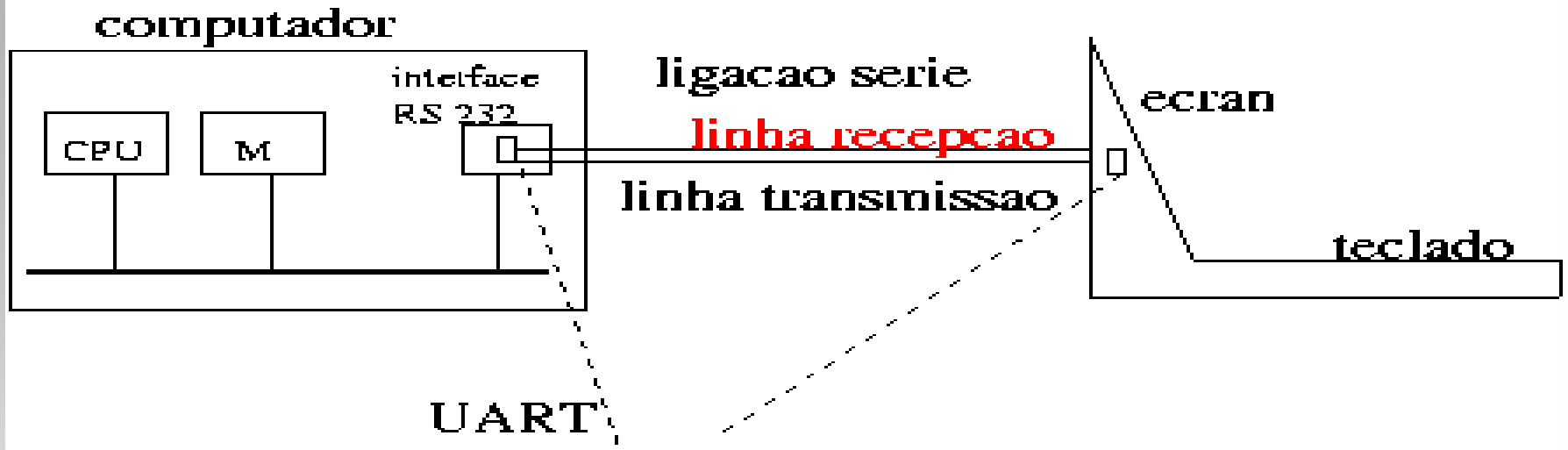
b) ao nível do Sistema de Ficheiros do SO

apresentam os dados, organizados em unidades lógicas de dimensão variável e formadas por sequências de bytes, escondendo a estrutura de blocos aos programas utilizadores

Terminal = teclado + écran

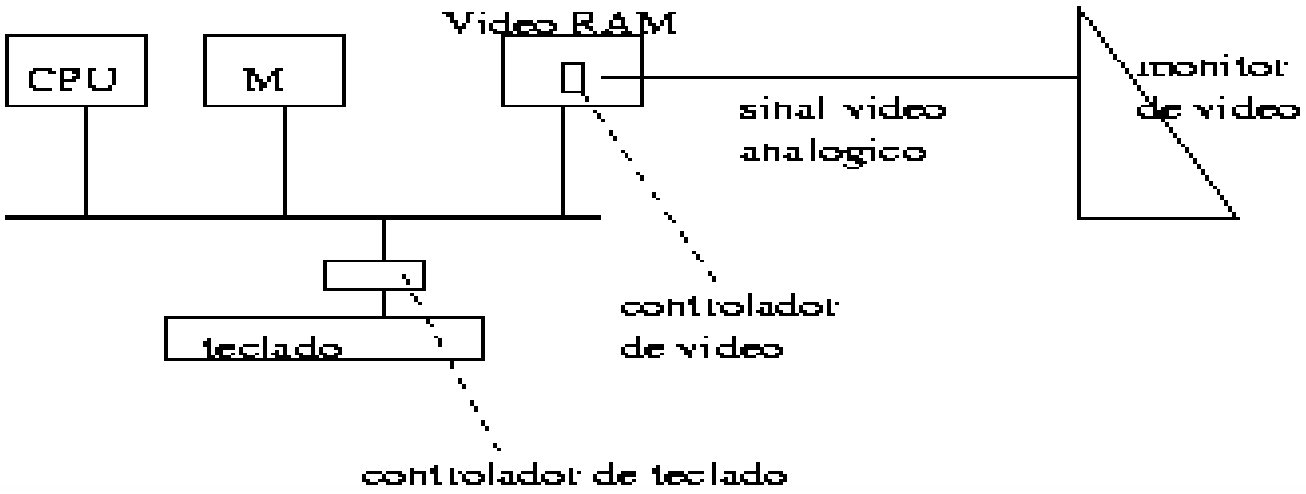


a) Terminais RS 232 antigos



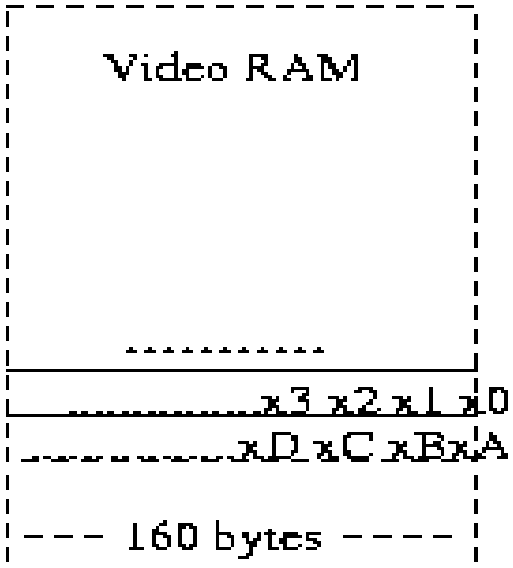
(Universal Asynchronous Receiver and Transmiter)

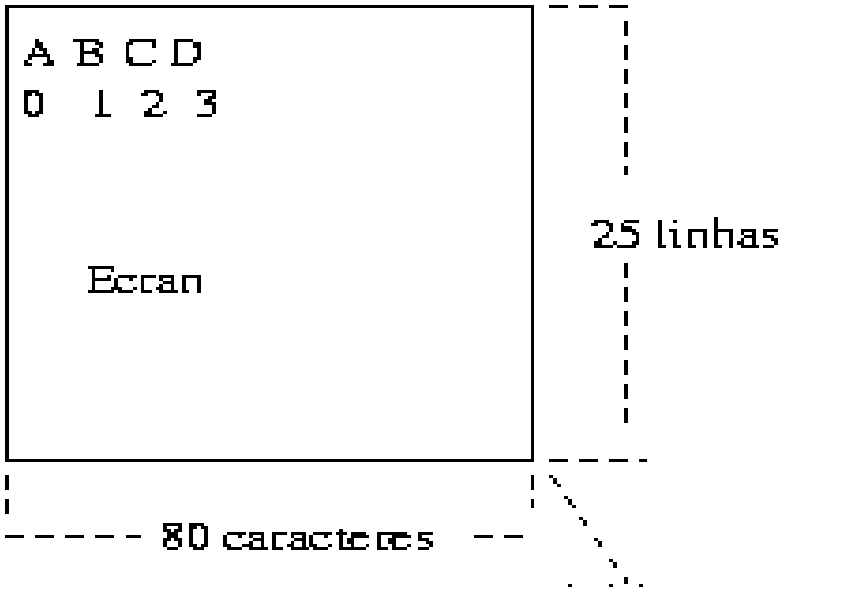
b) Monitores Memory–mapped



Video RAM: uma memoria especial fazendo parte do espaco de enderecos visto pelo CPU

Controlador de Video: converte os bytes colocados na Video RAM em sinais de video para controlar o monitor



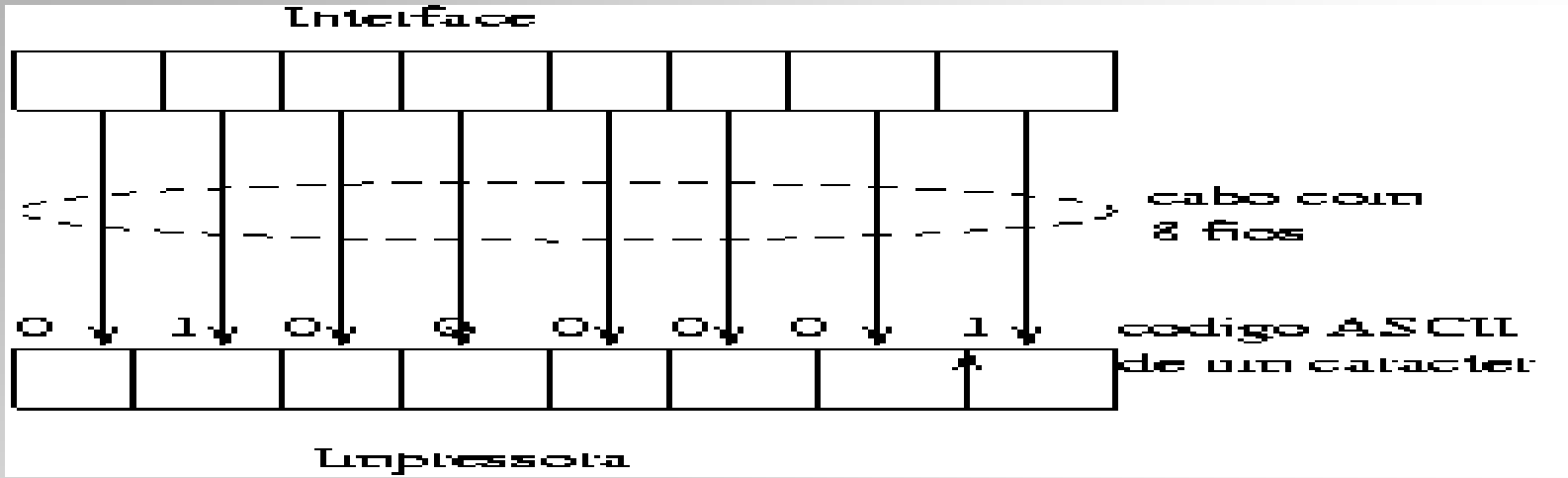


Cada caracter:
- 1 byte ASCII
- 1 byte ATRIBUTO

exigiria 4000 bytes de Video RAM

Nos ecran bit-map: cada bit em Video RAM controla directamente 1 pixel do ec

Interfaces com transmissão de bits em paralelo



Interfaces de I/O

Controlo / endereçamento / transferência de dados uniforme entre o CPU e os periféricos, através do Bus de sistema

- Controlo da operação do periférico, conforme os comandos do CPU
- Informa sobre o estado do periférico
- Conversão de formatos de representação dos dados

Uma interface genérica

Esquema genérico da interface

En geral, as interfaces têm três tipos de registadores (ou portas)

Porta de dados: para guardar os dados enviados pelo CPU ou recebidos do exterior pelo periférico

Porta de estado: indica informação sobre o estado corrente do periférico

- existência de dados para ler
- disponível para receber dados
- erros
- outras informações

Porta de controlo: para receber comandos vindos do CPU (inicializar interface, posicionar a interface em estados para receber ou enviar, e outros comandos que desencadeiam a oeração do periférico)

Especificação dos endereços das portas das interfaces dos periféricos

Há dois métodos principais:

i) Instruções especiais de I/O (ou I/O isolado)

formato das instruções: código I argumentos I endereço da porta de interface

Exemplo: organização lógica das portas como um vector

IN.PORT[np] -- de entrada

np – nº de porta

permite endereçar qualquer porta individual, dando o nº np

OUT.PORT[np] -- de saída

Neste caso têm-se instruções do tipo:

IN registadorCPU, np

registadorCPU := IN.PORT[np]

OUT np, registadorCPU

OUT.PORT[np] := registadorCPU

ou ainda instruções de salto condicionado ao estado de flags do periférico

Exemplo dos processadores 80x86

O CPU gera sinais de controlo para o Bus, indicando se o acesso é a memória ou a interfaces de periféricos

As instruções de I/O especificam endereços de 16 bits

→ até 64 Kilo portas de interfaces

acc = AX (2 bytes) ou AL(1 bytes)

port: um valor imediato de 0 a 255

Entrada:

IN acc, port

acc := IN.PORT[np]

Saída:

OUT port, acc

OUT.PORT[np] := acc

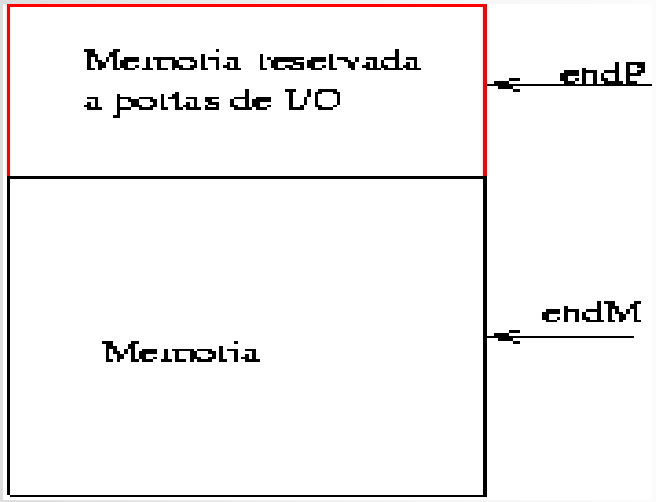
Outras formas:

IN acc, DX DX indica um nº de porta, 16 bits: de 0 a 65535 portas

OUT DX, acc

ii) Endereços das interfaces projectados na memória (Memory-mapped I/O)

não há instruções especiais de I/O: cada porta de interface tem associado um endereço de memória



No espaço real de endereços de memória há uma zona reservada, por hardware, para as portas de interface.

Qualquer instrução de referência de memória, ao aceder à memória reservada, desencadeia a respectiva transferência para a porta endereçada-

Exemplo de instruções:

load registadorCPU, endp

registadorCPU := Mem[endp]

store endp, registadorCPU

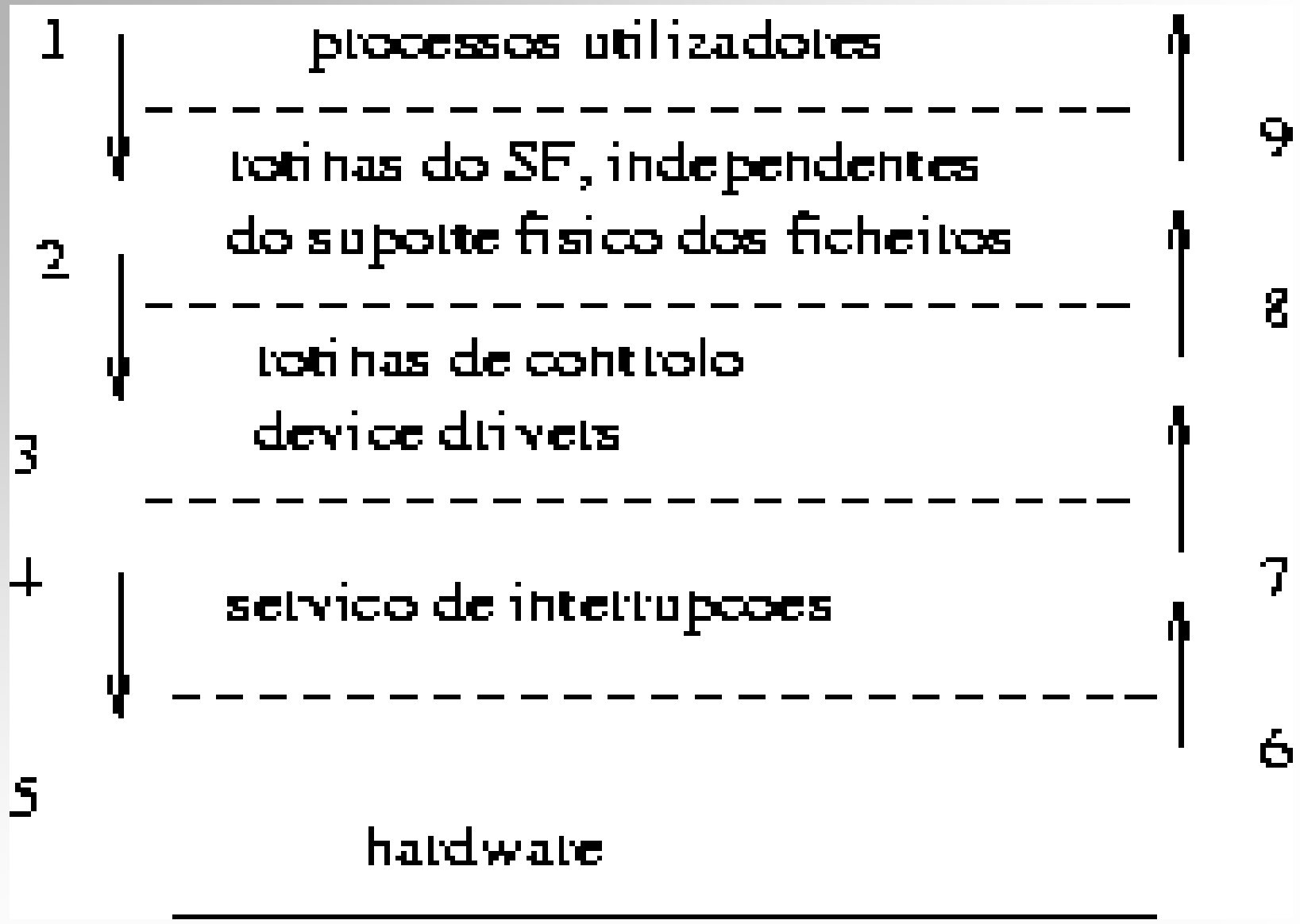
Mem[endp] := registadorCPU

***Device Driver*: conjunto de subrotinas para efectuar operações de entrada / saída, escondendo, aos programas utilizadores, os detalhes de controlo da interface de um periférico.**

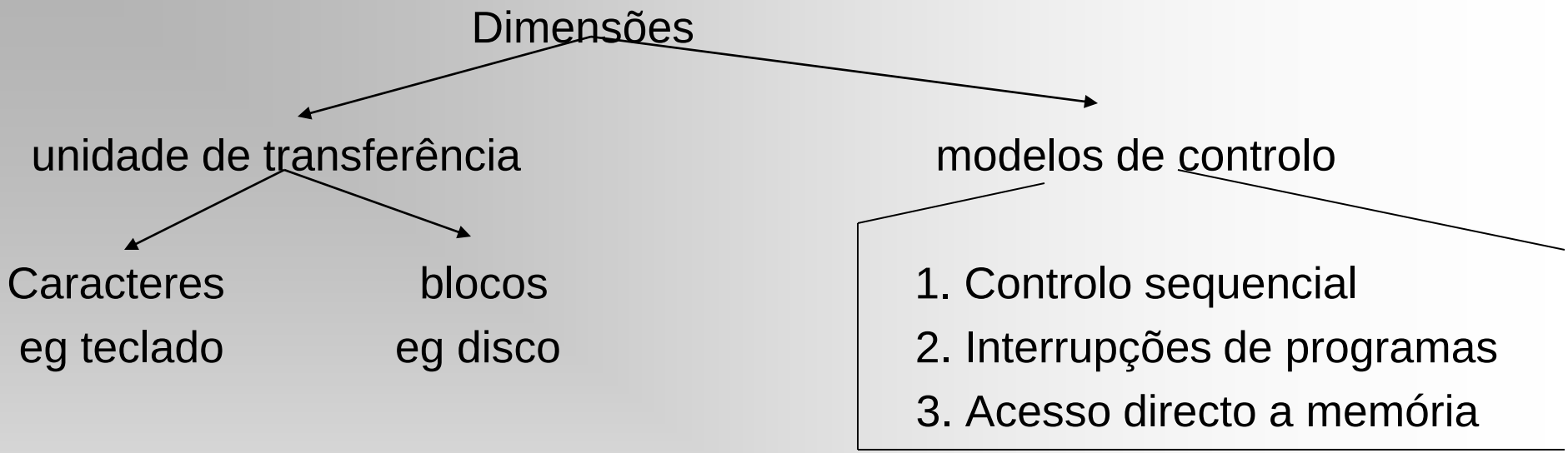
Para dispositivos idênticos, que só diferem nos respectivos endereços das portas de interface, as subrotinas do device driver podem ser partilhadas, desde que se possa dar, como parâmetro, o ´´endereço base´´ das portas de interface de cada periférico.

