Arquitetura de Computadores

MiEI – 2016/17

DI-FCT/UNL

Aula 16

Exemplo – Intel nos PC

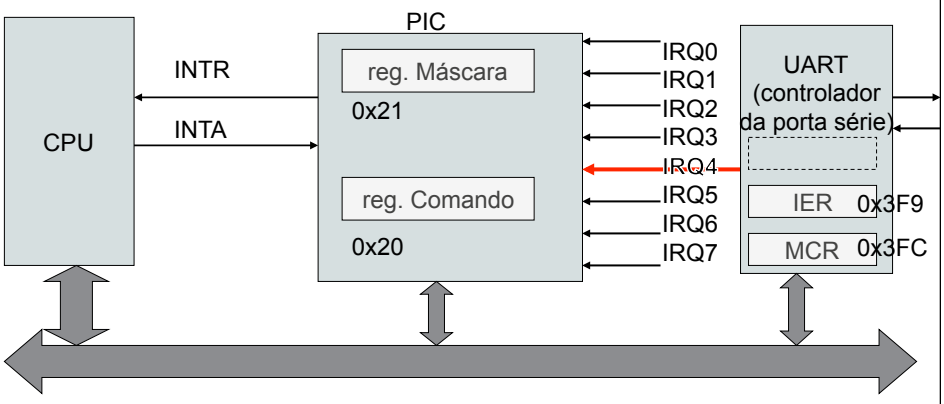
- Suporta até 255 tipos de interrupções vetorizadas
  - o vetor de endereços está na memória
- Tem uma linha de interrupções gerais externas
  - a flag IF é alterada pelas instruções STI e CLI
- Usa um controlador de interrupções programável -- PIC (8259A ou equivalente)
  - Identifica o tipo de cada interrupção, depois usado como índice no vetor de interrupções
  - Define as prioridades entre interrupções
- Instrução de interrupção por software: INT nº
- O retorno da rotina de tratamento é feito pela instrução IRET

Sequência de eventos

- Periférico envia interrupção
- 8259A aceita (ou não) a interrupção
- 8259A determina a prioridade
  - Se estiver em curso um tratamento de prioridade igual ou superior este fica pendente
- 8259A alerta o CPU (ativa o pino INTR)
- CPU faz "acknowledge" (INTA)
- 8259A coloca o identificador correspondente no bus de dados
- CPU salta para a rotina de tratamento de interrupções indicada por: `vetorIntr[identificador]`

Interrupções na Porta Série 1

- COM1:
  - tem atribuído o IRQ4; identificação 12



Interrupções na Porta Série 1

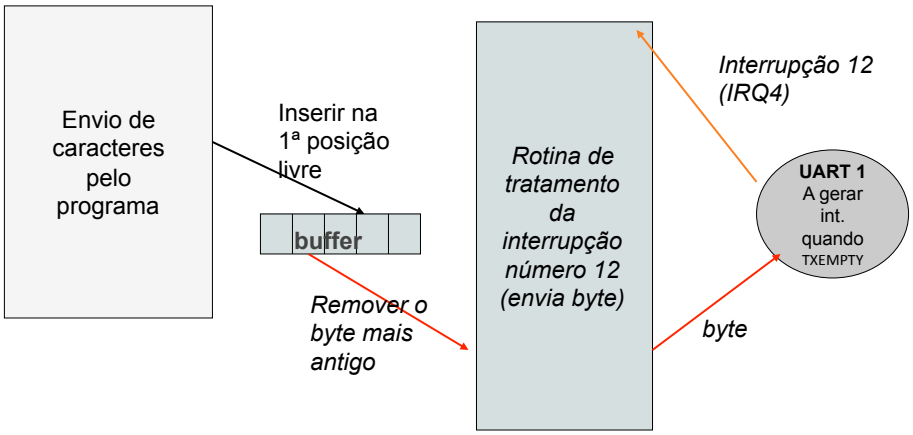
- Programação da UART para gerar interrupções:
  - Ligar interrupções no MCR (0x3FC):
    - Colocar a 1 o bit 3 do registo
  - Programar quais as interrupções pretendidas no registo IER (*Interrupt Enable Register*)(0x3F9):
    - quando chega um carácter (RXREADY): Colocar a 1 o bit 0
    - quando o registo de transmissão fica vazio (TXEMPTY) ou seja, pronto a enviar: Colocar a 1 o bit 1