Fácil: se há ´´memory-mapped I/O´´

No caso de ´´I/O isolado´´, podem-se usar as instruções que especificam os números das portas através de um registador do CPU (DX, no caso 80x86)


1
processos utilizadores
9
rotinas do SF, independentes
do suporte fisico dos ficheiros
2
8
rotinas de controlo
device drivers
3
4
7
servico de interrupcoes
6
5
hardware

Modelos de programação das entradas e saídas



Controlo das operações de entrada de dados

Bit Ready no RegEstado da interface:

Ready = 1 -- indica que a interface recebeu um caracter que pode ser lido pelo CPU

Ready = 0 – indica que não há caracteres para ler

O CPU envia comandos e testa bits de estado da interface:

-- no início envia START para inicializar a interface

-- o bit READY fica a 0 (a interface não tem caracteres para ler)

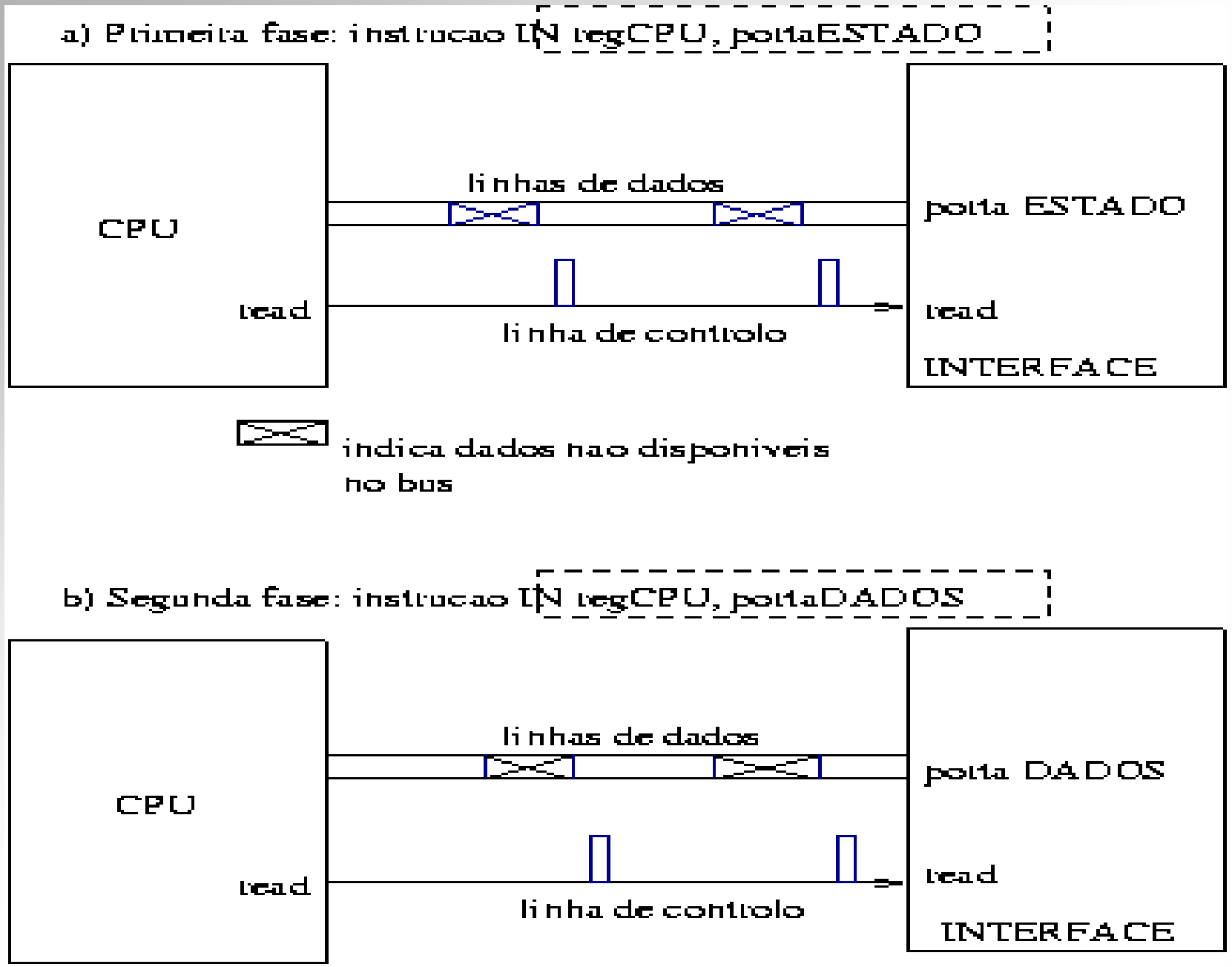
-- quando um caracter chegar …

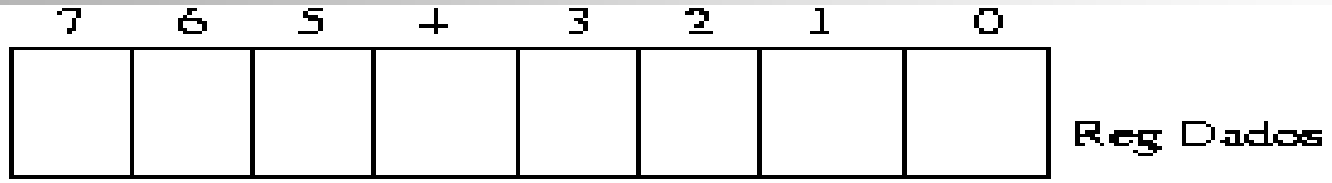
a interface põe o bit READY a 1

eg – ao premir uma tecla, o código do carácter fica no RegDados

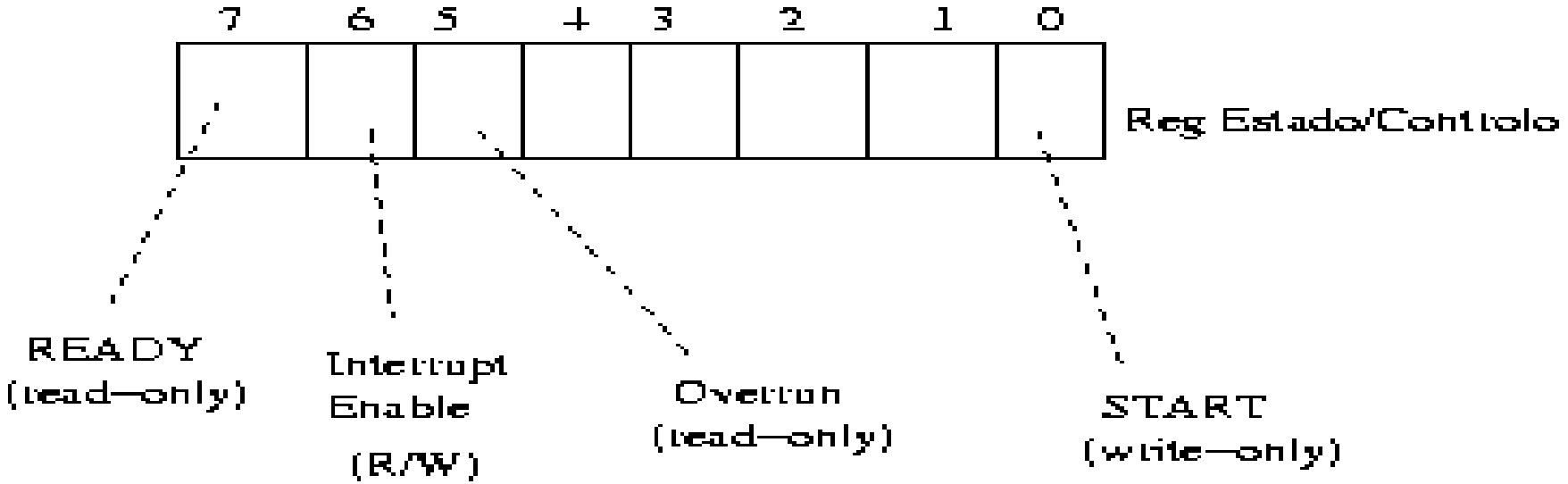
e o bit READY é posto a 1 (no RegEstado)

-- a subrotina de leitura só deve ir ler RegDados quando READY tiver vindo a 1


a) Primeira fase: instrucao IN regCPU, portaESTADO
CPU
read
linhas de dados
linha de controlo
porta ESTADO
read
INTERFACE
indica dados nao disponiveis no bus
b) Segunda fase: instrucao IN regCPU, portaDADOS
CPU
read
linhas de dados
linha de controlo
porta DADOS
read
INTERFACE



codigo do caracter recebido



Subrotina para Ler do Reg Dados:

START := 1 —— depende do periferico
WHILE READY = 0 DO ;
registadorCPY := RegDados;
Retorno ao programa;

O modelo de controlo por Espera Activa (*Busy Waiting*) pode originar desperdício na utilização do tempo do CPU, enquanto este espera que o periférico coloque os dados na interface.

Exemplo:

```
ciclo:  in al, RegEstado    -- lê o RegEstado para o CPU (reg AL)
        shl al, 1           -- testar bit 7 (Ready) através de CF
        jnc ciclo           -- salta se CF = 0
        in al, RegDados     -- lê o carácter do RegDados
```

Controlo das operações de saída de dados

O CPU envia comandos e testa bits de estado da interface:

-- no início envia START para inicializar a interface

-- o bit READY fica a 1 (a interface está disponível para receber caracteres)

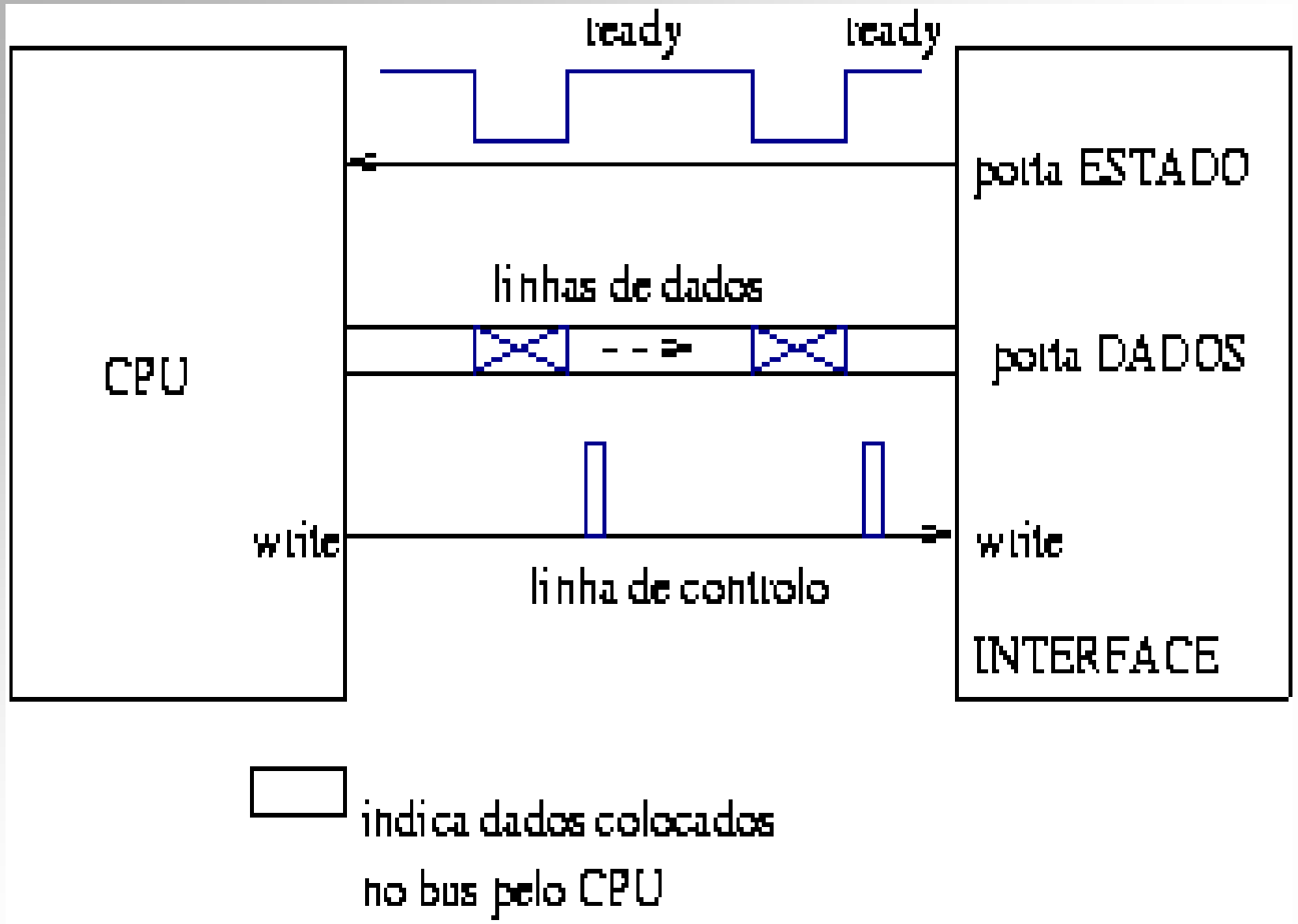
-- a subrotina de escrita só deve enviar um carácter para o RegDados quando READY estiver a 1

-- **quando o CPU enviar um carácter para o RegDados …**

a interface põe o bit READY a 0

começa a operação de fazer sair o carácter…

-- **quando o carácter acabar de sair, a interface volta a pôr READY a 1**


ready
ready
porta ESTADO
linhas de dados
CPU
porta DADOS
write
write
linha de controlo
INTERFACE
indica dados colocados
no bus pelo CPU

Fazer sair um caracter:

```
WHILE READY = 0 DO ;
RegDados := RegistadorCPU;
```

Exemplo:

```
ciclo:  in al, RegEstado      -- lê o RegEstado para o CPU (reg AL)
        shl al, 1                     -- testar bit 7 (Ready)
        jnc ciclo
        mov al, Caracter_a_Enviar
        out  RegDados, al     -- envia o carácter para o RegDados
```

Espera activa:

- má utilização do tempo do CPU

-- boa, para controlo dedicado de interfaces simples, se houver N CPUs

- má, para o controlo ´´simultâneo´´ de múltiplos periféricos, em situações de *controlo em tempo real*

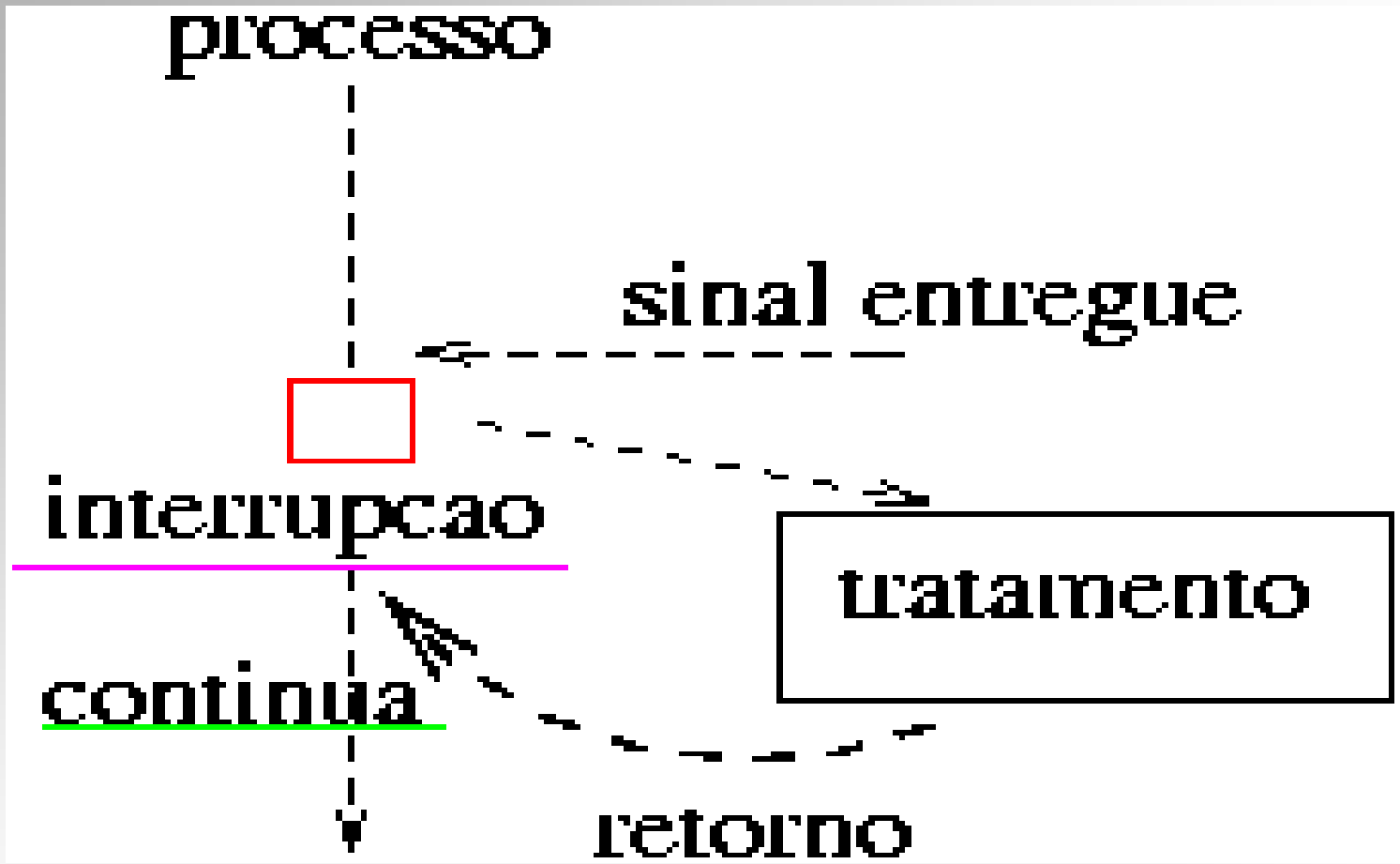
- má, para suportar multiprogramação

Mecanismo de interrupções de programa:

- o periférico fica responsável por assinalar eventos ao CPU, através de uma linha do Bus (*Interrupt Request*)
- ao receber o sinal de pedido de interrupção, o CPU:
  - -- interrompe a sequência de execução de instruções corrente
  - -- força um salto (´´*call*´´) para uma subrotina de tratamento do pedido de interrupção.

Permite-se, assim, que:

- durante as operações dos periféricos, o CPU fique disponível para executar outros programas, sem perda de tempo em ciclos de espera activa, para teste do bit Ready do RegEstado da interface.


processo
sinal entregue
interrupcao
tratamento
continua
retorno

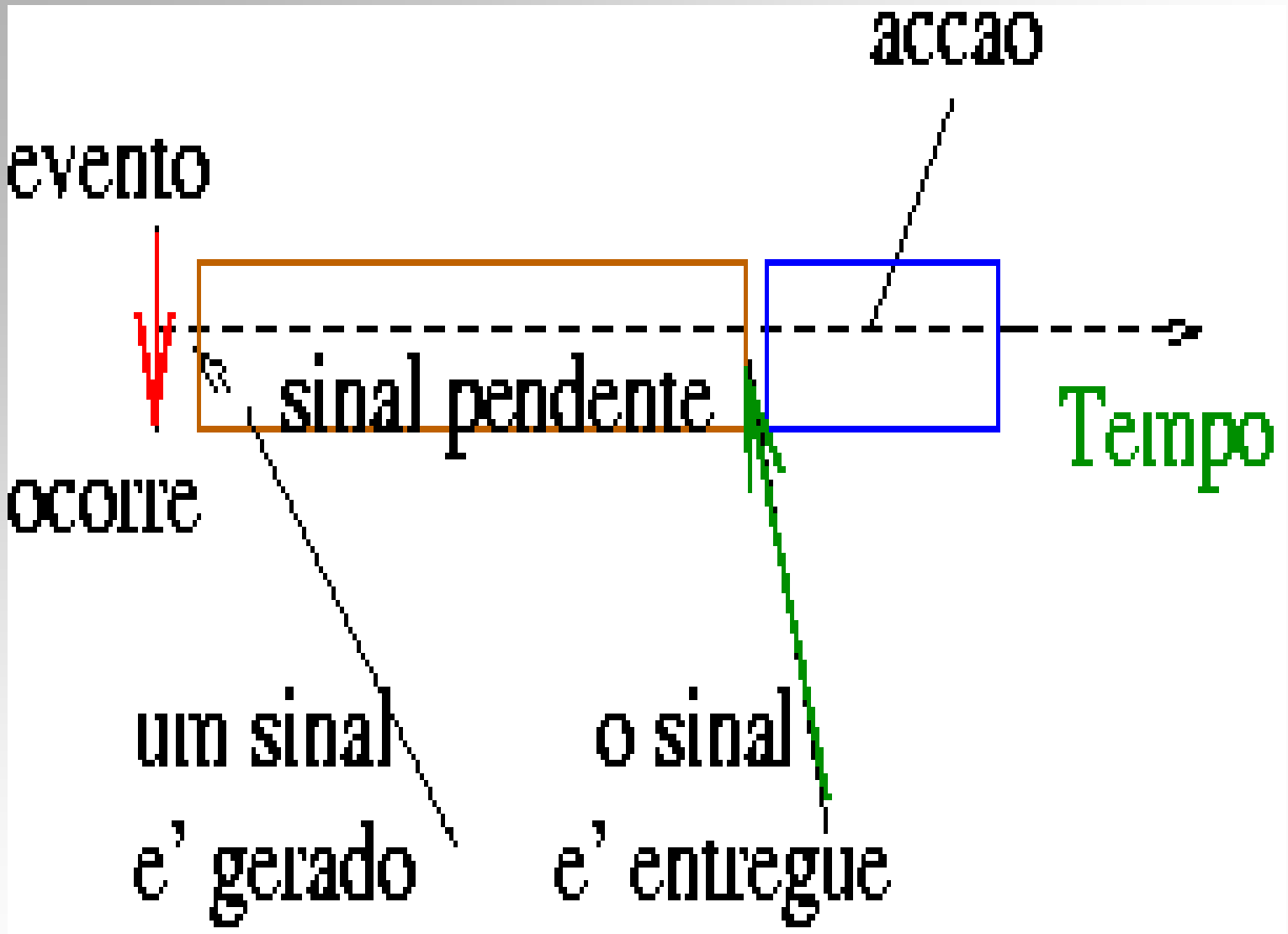
Um periférico envia um pedido de interrupção ao CPU:

-- colocando um sinal na linha **InterruptRequest** do Bus

Quando o CPU aceita o pedido:

- ´´suspende´´ a execução do programa corrente
- salta para uma subrotina RSI (RotinaServiçoInterrupção)
- após a execução de RSI, o CPU retoma a execução do programa interrompido

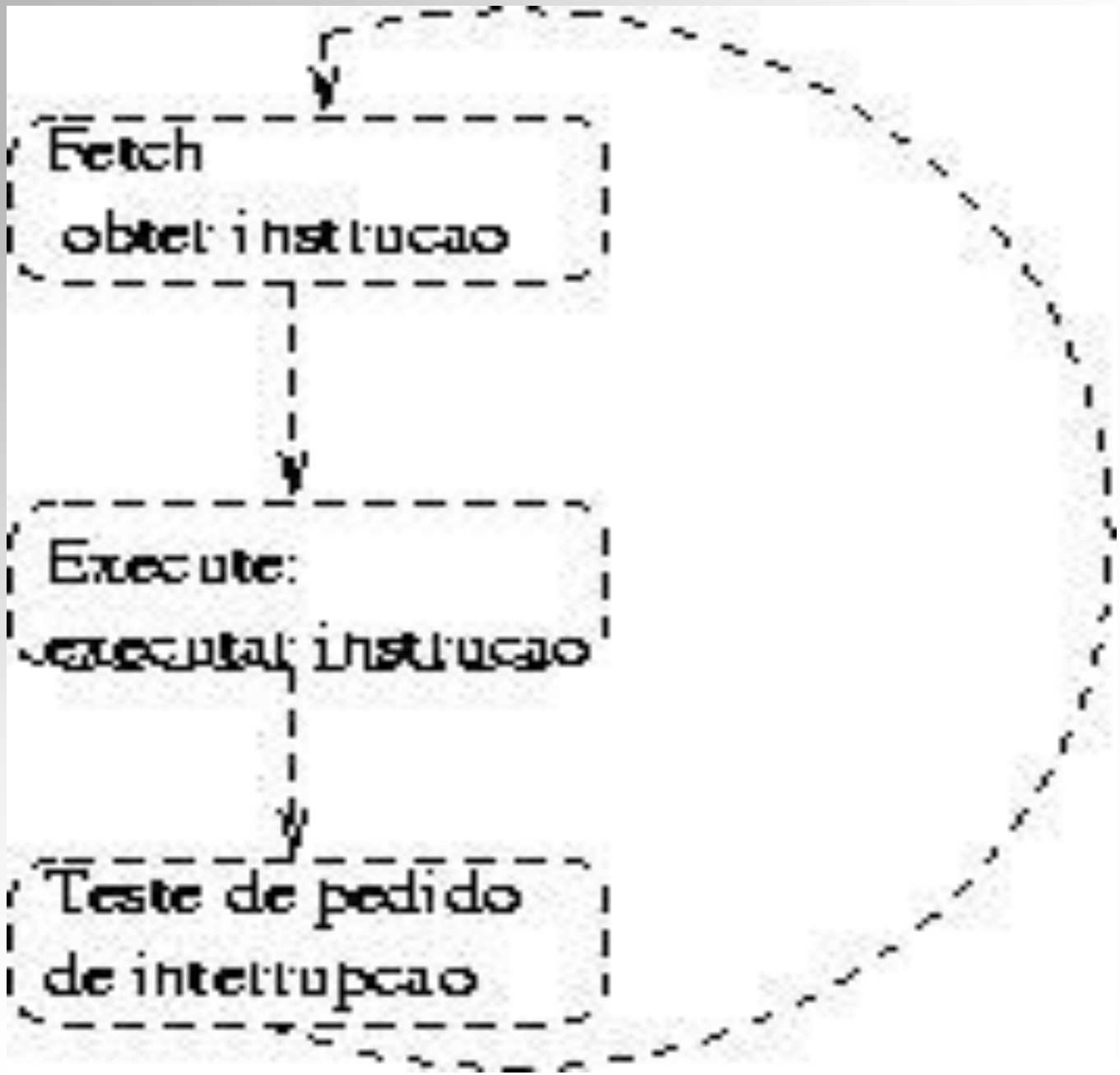
(excepto pelo intervalo de tempo decorrido, o programa interrompido não se apercebe da ocorrência da interrupção)


accao
evento
sinal pendente
Tempo
ocorre
um sinal
e' gerado
o sinal
e' entregue

Questão: quando se poderá interromper um programa em execução, com segurança?

-- tal que se possa, depois de tratar a interrupção, retomar a execução do programa, com o CPU e a Memória no mesmo estado

- Noção de ´´Contexto de execução do programa´´: definido pelos valores dos registadores do CPU e das células de memória usadas pelo programa
  - **É bem definido no fim da execução de cada instrução, mas não durante a execução de cada instrução**


Fetch
obter instrucao
Execute:
executar instrucao
Teste de pedido
de interrupcao

Para controlar a aceitação de interrupções dos periféricos, o CPU tem dois estados especiais:

´´Atento´´ ou ´´Não atento´´ às interrupções

Indicado pela ´´Interrupt Flag´´ IF: 1 bit no registador de estado do CPU (Processor Status Word ou Flags Register)

IF = 1 → CPU atento a interrupções

IF = 0 → CPU não atento

O programa pode modificar IF explicitamente, pelas instruções (no 80x86):

STI -- Set Interrupt Flag: IF = 1

CLI – Clear Interrupt Flag: IF = 0

Tratamento de uma interrupção do programa:

- uma parte é feita automaticamente por mecanismos ´´hardware´´
- outra parte é explicitamente programada, na rotina de serviço RSI

Esquema genérico:

/*acções hardware*/

- uma vez aceite a interrupção:
  - **o CPU passa ao estado ´´não atento´´ (equivale a IF = 0);**
  - **salvaguarda-se a parte essencial do contexto de execução do CPU;**
  - **o fluxo de execução de instruções ´´salta´´ para um endereço de memória, a partir do qual está carregada a rotina de serviço RSI associada à origem do pedido;**

/*acções software*/

uma vez iniciada a execução da Rotina de Serviço da Interrupção:

salvaguarda-se o resto do contexto de execução do CPU;

trata o pedido de interrupção (acedendo às portas da interface do periférico);

restaura o contexto de execução do CPU, passa ao estado ´´atento´´ e retorna ao ponto do programa onde este havia sido interrompido

(idêntico a um retorno de subrotina) → instrução IRET (Interrupt Return)

Que parte do contexto de execução deve ser salvaguardada por ´´hardware´´?

-- o mínimo essencial para permitir, depois, retomar a execução a partir do mesmo ponto:

→ o Program Counter e o Flags Register

-- mas, se a RSI vai usar diversos registadores do CPU, há que guardá-los também.

Compromissos:

-- guardar sempre, por ´´hardware´´, todos os registadores do CPU seria mais geral, facilitaria escrever as RSI, mas seria mais lento e nem sempre necessário.

A solução adoptada depende do suporte ´´hardware´´ oferecido:

→ **O mais vulgar é:**

a) guardar só o PC e o FlagRegister por ´´hardware´´ e o resto pela RSI

Outras possibilidades:

b) ter instruções especiais load/store ou push/pop, operando sobre múltiplos registadores…

ou o próprio ´´hardware´´ fazer isto automaticamente, antes de saltar para a RSI

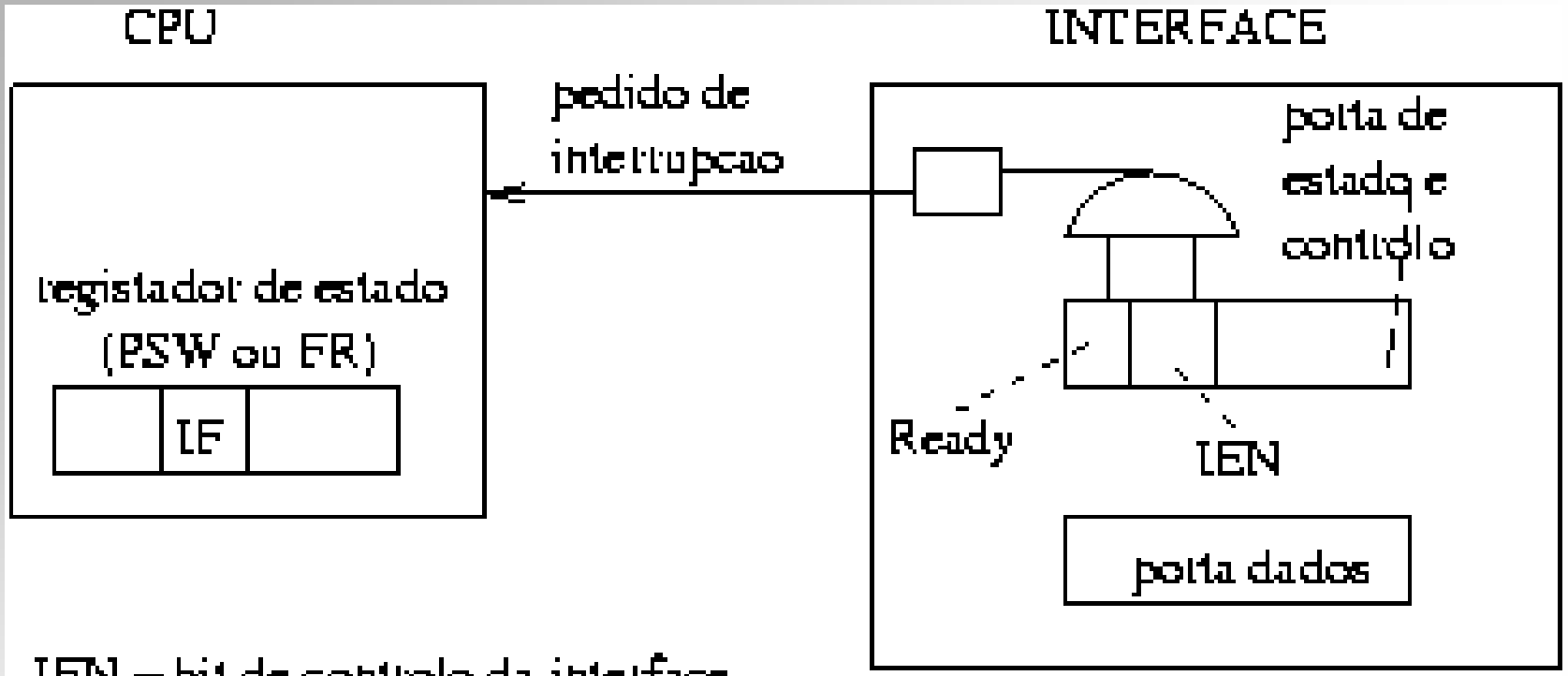
c) ter os registadores do CPU duplicados e limitar-se a comutar

do conjunto de registadores corrente

para o conjunto de registadores para o serviço de interrupções.

São soluções para apressar a comutação de contextos (*context switching*).

Aceitar uma interrupção:



IEN – bit de controlo da interface que permite inibir a geracao de pedidos de interrupcao:

se pusermos IEN = 0, a interface nao faz pedidos, mesmo que tenha o bit Ready = 1

se pusermos IEN = 1, a interface faz um pedido logo que tenha o bit Ready = 1; este pedido fica pendente ate ser atendido pelo CPU

a) Pedido de interrupção: feito pela interface de um periférico

-- o periférico põe o bit Ready a 1

-- se tem o bit InterruptEnable (IEN) = 1, essa interface pode gerar pedidos de interrupções

b) Aceitação do pedido pelo CPU, quando:

-- tem IF = 1 (estado ´´atento a interrupções´´)

-- acabou de executar a instrução corrente

-- há pedido(s) de interrupção pendente(s)

No caso de saída de caracteres:

no RegControlo da interface:

bit IEN – posto a 1 ou a 0, pelo programa, para a interface poder gerar interrupções ou não

no RegEstado da interface:

bit Ready – posto a 1 pela interface, quando esta está disponível para receber mais caracteres

quando o CPU envia um novo caracter:

→ o bit Ready fica a 0, durante a sua saída.

Se IEN = 1 e Ready = 1:

→ a interface gera um novo pedido de interrupção ao CPU,

logo que acaba de fazer sair o caracter corrente.

Exemplo: imprimir uma cadeia de caracteres

O programa faz:

preparar argumentos

call Imprimir -- chama subrotina que assume que

-- Endereço inicial da cadeia é ´´End´´ em memória

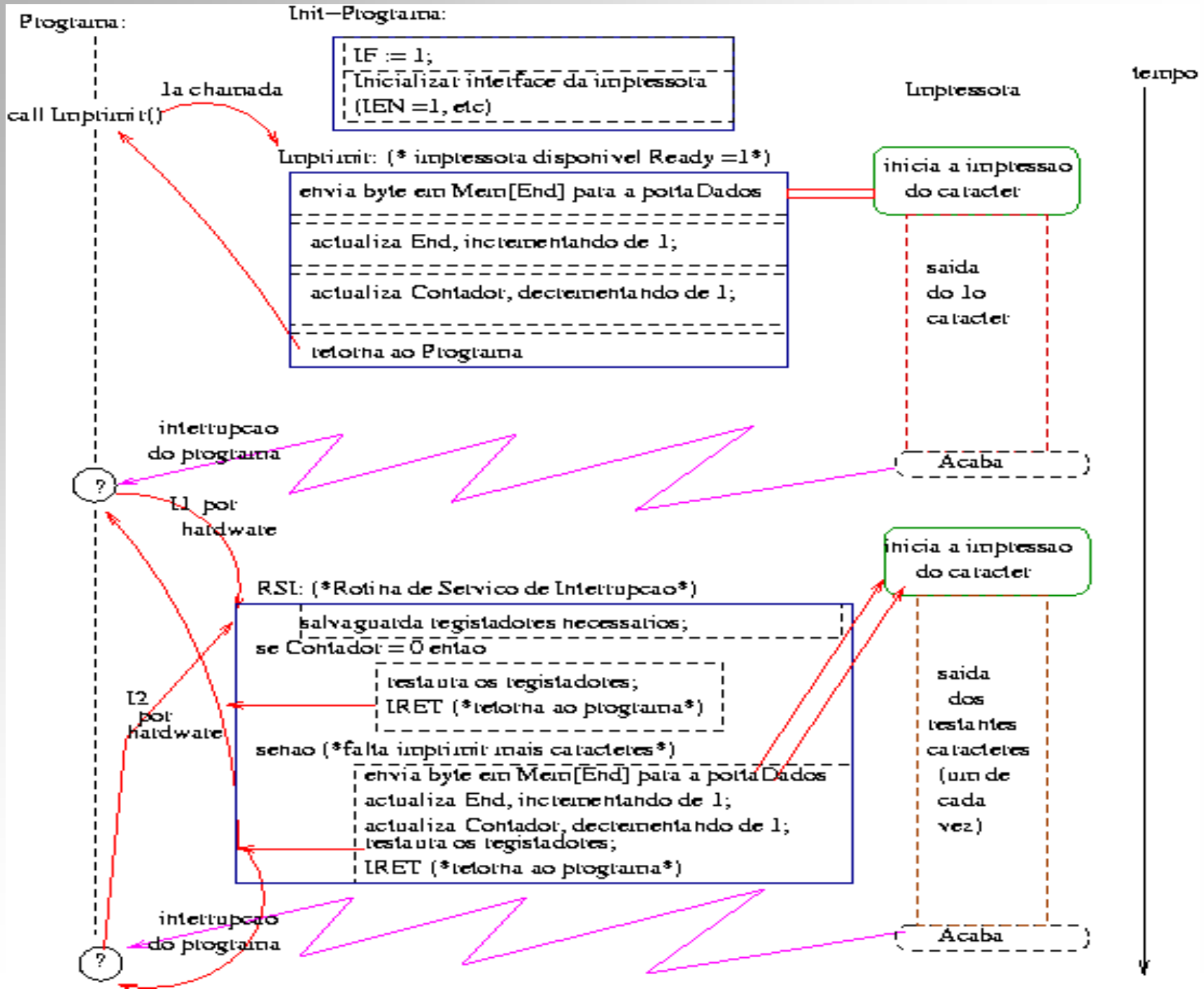
-- Número de caracteres é indicado por ´´Contador´´

Pretende-se que o programa possa continuar a sua execução, durante a saída da cadeia de caracteres.