Operação por interrupções

- Inicia-se o vetor de interrupções, o PIC e o UART
  - A porta série 1 tem associada a interrupção 12
- Quando existe uma transferência para fazer, o controlador da porta série passa a indicar (por interrupção) quando se pode fazer a leitura/escrita nos registos de dados
- É a rotina de tratamento da interrupção que escreve (OUT) no porto de Dados TX ou lê (IN) do porto de Dados RX
- A rotina usa um **buffer** como origem ou destino do próximo byte a enviar/recebido
- O programa principal usa o mesmo buffer para enviar/receber os dados
  - **Buffer** é o meio de comunicação entre programa principal e a rotina das interrupções

Transmissão com interrupções

- Em qualquer instante o *buffer* pode ser manipulado pelo programa ou pela rotina de atendimento → problemas?



Transmissão com interrupções

```
send_serial(ch){
   while bufFull()
       esperar ou liberta CPU;
   bufPut(ch);
   if inter. desligadas
       ligar inter. TXEMPTY
}
```

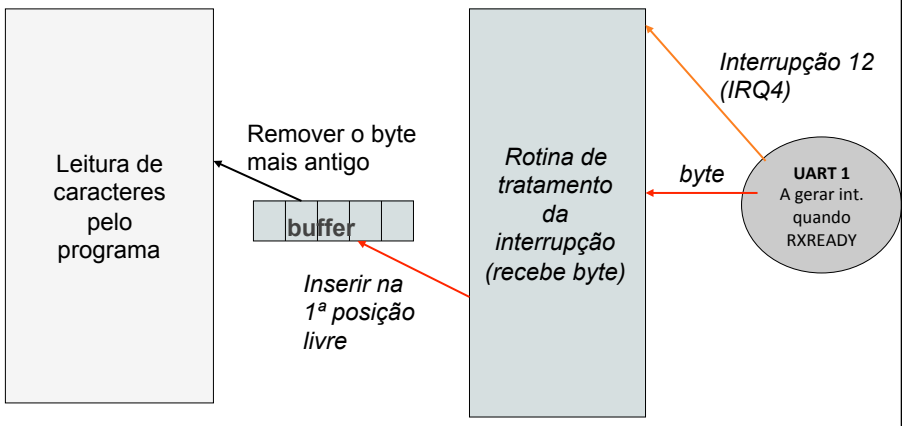
```
Rotina de tratamento de
interrupções TXEMPTY{
   ch = bufGet();
   outByte( TXDATA, ch );
   if bufEmpty()
      desligar inter. TXEMPTY
   outByte(PIC_COMMAND, EOI);
}
```

Notas:

- A rotina send_serial necessita de espaço no "buffer" (esta situação não deve ser comum)
- A rotina send_serial verifica se as interrupções necessárias estão desligadas; se tal acontecer liga as interrupções na UART
- A rotina de interrupções desliga as interrupções da UART se o "buffer" ficar vazio

Recepção com interrupções

- Em qualquer instante o *buffer* pode ser manipulado pelo programa ou pela rotina de atendimento → problemas?



Recepção com interrupções

```
unsigned char recv_serial(){
   while bufEmpty()
       esperar ou libertar CPU;
   ch = bufGet();
   return ch;
}
```

```
Rotina de tratamento de
interrupções RXREADY{
   ch = inByte( RXDATA );
   if bufFull()
      /*Erro! Não há espaço no
        buffer mas não se pode
        esperar … */
   else bufPut(ch);

   outByte( PIC_COMMAND, EOI)
}
```

Notas:

- a rotina recv_serial espera (com as interrupções ligadas) que haja um elemento no "buffer"
- A rotina de interrupções pode encontrar o " buffer" cheio; se tal acontece não pode ficar em espera ativa (porquê ?)… perde-se o byte que chegou…