Isto é, o retorno de ´´Imprimir´´ ao programa principal deve ser imediato…

Não se pode usar o modelo de Espera Activa.

Porquê?


Programa:
Init–Programa:
IF := 1;
Inicializar interface da impressora (IEN =1, etc)
1a chamada
call Imprimir()
Imprimir: (* impressora disponivel Ready =1*)
envia byte em Mem[End] para a porta Dados
actualiza End, incrementando de 1;
actualiza Contador, decrementando de 1;
retorna ao Programa
Impressora
inicia a impressao do caracter
saida do 1o caracter
tempo
interrupcao do programa
Acaba
I1 por hardware
RSI: (*Rotina de Servico de Interrupcao*)
salvaguarda registadores necessarios;
se Contador = 0 entao
restaura os registadores;
IRET (*retorna ao programa*)
senao (*falta imprimir mais caracteres*)
envia byte em Mem[End] para a porta Dados
actualiza End, incrementando de 1;
actualiza Contador, decrementando de 1;
restaura os registadores;
IRET (*retorna ao programa*)
I2 por hardware
inicia a impressao do caracter
saida dos restantes caracteres (um de cada vez)
interrupcao do programa
Acaba

E se o programa voltar a chamar Imprimir(…) com um novo pedido, antes da cadeia de caracteres anterior ter acabado de ser impressa?

Uma solução parcial…

→ Indicador booleano ´´**Ocupada**´´, gerido pelo programa:

no início **Ocupada = FALSE;**

… antes de cada pedido….o programa faz:

WHILE **Ocupada** aguardar….

… ao iniciar a impressão de um novo pedido…

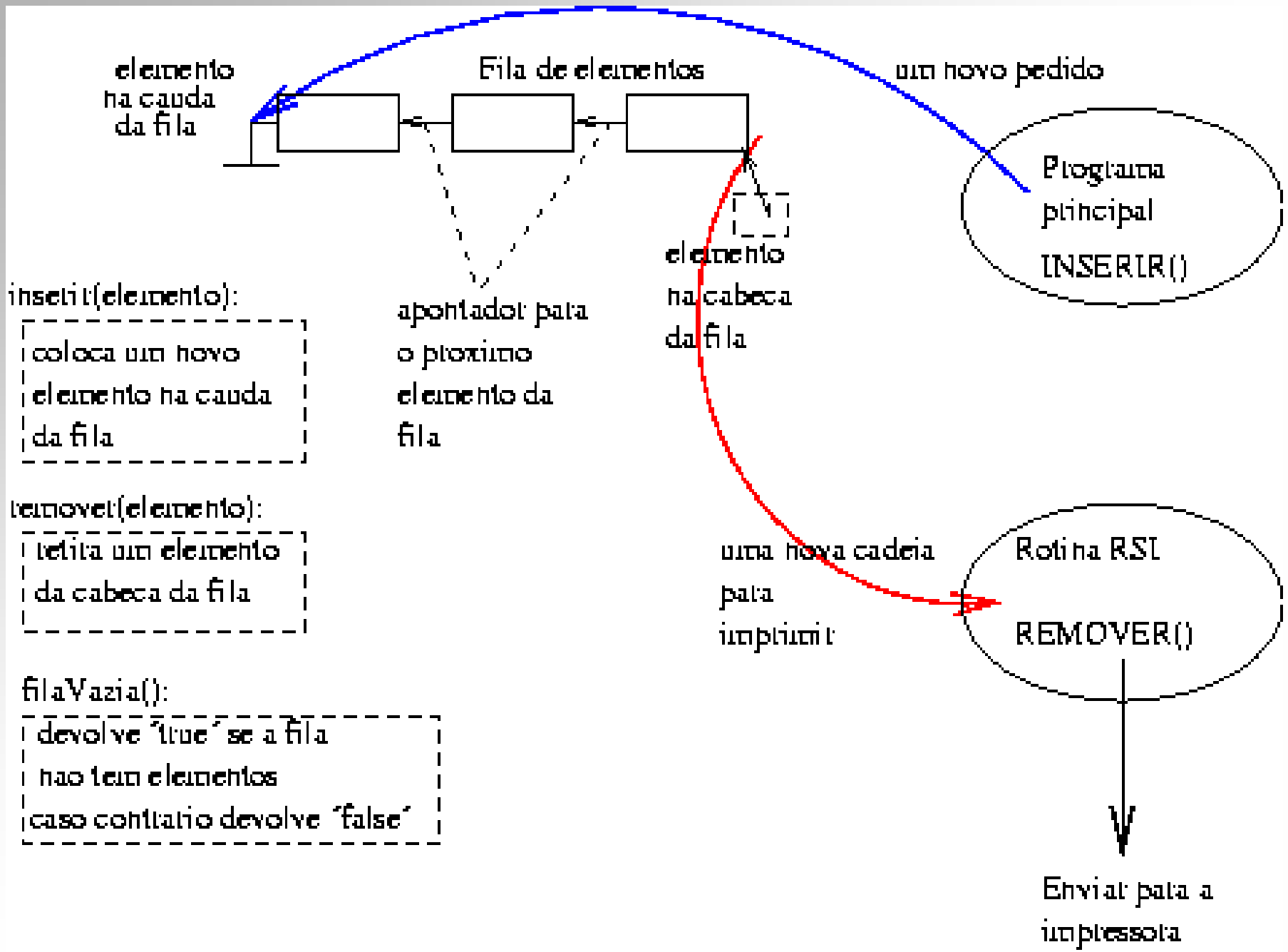
Ocupada = TRUE;

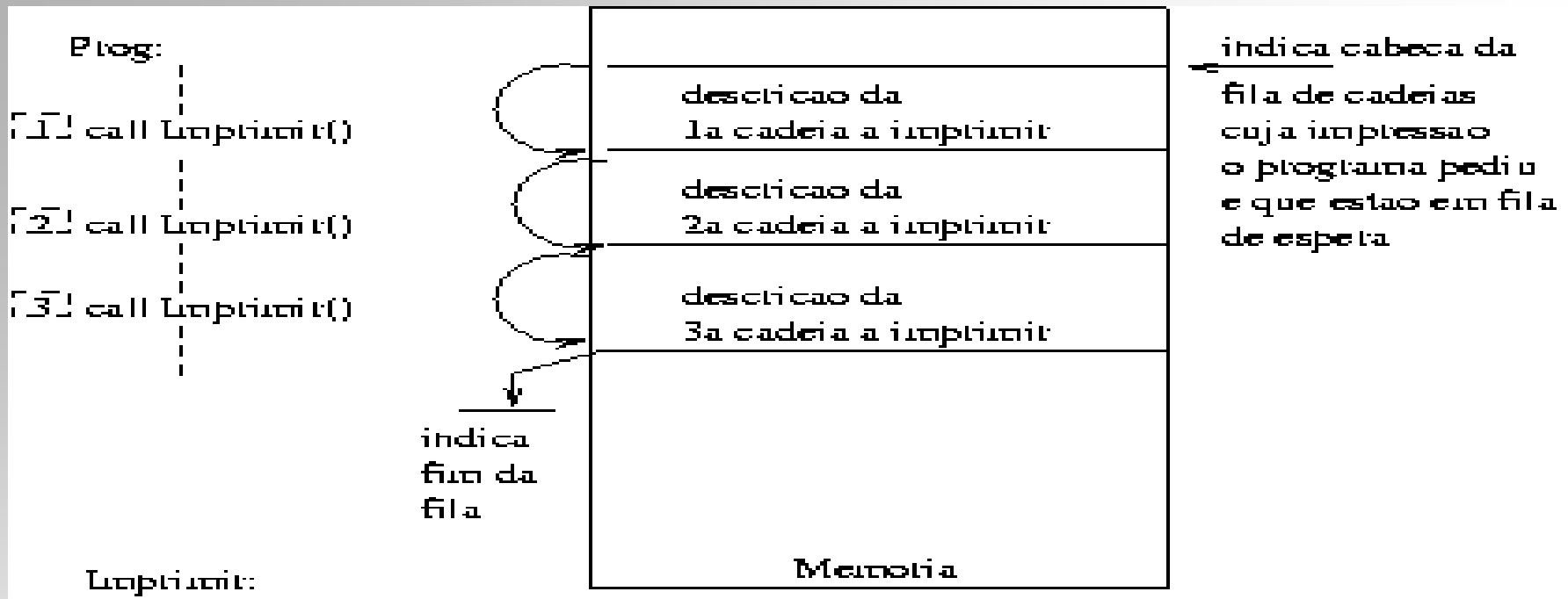
… ao completar a impressão de cada pedido….

Ocupada = FALSE;

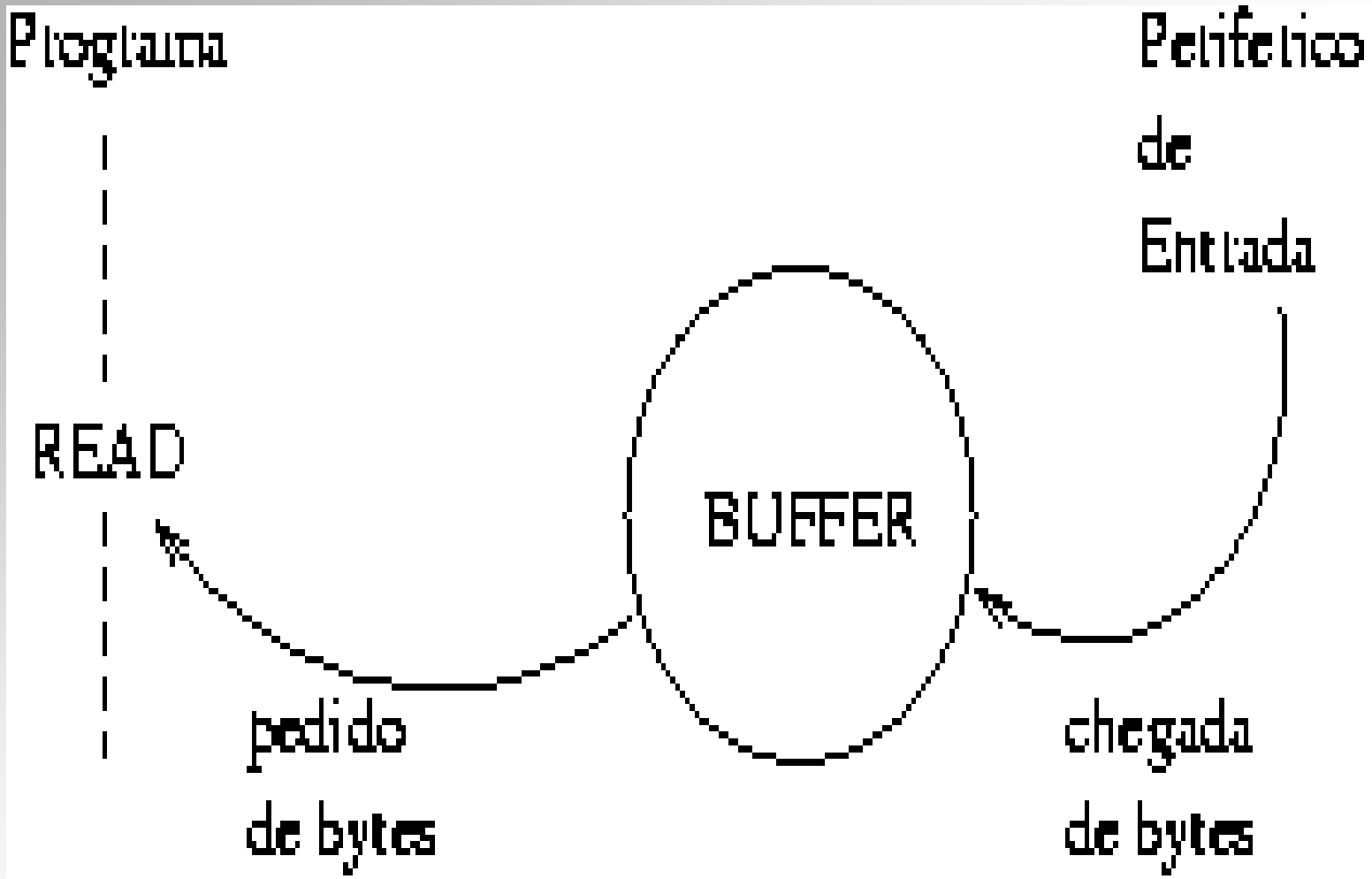
```
Init—Programa:
    IF := 1;   Ocupada := FALSE
    Inicializar interface da impressora
    (LEN =1, etc)

Imprimir:
A   enquanto Ocupada fazer ;
    Ocupada := TRUE;
    End := EnderecoInicial;
    Contador := Tamanho;
    envia byte em Mem[End] para a portaDados
    actualiza End, incrementando de 1;
    actualiza Contador, decrementando de 1;
    retorna ao Programa

RSI: (*Rotina de Servico de Interrupcao*)
    salvaguarda registadores necessarios;
se Contador = 0 entao
        Ocupada := FALSE
        restaura os registadores;
        IRET (*retorna ao programa*)
senao (*falta imprimir mais caracteres*)
        envia byte em Mem[End] para a portaDados
        actualiza End, incrementando de 1;
        actualiza Contador, decrementando de 1;
        restaura os registadores;
        IRET (*retorna ao programa*)
```

Uma solução melhor…


elemento
na cauda
da fila
Fila de elementos
um novo pedido
Programa
principal
INSERIR()
elemento
na cabeça
da fila
inserir(elemento):
coloca um novo
elemento na cauda
da fila
apontador para
o próximo
elemento da
fila
remover(elemento):
retira um elemento
da cabeça da fila
uma nova cadeia
para
imprimir
Rotina RSI
REMOVER()
filaVazia():
devolve "true" se a fila
não tem elementos
caso contrário devolve "false"
Enviar para a
impressora




```
Lnprimir:
  IF := 0, (*desligar interrupcoes*)
  inserir(Endereco,Tamanho);
  se Ocupada entao
     IF := 1, (*ligar interrupcoes*)
     retornar
  senao
     Ocupada:=true;
     End:= Endereco
     Contador:= Tamanho;
     envia lo byte para a interface;
     incrementa End
     decrementa Contador
     IF := 1, (*ligar interrupcoes*)
     retornar
```


Programa
Periférico
de
Entrada
READ
BUFFER
pedido
de bytes
chegada
de bytes

Ler caracteres de um periférico

Quando o programa invoca READ:

esta subrotina tenta ler um caracter de uma zona temporária de memória – Buffer

se o Buffer contiver dados → um deles é lido e devolvido ao programa

se o Buffer estiver vazio → deve obter-se um byte da interface do periférico de entrada

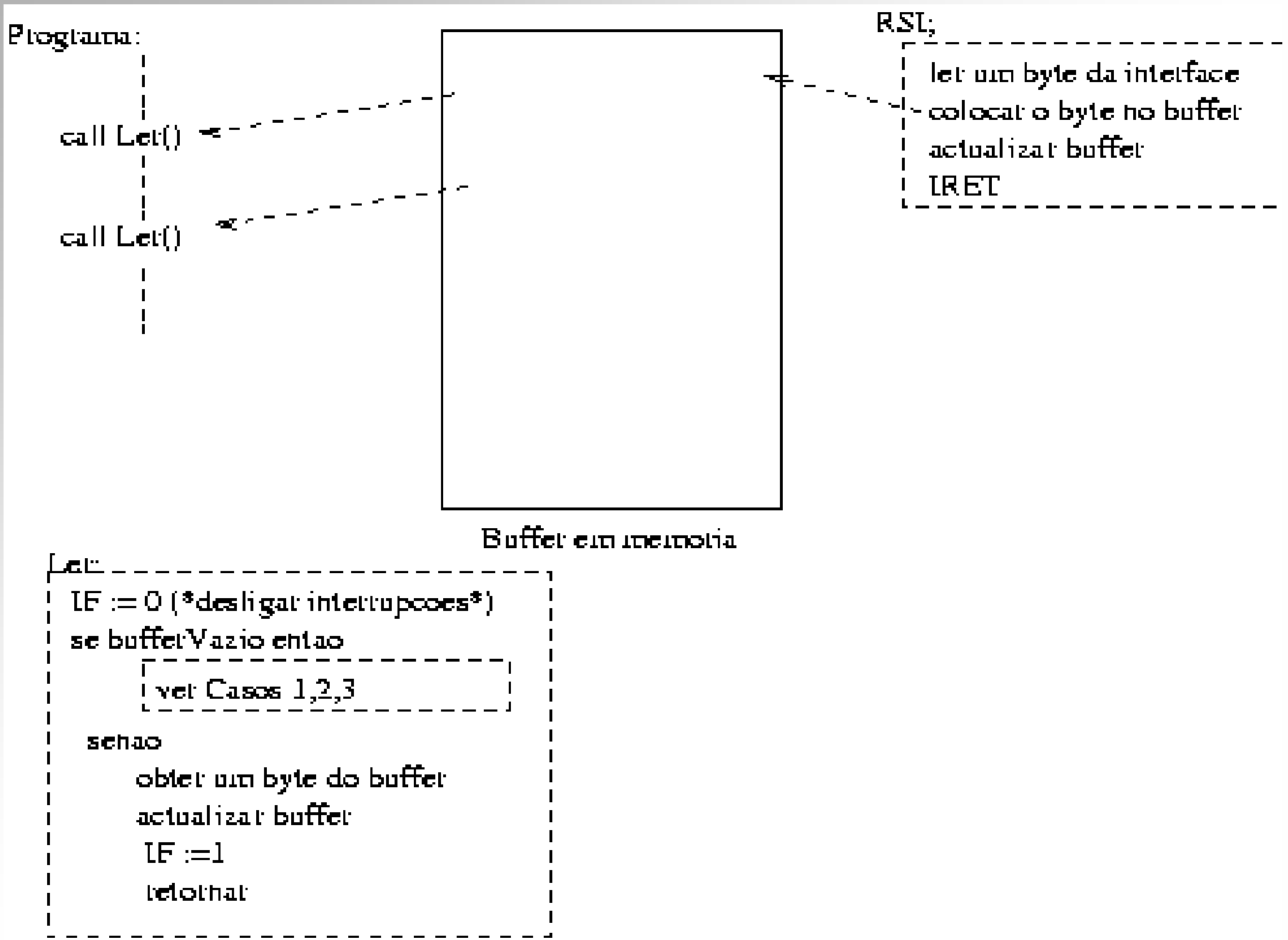
-- neste caso:

(1) a subrotina READ aguarda em espera activa…

ou (2) a subrotina READ retorna de imediato ao programa com a indicação de que não há dados disponíveis (e o programa deverá voltar a tentar mais tarde)

-- chama-se uma Leitura Assíncrona

ou (3) a subrotina READ não retorna ao programa e a sua execução fica SUSPENSA até que haja dados na interface do periférico…


Programa:
call Let()
call Let()
RSI:
let um byte da interface
colocar o byte no buffer
actualizar buffer
IRET
Buffer em memoria
Let:
IF := 0 (*desligar interrupcoes*)
se buffetVazio entao
vet Casos 1,2,3
senao
obter um byte do buffer
actualizar buffer
IF :=1
retornar

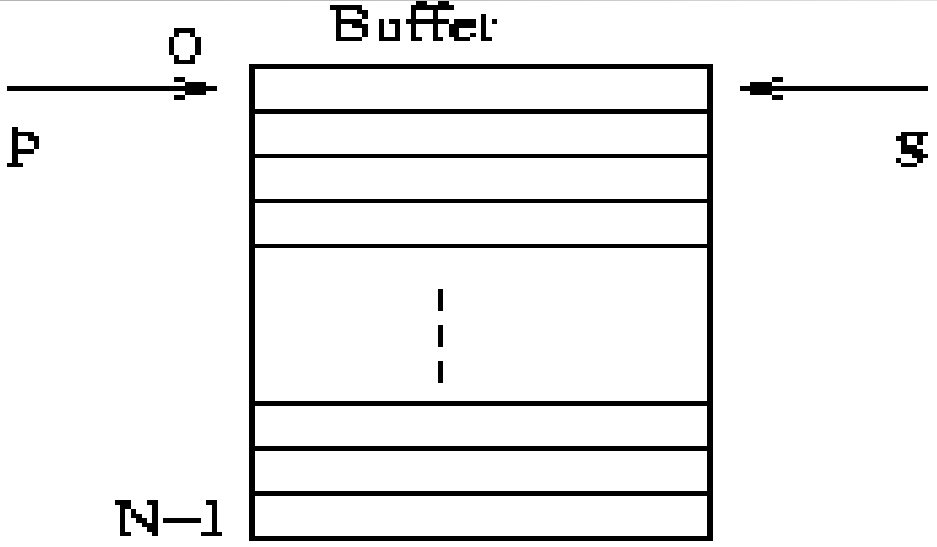
READ:

```
while (cont = 0) do;
```

ciclo de espera activa

```
x := get();
```

leitura ao programa

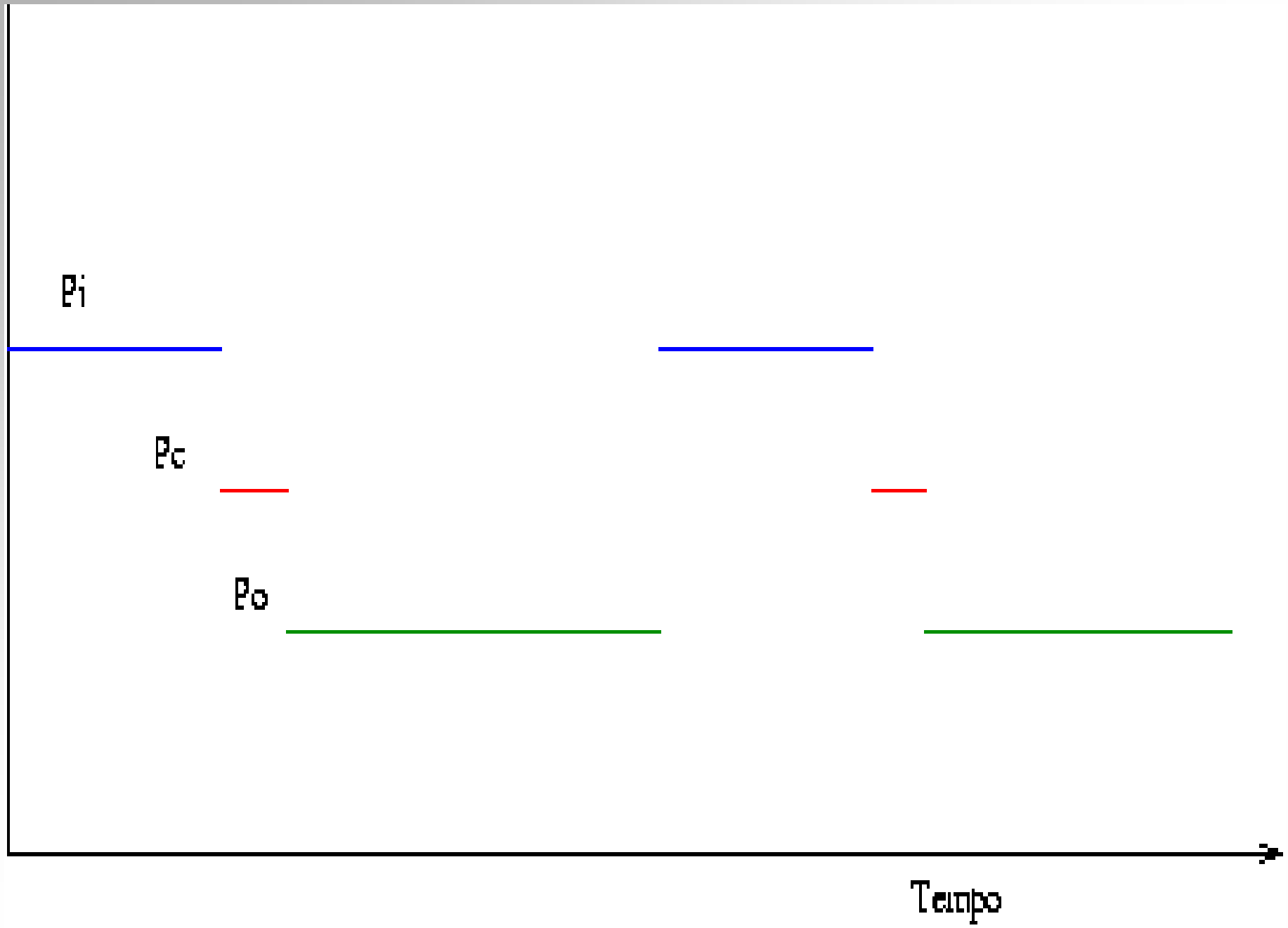



```
put: M[p] := x;
     p := (p+1) mod N;
     cont := cont + 1;

get: x := M[g];
     g := (g+1) mod N;
     cont := cont – 1;
```

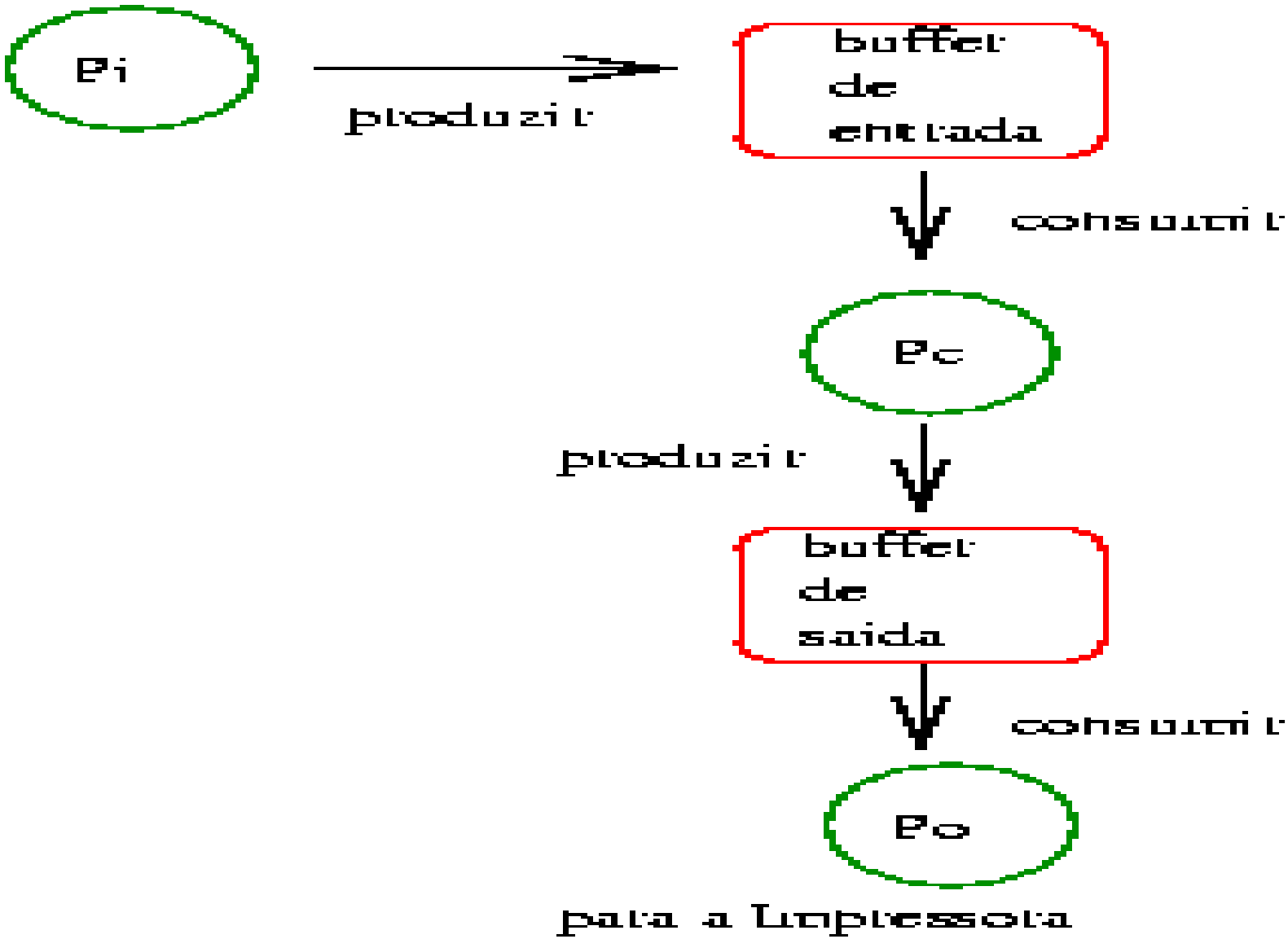
```
p, g: (0 .. N–1)

inicio:
     p = g = 0
     cont = 0

antes de put:
  if cont = N then
        —— buffer cheio
  else
        put(x);

antes de get:
  if cont = 0 then
        —— buffer vazio
  else
        x := get();
```

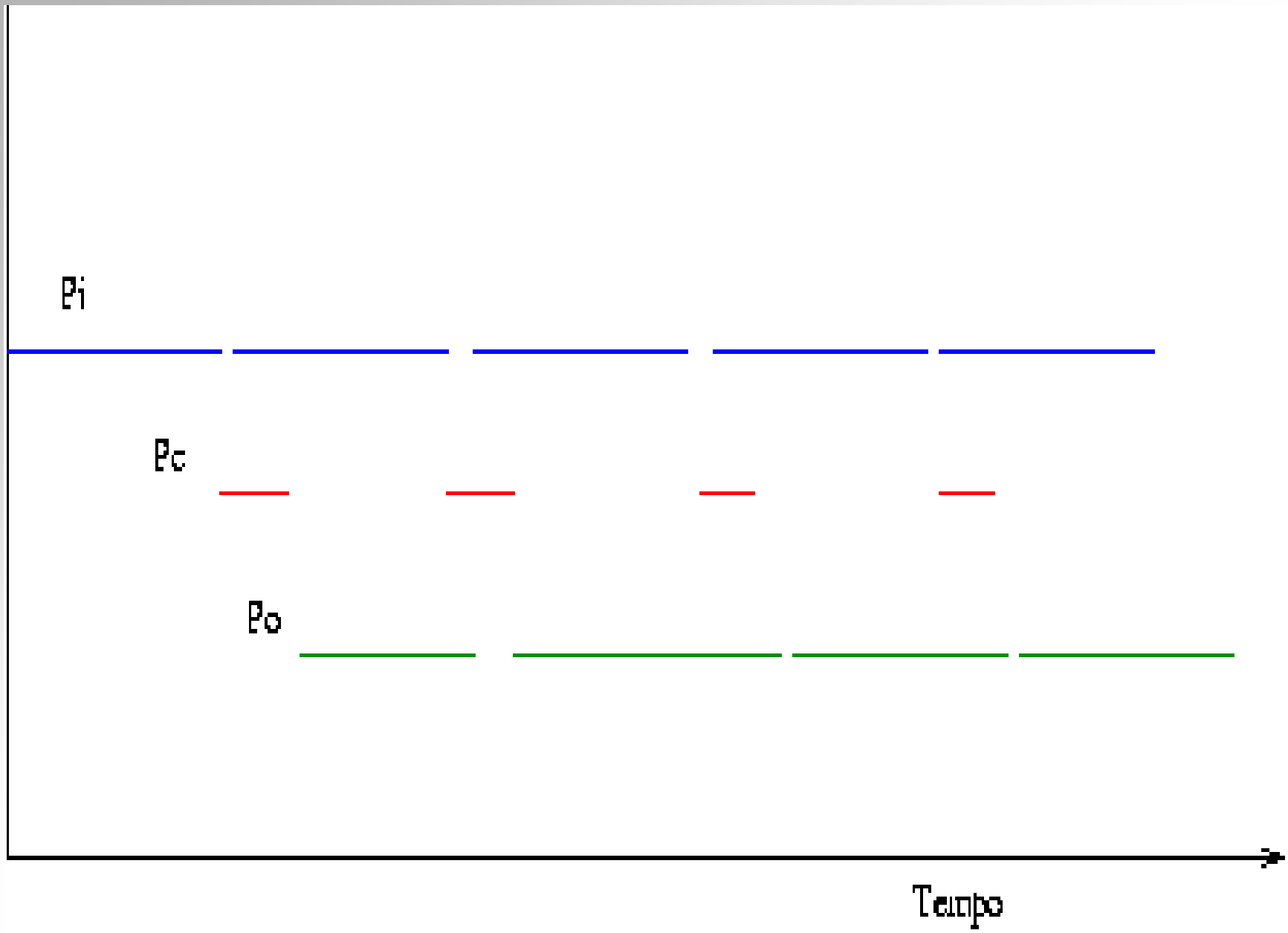
ROTINA SERVICO INTERRUPCOES:

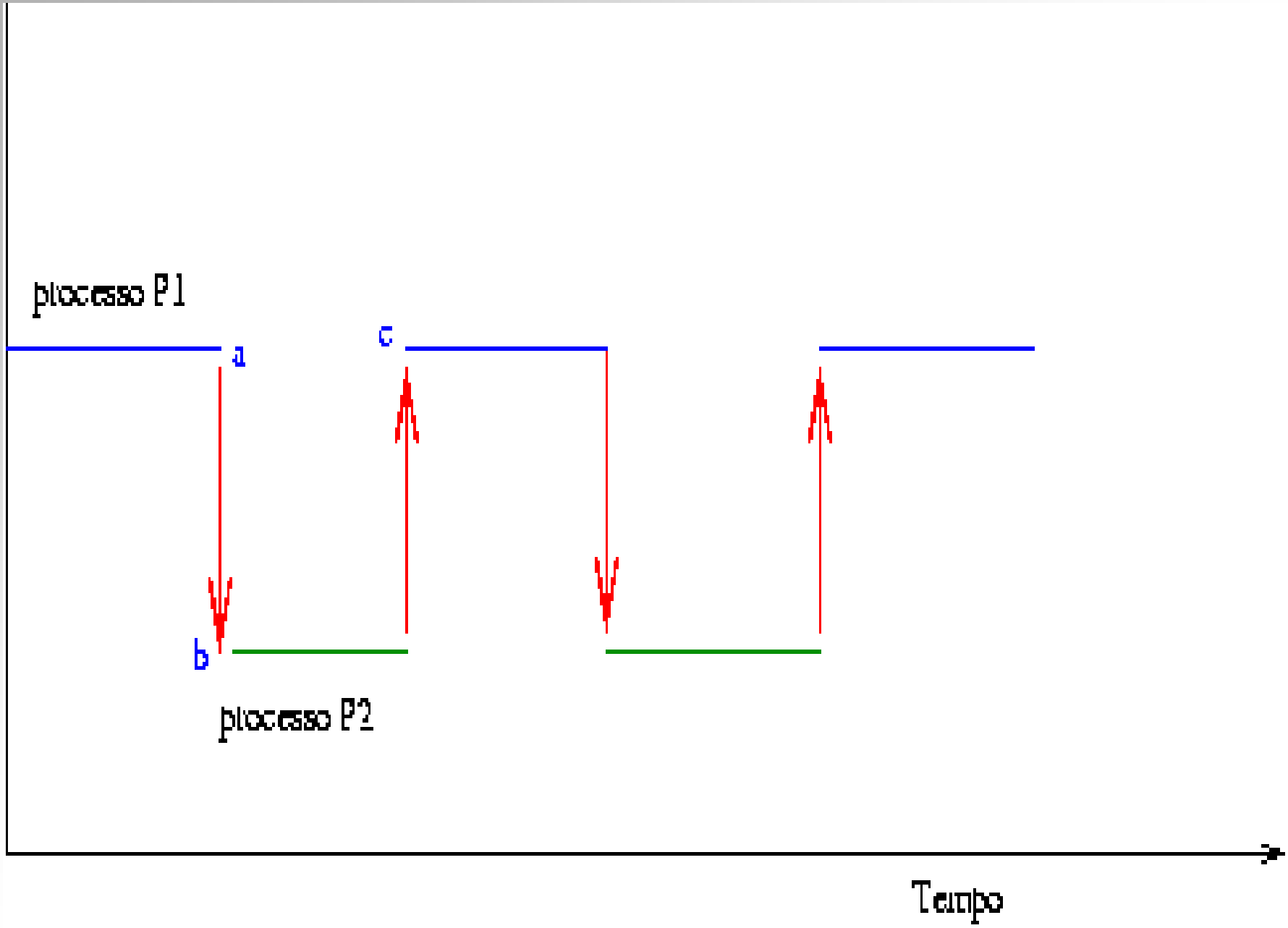
```
RegCPU := PortaDadosTecla;
if (cont = N)
        then
                -- buffer cheio
        else
                put(RegCPU);

IRET;
```