Implementação do Buffer de I/O

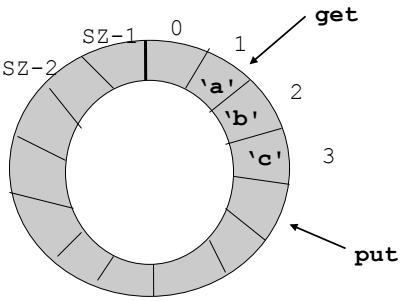
- Características:
  - Zona de memória acessível ao programa principal e à rotina de tratamento da interrupção
    - Pode ser indiretamente (p.ex. o programa recorre a chamadas ao sistema)
  - Capacidade suficiente para que o programa e o periférico trabalhem dessincronizados
    - Não perder os dados que "entram" do periférico
    - Manter o periférico ocupado com as "saídas"
  - Disciplina FIFO (First-In-First-Out)
  - Operações de pôr e tirar: bufPut() e bufGet(); testar buffer vazio/cheio: bufEmpty() e bufFull()

"Buffer" circular - exemplo

```
unsigned char buffer[SZ];
int put = 0;  // 1ª casa livre
int get = 0;  // casa ocupada há mais tempo
int nc = 0;   // nº de bytes no buffer
```

```
void bufPut( unsigned char c)
{  /* assume que buf não está cheio */
  buffer[put] = c;
  put = (put + 1) % SZ;
  nc ++;
}
```

```
unsigned char bufGet(void)
{  /* assume que buf não está vazio */
  unsigned char x = buffer[get];
  get = (get + 1) % SZ;
  nc --;
  return x;
}
```




```
int bufFull()
{  return (nc == SZ);
}
```

```
int bufEmpty()
{  return (nc == 0);
}
```

Possível problema: Actualização do nº de bytes no "buffer" (1)

- bufPut incrementa o número de bytes no buffer: *nc ++*
  - Suponhamos que o compilador traduz para

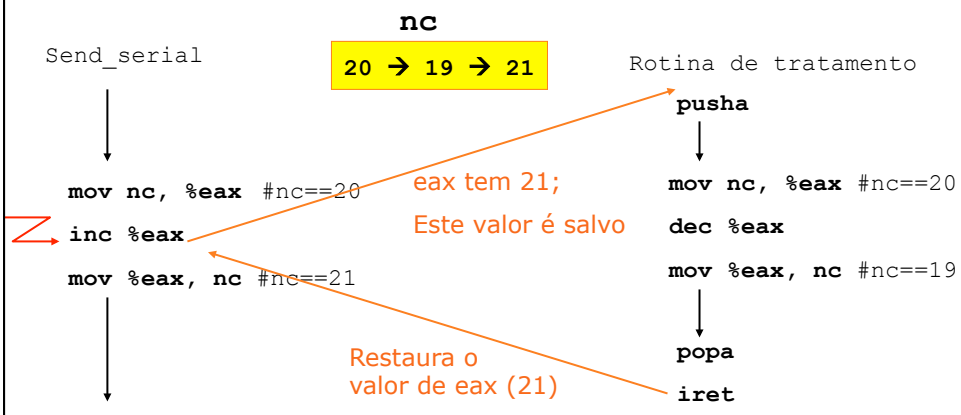
```
mov nc, %eax
inc %eax
mov %eax, nc
```

- A **rotina de atendimento de interrupções** usa bufGet que decrementa o número de bytes no buffer: *nc --*
  - Suponhamos que o compilador traduz para

```
mov nc, %eax
dec %eax
mov %eax, nc
```


Possível problema: Actualização do nº de bytes no "buffer" (2)

- Suponhamos que ***nc*** é 20 e que, enquanto a rotina escrever_série deposita um carácter, ocorre uma interrupção motivada pelo facto do registo TX ter ficado vazio …



Actualização do nº de bytes no "buffer" (3)