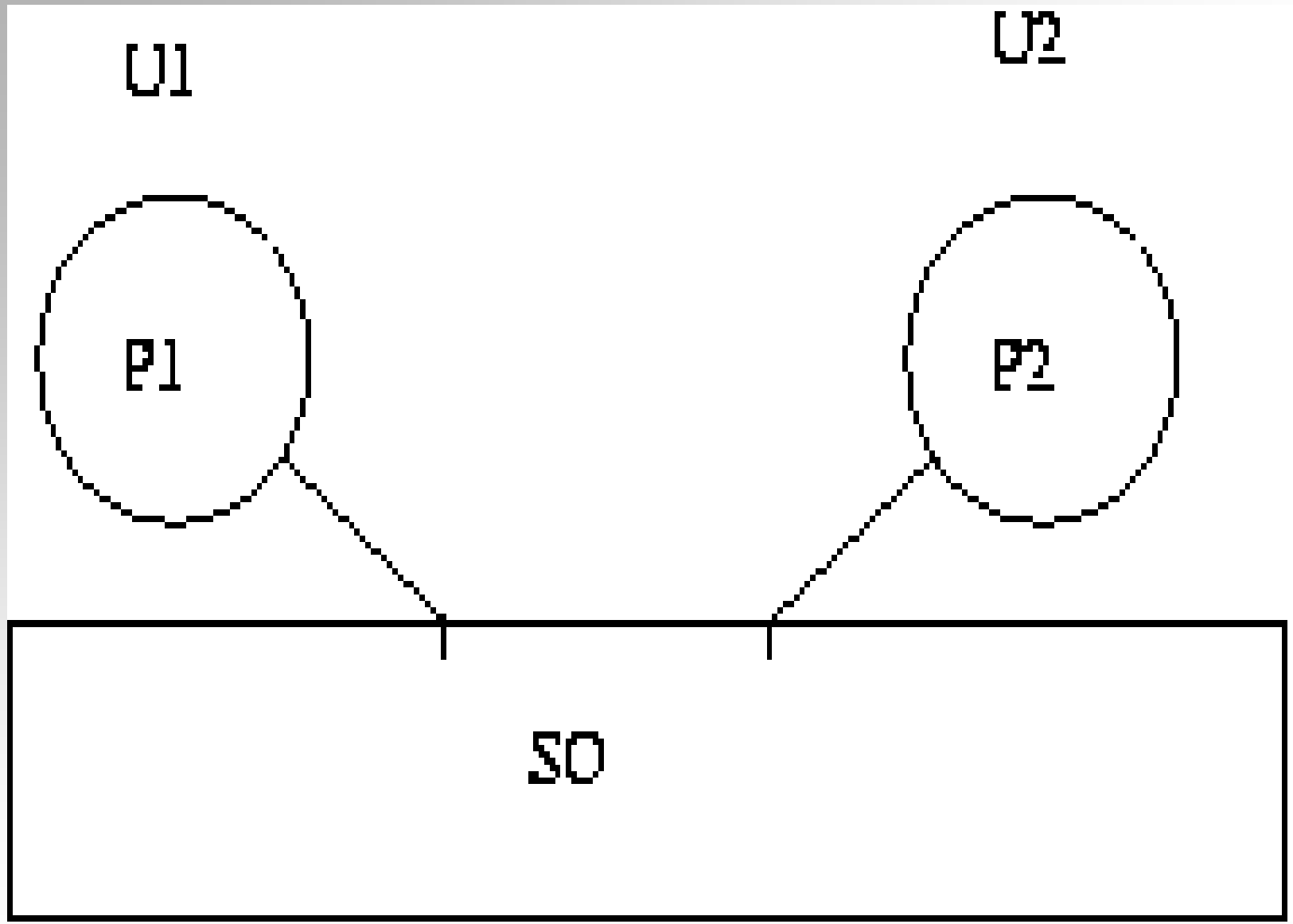

Pi
Pc
Po
Tempo

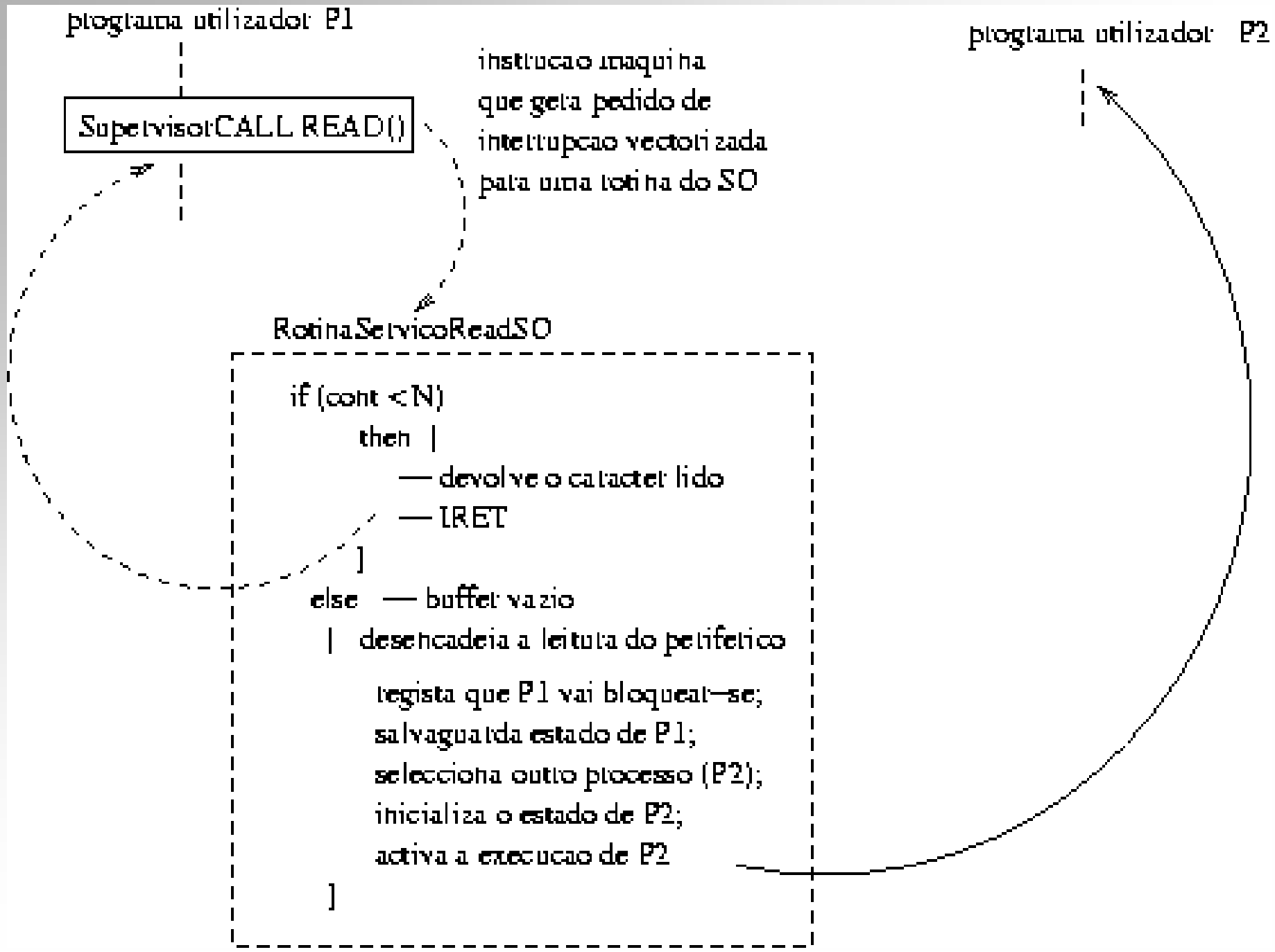
do Leitor de Cartoes

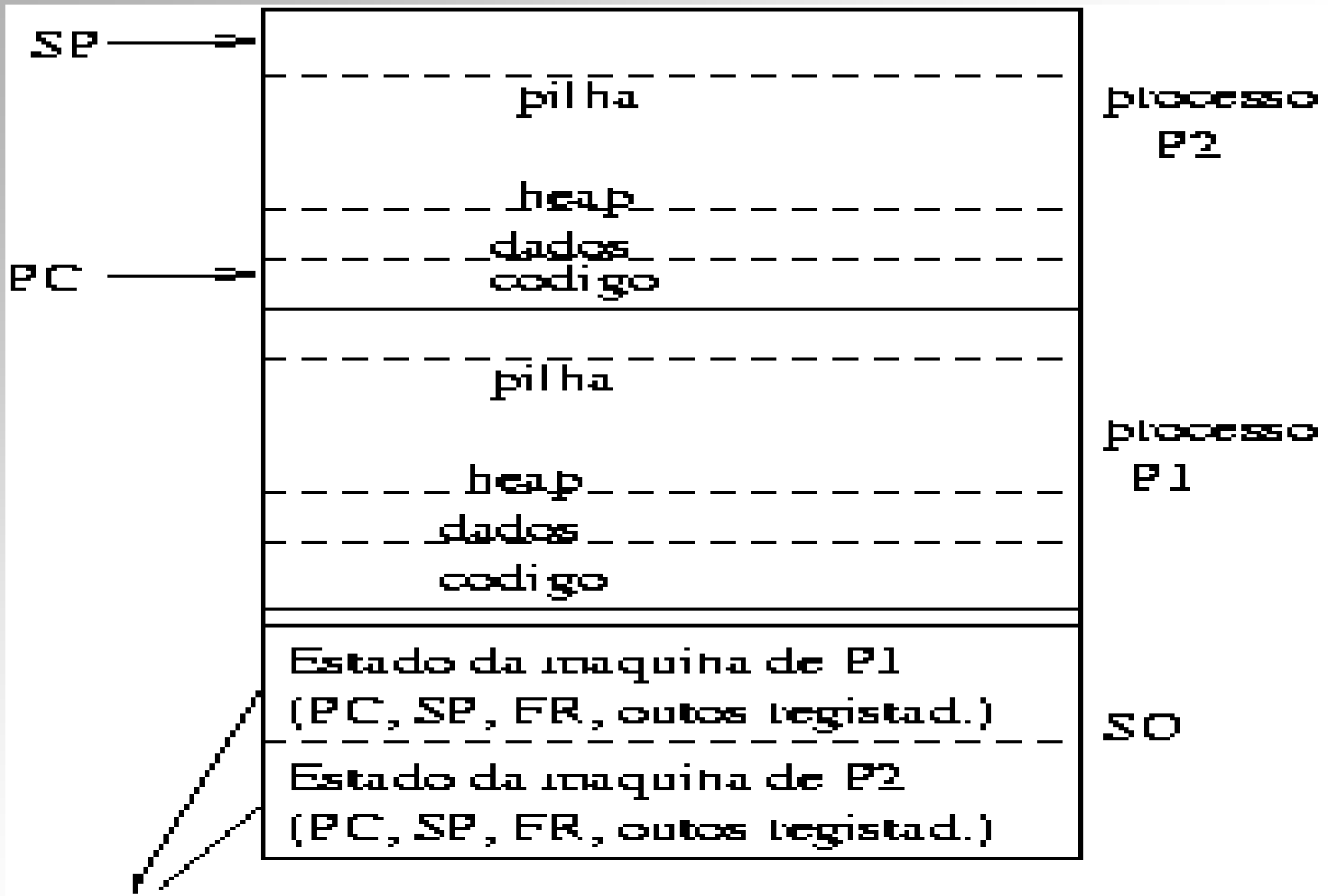


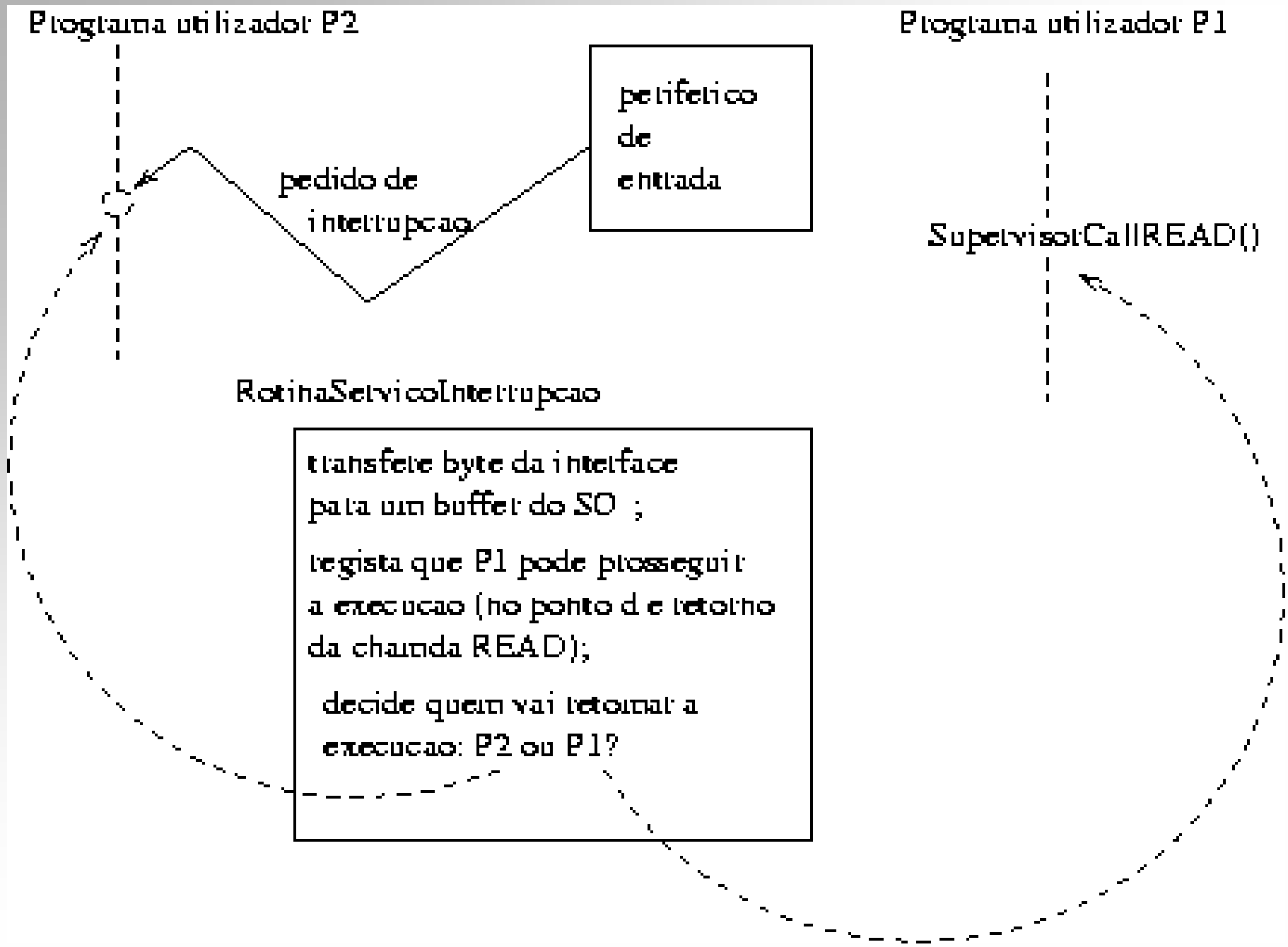

Pi
Pc
Po
Tempo


processo P1
a
c
b
processo P2
Tempo


U1
U2
P1
P2
SO


programa utilizador P1
programa utilizador P2
SupervisorCALL READ()
instrucao maquina
que gera pedido de
interrupcao vectorizada
para uma rotina do SO
RotinaServicoReadSO
if (cont <N)
then |
— devolve o caracter lido
— IRET
]
else — buffer vazio
| desencadeia a leitura do periferico
regista que P1 vai bloquear-se;
salvaguarda estado de P1;
selecciona outro processo (P2);
inicializa o estado de P2;
activa a execucao de P2
]


SP
pilha
heap
dados
PC
codigo
processo P2
pilha
heap
dados
codigo
processo P1
Estado da maquina de P1
(PC, SP, FR, outos registad.)
Estado da maquina de P2
(PC, SP, FR, outos registad.)
SO
entradas na Tabela de processos
que descreve os processos presentes
num dado momento


Programa utilizador P2
periferico
de
entrada
pedido de
interrupcao
Programa utilizador P1
SupervisorCallREAD()
RotinaServicoInterrupcao
transfere byte da interface
para um buffer do SO ;
regista que P1 pode prosseguir
a execucao (no ponto de retorno
da chamda READ);
decide quem vai retomar a
execucao: P2 ou P1?

Identificação da interface do periférico que gerou o pedido de interrupção

→ É necessária para localizar a respectiva Rotina de Serviço da Interrupção

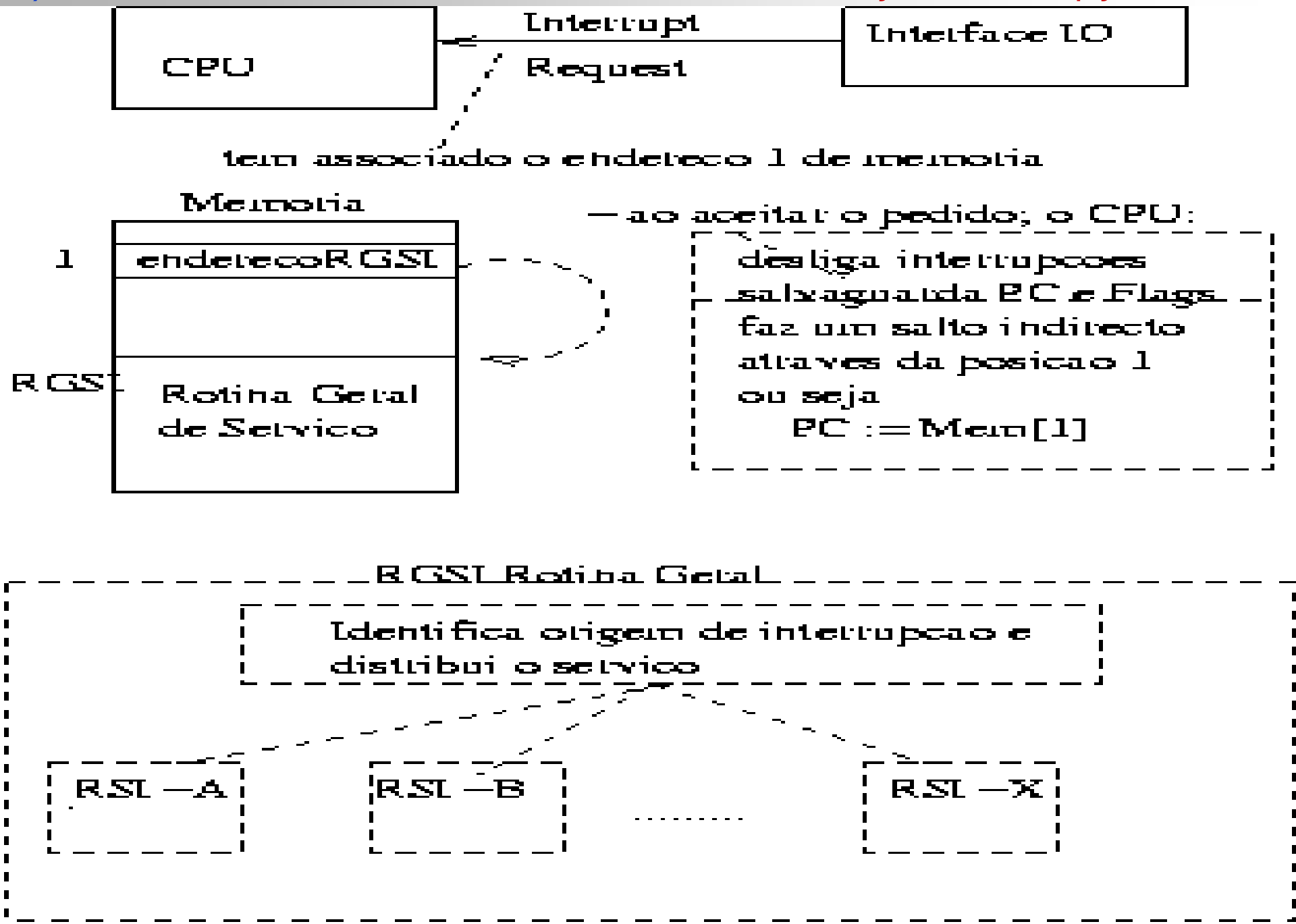
Os principais métodos envolvem uma parte hardware e outra parte software:

a) Baseados numa Rotina Geral de Serviço de Interrupções (RGSI)

-- um endereço pré-definido de memória, onde deverá estar contido o endereço inicial da RGSI é associado, à linha de pedido de interrupções do Bus, por hardware.

b) Baseados em Rotinas de Serviço de Interrupção Vectorizadas

a) Método baseado numa única Rotina Geral de Serviço de Interrupções



Como identificar o periférico que gerou o pedido de interrupção?

i) Por interrogação programada (*polling*) das flags de estado (READY e IEN) de todas as interfaces de I/O ligadas à linha de pedido de interrupções:

IF (READY_1 and IEN_1) THEN …

IF (READY_2 and IEN_2) THEN … **…**

-- a ordem pela qual se testam as flags das interfaces define as prioridades que se lhes atribuem, no atendimento de pedidos de interrupções

-- periféricos ´´mais lentos´´ → menor prioridade no atendimento

Exemplo de duas variantes:

prioridades fixas: usa uma tabela em que cada entrada corresponde a um nível de prioridade associado a uma interface; começa sempre pelo início da tabela, para testar as flags das interfaces

prioridades rotativas (*round robin*): um apontador para a entrada onde começa a testar as flags, é usado de forma circular.

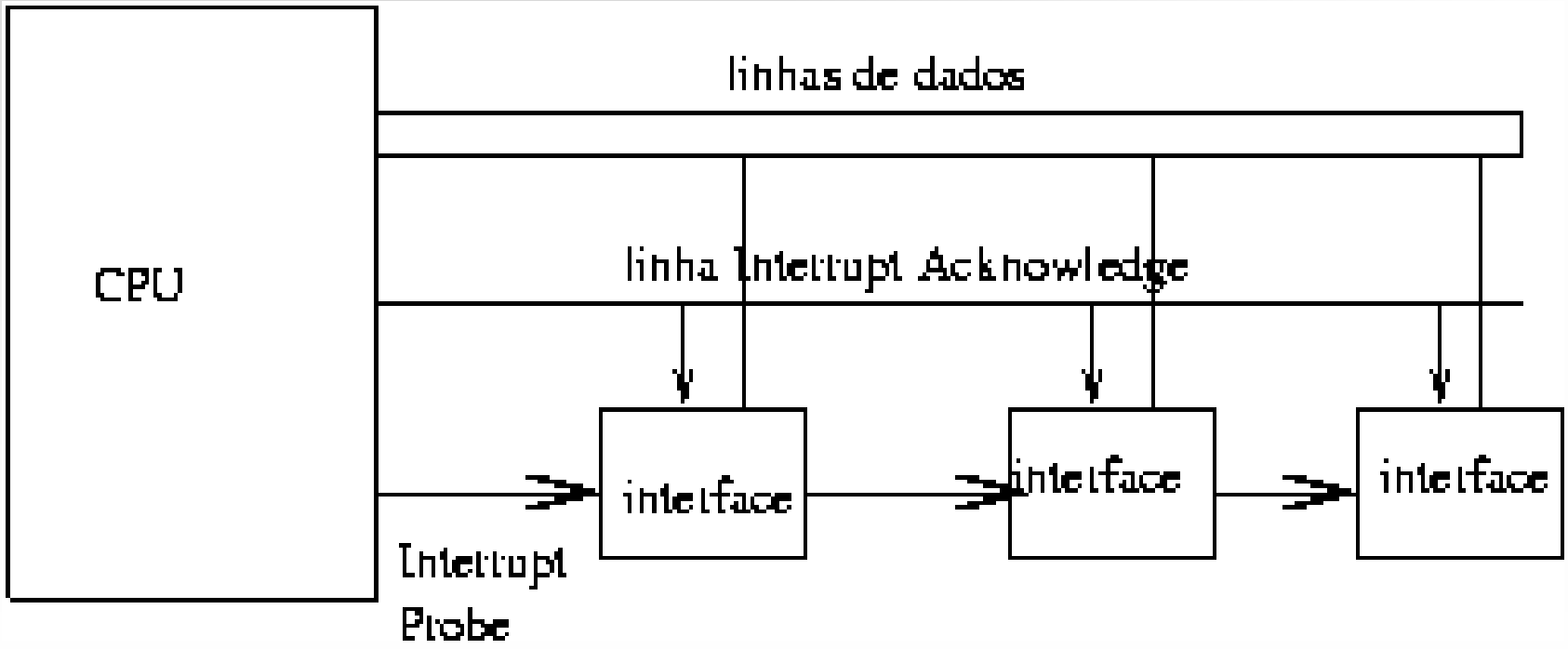
→ Fazer todos os testes por software é pouco eficiente.

ii) Por interrogação (*polling*) das flags de estado, suportada por uma instrução máquina especial: INTERRUPT_ACKNOWLEDGE:

INTA RegistadorCPU

-- coloca, em RegistadorCPU, o número identificador da interface ´´mais prioritária´´ que fez um pedido.

Exemplo de esquema de ligação em *daisy-chain:*



Esquemas para definir as prioridades, sem ser completamente por software :

--- Um componente externo ao CPU decide as prioridades: *árbitro programável.*

-- o árbitro decide qual o pedido mais prioritário e envia esse pedido ao CPU pela linha InterruptRequest

-- o esquema de prioridades é programável.

--- Num esquema de ligação em *daisy-chain* *a* prioridade é função da distância da interface ao CPU e é fixa por hardware.

b) Método baseado em Interrupções Vectorizadas

-- quando o CPU aceita um pedido de interrupção:

→ envia automaticamente, pelo Bus, um sinal que obriga a interface de ´´maior prioridade´´ que fez um pedido, a enviar um código de identificação para o CPU, pelo Bus de dados

→ o CPU usa esse número como o índice de uma tabela de endereços de rotinas de serviço de interrupções (uma entrada por interface)

→ a tabela está em posições pré-definidas de memória.

→ a tabela é inicializada pelo SO

-- o conteúdo de cada entrada contém o endereço para onde o fluxo de execução salta, para cada tipo de interrupção.

Exemplo: PDP-11 (DEC) : ao aceitar um pedido de interrupção:

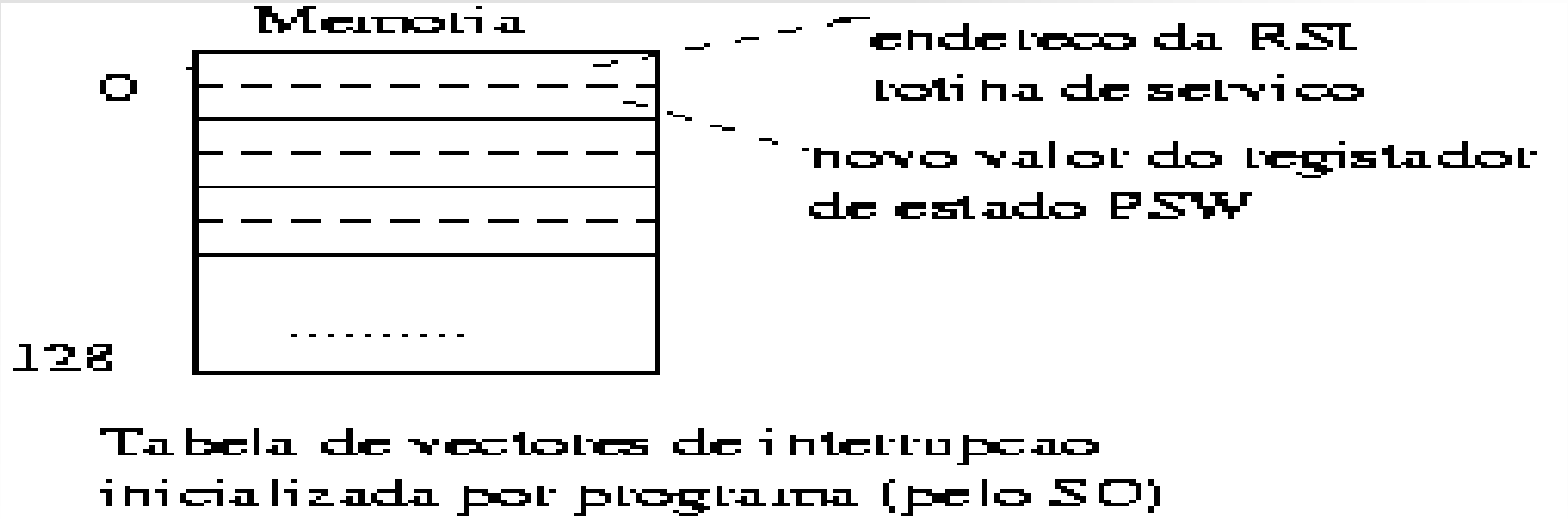
-- salvaguarda o estado corrente (mínimo) do CPU na pilha:

ProgramCounter + ProcessorStatusWord

-- a interface que pediu atenção, envia o seu identificador para o CPU

-- o CPU usa esse número como índice de uma tabela de 64 vectores de interrupções: → cada entrada, inicializada pelo SO, contém o endereço inicial da rotina de serviço de interrupções, correspondente a cada periférico.

No fim da execução da rotina de serviço, uma instrução máquina RTI (Return from Interrupt) desempilha os valores salvaguardados do PC + PSW



Prioridades no atendimento dos pedidos de interrupção:

-- durante a execução de cada instrução, podem ocorrer múltiplos pedidos de interrupção, vindos de distintas interfaces:

-- quando o CPU decide aceitar um pedido, escolhe o pedido mais prioritário.

-- durante a execução de cada RSI, o mais seguro é não aceitar novos pedidos de interrupção, enquanto não acabar essa RSI

-- mas… em caso de periféricos ´´muito rápidos´´ e/ou situações de controlo em tempo real…

→ pode ser necessário deixar interromper a execução de uma RSI para atender um pedido de um periférico mais prioritário

Múltiplas linhas de pedidos de interrupções:

-- o Bus pode ter múltiplas linhas físicas de pedidos de interrupções

-- a cada linha está ligado um conjunto de interfaces de periféricos

-- cada linha física tem associado um nível de prioridade

Um modelo genérico para gerir múltiplos níveis de prioridade

-- cada interface de I/O tem associado um nível de prioridade **i** (0..7)

-- quando um programa é posto em execução, é-lhe associado um nível de prioridade **x** (entre 0 e 7)

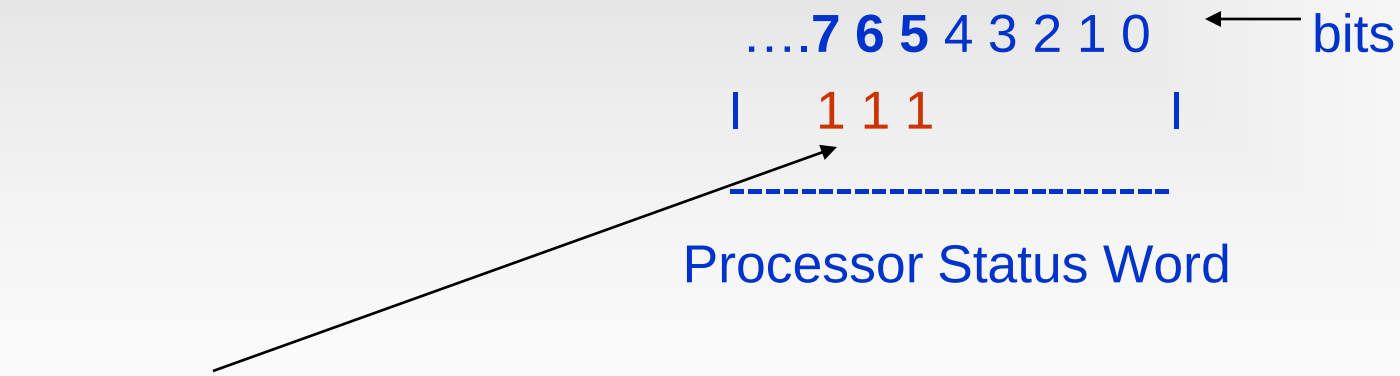
→ um programa em execução só pode ser interrompido por uma interface que tenha um nível **i > x**

→ se pusermos **x = 7**, a execução do programa não é interrompível

(equivale, no 80x86, a ter InterruptFlag = 0)

-- o nível **x** do programa corrente está definido por 3 bits do registador de estado/flags do CPU:

-- exemplo do PDP-11:



No PDP-11:

-- cada interface é fisicamente ligada a um de 4 níveis de prioridades, correspondentes a 4 linhas físicas do Bus, para pedidos de interrupção:

BR4, BR5, BR6, BR7 -- dentro de cada nível as interfaces estão ligadas em *daisy-chain*

exemplo de associações:

BR4 – terminal, impressora

BR5 – disco, linha de comunicação

BR6 – temporizador

Condições para que o CPU aceite um pedido de uma dada interface i:

- **bit IEN = 1 na porta de controlo desta interface**
- **bit READY = 1 na porta de estado desta interface**
- **nenhuma interface com BR > Br(i) tem um pedido pendente**
- **nenhuma interface com BR = Br(i), mais ´´próxima do CPU´´ tem um pedido pendente**
- **o programa corrente tem menor prioridade: PSW[5..7] < BR(i)**

Quando o CPU aceita um pedido:

- a interface que foi escolhida identifica-se, pondo um código no Bus
- o CPU obtém o vector:
  - Novo_PC → endereço da Rotina de Serviço RSI
  - Nova_PSW → define a prioridade a que a Rotina RSI se executa:
    - deve ser **>=** à prioridade da interface que fez o pedido
- o CPU empilha PSW e PC
- afecta PC e PSW a partir do vector da RSI

…. Equivale a saltar para o endereço inicial de RSI

Nos processadores 80x86: 2 níveis/linhas de prioridades, RSI vectorizadas

-- 2 linhas de pedido de interrupção, no Bus:

NonMaskableInterrupt: aceite logo que reconhecido pelo CPU

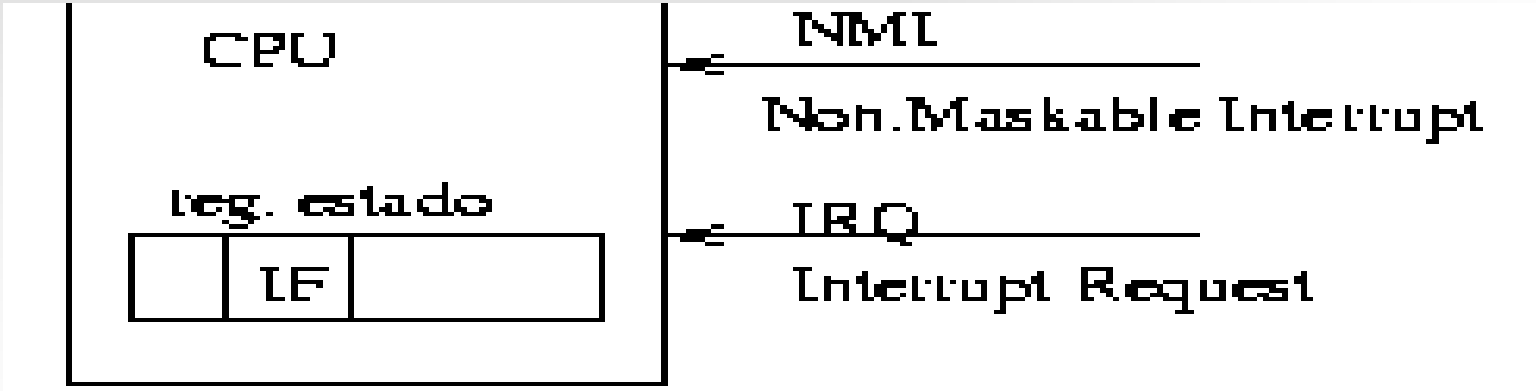
→ para ligar temporizadores

→ para indicação de falha de energia

→ situações de emergência/pânico

InterruptRequest: aceite só se a flag IF do CPU estiver a 1

-- Em cada nível, componentes externos podem arbitrar múltiplos periféricos.



NMI nao se podem inibir e sao mais prioritarios do que os pedidos da linha IRQ

Ao aceitar um pedido de interrupção, o CPU:

-- envia um sinal InterruptAcknowledge pelo Bus

-- o hardware (árbitro) externo ao CPU coloca, no Bus de dados, um valor de 8 bits, com o código da interface mais prioritária que fez um pedido de interrupção.

Cada tipo de interrupção tem um código n de 8 bits: 0..255

O CPU usa o valor n como o índice da entrada da **Tabela de Vectores de Interrupções**, que contém o endereço inicial da Rotina de Serviço de Interrupção: os novos valores CS:IP

Acções do CPU no atendimento de um pedido de interrupção:

inibe o atendimento de novos pedidos (equivale a IF = 0)

empilha FlagsRegister, CS e IP

carrega CS e IP com os novos valores

Acções no atendimento de uma interrupção de código n:

-- por hardware:

SP := SP – 2;
Mem[SP] : = FlagsRegister;

TrapFlag := 0; InterruptFlag := 0;

SP := SP – 2;
Mem[SP] := CS;

SP := SP – 2;
Mem[SP] := IP;

IP := OffsetdoVector(n)
CS := EnderecoBaseSegmentodoVector(n)
→ faz saltar para a RSI…

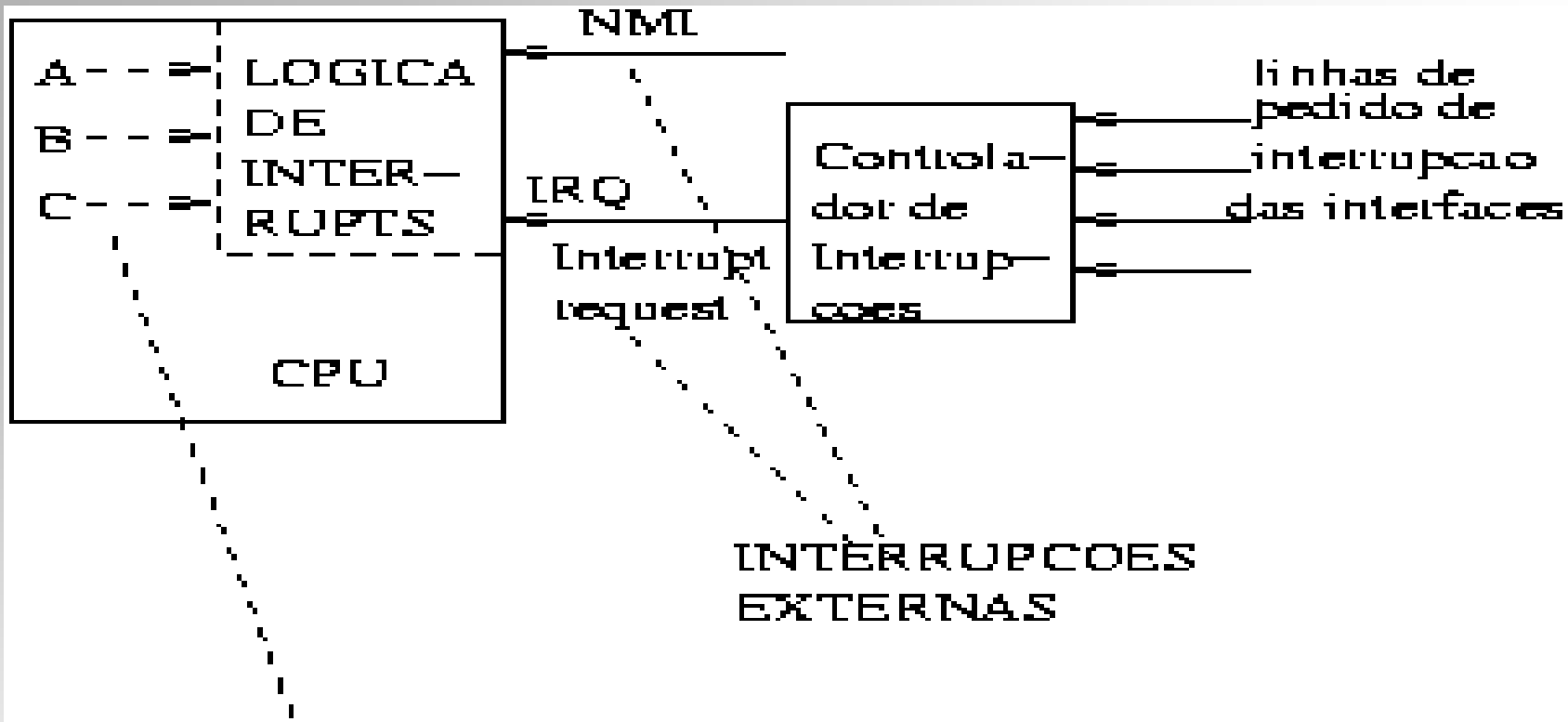
-- por software:

… as acções da RSI …

As posições de endereços 0 até ao endereço 4*255 de Memória estão reservadas para os vectores das RSI dos diversos tipos de interrupções.

Tabela de Vectores de Interrupção:

Índice	Endereços da entrada da tabela	Conteúdo da entrada
0	0 a 3	Endereço de RSI
1	4 a 7	Endereço de RSI
2	8 a 11	Endereço de RSI (NMI)
….	…	…
n	4n a 4n+3	Endereço de RSI (ocupa 4 bytes: CS:IP)

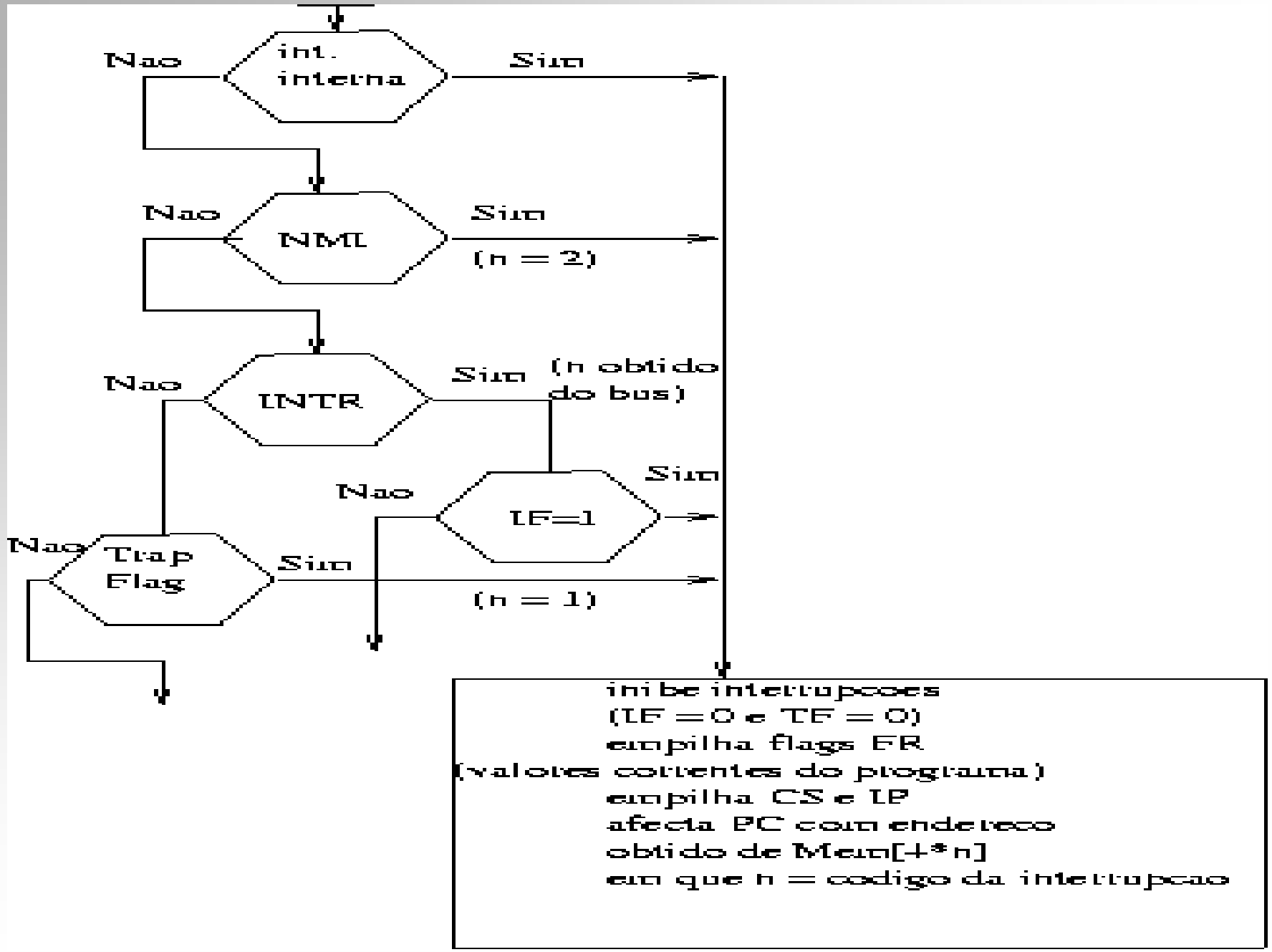


A: INT n

B: overflow e divisao por 0

C: flag Trap TF

TIPOS DE INTERRUPCOES INTERNAS


int. interna
Nao
Sim
NMI
Nao
Sim
(n = 2)
INTR
Nao
Sim
(n obtido do bus)
IF=1
Nao
Sim
Trap Flag
Nao
Sim
(n = 1)
inibe interrupcoes
(IF = 0 e TF = 0)
empilha flags FR
(valores correntes do programa)
empilha CS e IP
afecta PC com endereco
obtido de Mem[4*n]
em que n = codigo da interrupcao

RotinaServiçoInterrupção:

é invocada com IF = 0 (o hardware garante isso)

deve salvaguardar os registadores do CPU

o tratamento, em geral, consiste em aceder às portas da interface

no fim: a instrução IRET (InterruptReturn)

desempilha IP, CS e FlagsRegister

NOTA:

o atendimento de interrupções pode ser ligado após o tratamento, com a instrução STI (põe IF = 1)…

Interrupções Internas

São **geradas pela execução das próprias instruções máquina:**

-- em situações de erro /excepção:

-- violação de protecções

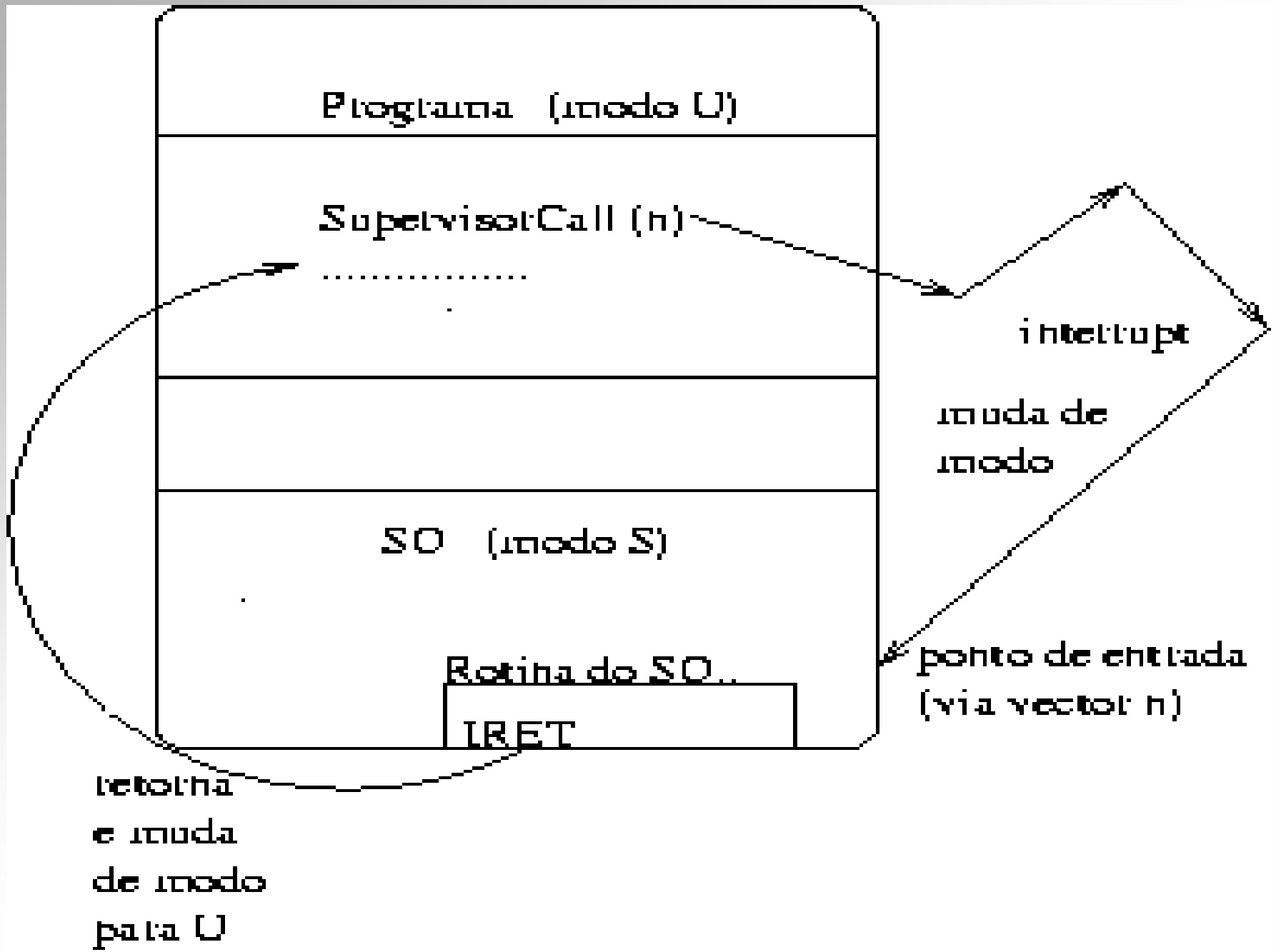
-- divisão por zero

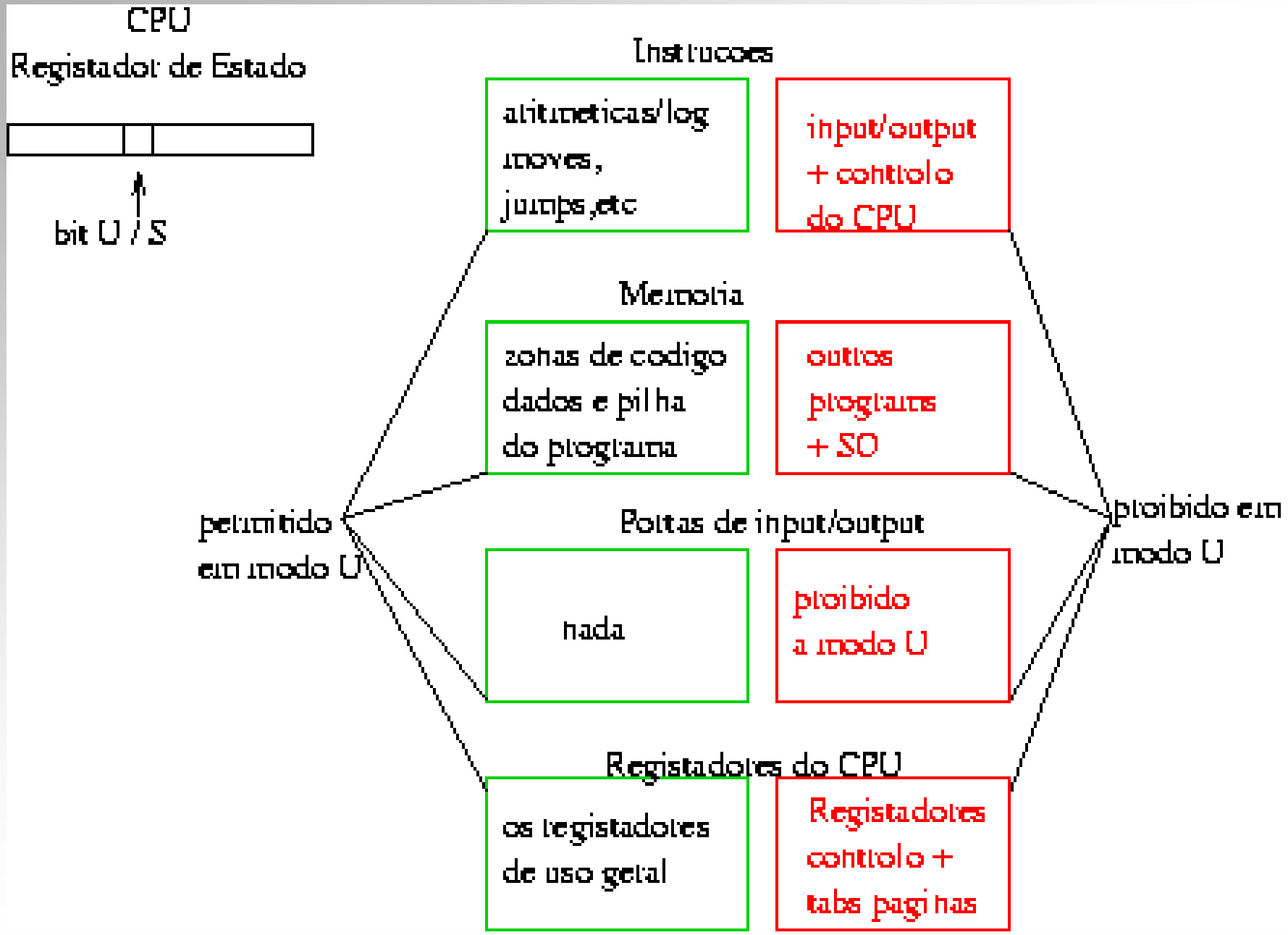
-- gestão do acesso à memória virtual (a ver mais adiante)

-- em modo de execução de instruções passo a passo

-- para chamar funções do SO, de forma controlada

São tratadas como as interrupções externas (à parte não serem dependentes do estado de IF para o seu atendimento).


Programa (modo U)
SupervisorCall (n)
.....
interrupt
muda de
modo
SO (modo S)
Rotina do SO
IRET
ponto de entrada
(via vector n)
retorna
e muda
de modo
para U


CPU
Registador de Estado
bit U / S
Instrucoes
aritmeticas/log
moves,
jumps,etc
input/output
+ controlo
do CPU
Memoria
zonas de codigo
dados e pilha
do programa
outros
programas
+ SO
permitido
em modo U
proibido em
modo U
Portas de input/output
nada
proibido
a modo U
Registadores do CPU
os registadores
de uso geral
Registadores
controlo +
tabs paginas

Trap Flag TF

O CPU gera uma interrupção com código 1, após acabar a execução de cada instrução máquina.

É usada para suportar a depuração (*debugging*) de programas com execução passo a passo (*single stepping*).

Entrada INTR (InterruptRequest):

é sensível ao nível

o CPU aceita um pedido de interrupção se INTR = 1 e IF = 1

ao aceitar: o CPU põe automaticamente IF = 0

-- para impedir que a RSI fosse continuamente interrompida para mesma condição de interrupção que está a tratar…

→ logo que a RSI puser a flag READY = 0 da interface,

→ tem-se a linha INTR = 0

pelo que RSI já pode decidir pôr IF = 1 e aceitar interrupções dentro de RSI ou deixar IF = 0 até se completar a RSI.

Entrada NMI (NonMaskableInterrupt):

é sensível à transição

Como não se pode inibir o seu atendimento (via IF), há que impedir que estivesse continuamente a gerar o mesmo pedido…

Um 2º pedido em NMI só é atendido, depois de ´´limpa´´ a condição que gerou o 1º pedido, o que permitirá detectar uma 2ª transição.

Acesso directo a memória

a) Periféricos ´´lentos´´ e orientados para o byte:

geram 1 pedido de interrupção por carácter transferido

deixam muito tempo disponível ao CPU…

compatível com um modelo de programação:

-- mover 1 byte da interface para o CPU

-- movre o byte do CPU para um buffer em Memória

b) Periféricos ´´rápidos´´ e orientados para o bloco

exemplo blocos de 2Kilo ou 4 Kilo Bytes

1ª fase: aceder ao bloco seleccionado

-- comando para mover cabeça de R/W do disco

-- ao fim do comando: READY = 1 → gera interrupt

2ª fase: desencadear transferência de bytes

→ a interrupção por cada byte transferido é inconveniente

DMA –Direct Memory Access:

Permite a uma interface de um periférico controlar a transferência directa dos dados de/para a Memória

-- Habitualmente é o CPU que controla o aceeso a Memória:

posicionando linhas de endereço, controlo e dados

-- a interface DMA, periodicamente, pede o acesso a Memória:

A lógica de controlo do bus (Árbitro):

arbitra os pedidos de acesso à Memória, da parte do CPU e do DMA →dando prioridades aos pedidos do DMA:

em caso de conflito: a execução do CPU é atrasada.

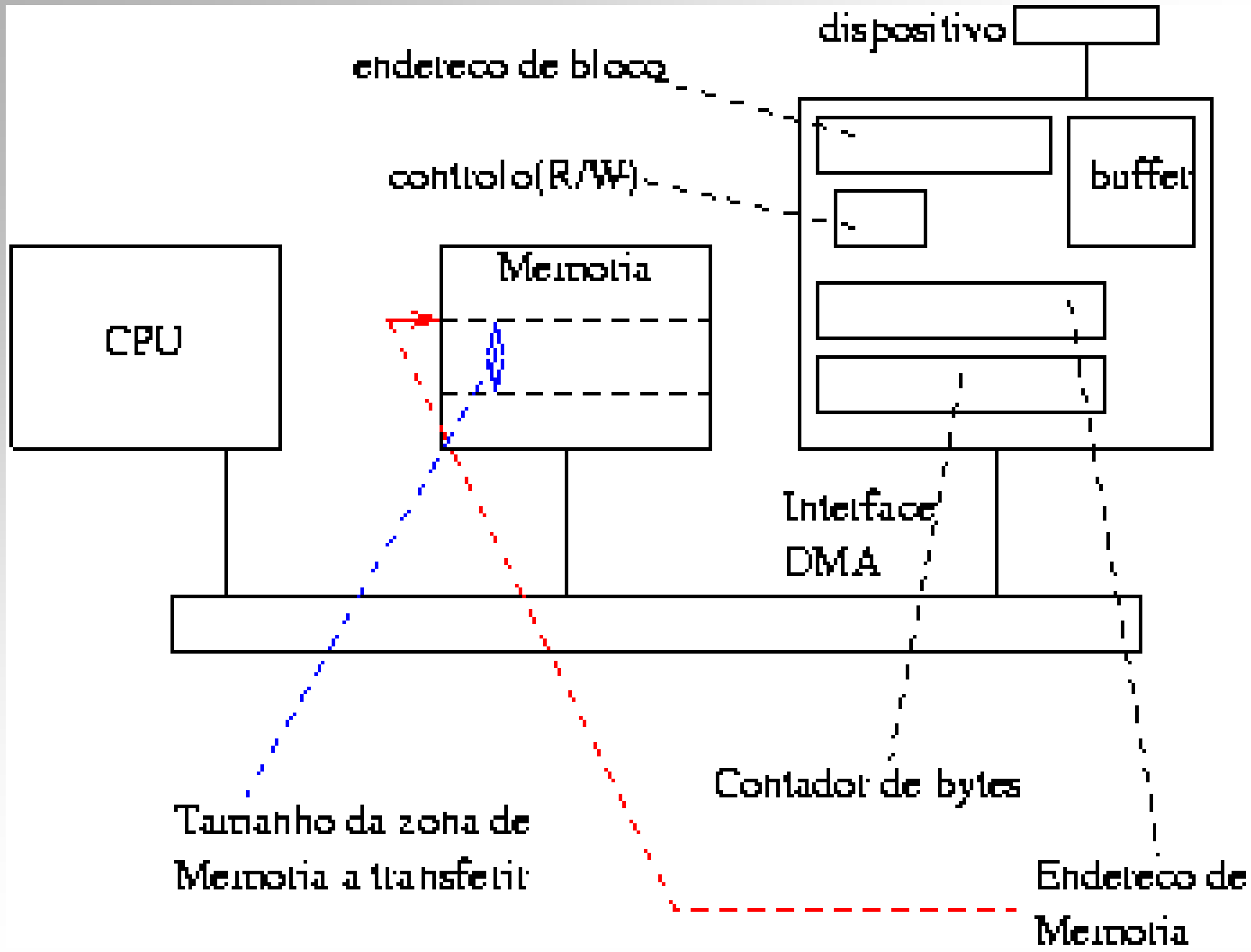
Modelo de programação de uma interface DMA:

Na interface:

- uma porta de controlo:
  - bit START: para activar a operação
  - bit R/W: para indicar Read ou Write de Memória
  - bit IEN: para que a interface gere pedidos de interrupção
- uma porta de estado:
  - bit READY: indica fim de operação
- uma porta EndereçoMemória:
- uma porta EndereçoDisco:
- uma portaContadorBytes:

Exemplo:

- portaEnderecoMemoria := EnderecoM
- portaEnderecoDisco := <disco, pista, sector>
- portaContadorBytes := numero_bytes_a_transferir
- R/W := 1; IEN := 1; START := 1


dispositivo
endereco de bloco
controlo(R/W)
buffer
CPU
Memoria
Interface
DMA
Tamanho da zona de
Memoria a transferir
Contador de bytes
Endereco de
Memoria