- A *nc* ficou com 21 o que está errado e vai provocar problemas no futuro para *bufEmpty()* e *bufFull()*
- Isto aconteceu porque a acção de actualização da variável foi interrompida a meio e foi chamada uma rotina que actualiza a mesma variável → ***race condition (corrida)***
- A variável *nc* é partilhada pela rotina send_serial/recv_serial e pela rotina de tratamento de interrupções. A sua actualização não pode ser interrompida – constitui o que se chama uma **secção crítica**
- Para resolver isto é preciso alterar as rotinas para que não sejam interrompidas na alteração de *nc*:
  - Por ex. usando as instruções máquina CLI e STI
  - Assim garante-se que *nc* é atualizado corretamente

Transferência de dados conduzida por interrupções

- Nas transferências de bytes para periféricos, o CPU só está envolvido quando é realmente necessário → executando o código da rotina de tratamento que faz IN/OUT desses bytes para o controlador do periférico
  - não existe CPU em espera…
- No caso dos blocos dos discos?
  - Podemos transferir byte-a-byte, ou 4 ou 8 bytes de cada vez, usando a largura do bus de dados
- *Mas poderá ser melhor?*
  - *Nos periféricos orientados ao bloco, pedir logo a transferência do bloco completo de bytes, num único pedido ao controlador?*

Unidades de transferência: em bytes

- Exemplos: porta série (RS232), teclados
- Um teclado, disponibiliza um byte com o carácter premido; a porta série transfere um único byte
- O software tem de manipular cada um desses bytes individualmente
- Podemos "ver" o teclado como um periférico de leitura que providencia uma sequência (*stream*) de bytes
- Podemos "ver" a porta série como um periférico de leitura e escrita de sequências de bytes

Unidade de transferência: em blocos

- Exemplo: disco, DVD, pen, etc
- O periférico só permite leitura/escrita de um bloco de bytes (512 bytes, 1K, etc)
- Podemos "ver" estes periféricos como endereçáveis mas orientados a blocos
  - Cada endereço representa um bloco
- Como o software copia esses blocos de memória para o respectivo controlador e vice-versa?
  - Em bytes/words etc por espera activa?
  - Em bytes/words etc usando interrupções por cada um?

Transferência de blocos de bytes

- Periféricos orientados ao bloco. Exemplo: enviar 1000 bytes

```
enviar_bloco()
    por_bloco_no_buffer()
    ligar_transf_intr()
    aguarda_transferencia_p_controlador()
    manda_escrever_bloco()

RotinaServiço:
    envia prox. byte (ou word ou dword) do buffer p/ controlador
    se último, desliga_transf_intr()
```

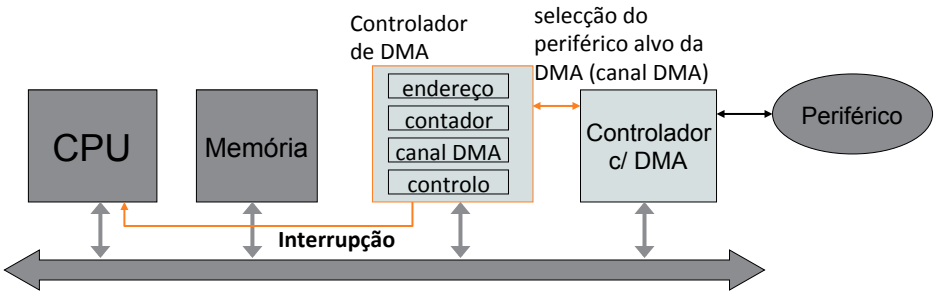
O CPU pode ser interrompido até 1000 vezes para executar a RotinaServiço !

Acesso directo à memória *Direct Memory Access* (DMA)

- As E/S usando interrupções ainda requerem uma intervenção ativa do CPU nestes periféricos
  - Para um bloco de dados o CPU precisa de intervir muitas vezes
  - A resposta a estes problemas é dispensar, nestes casos, a necessidade de interrupções a cada byte/word/dword
- Um novo dispositivo *hardware* que permita transferências entre Memória e Controladores sem intervenção do CPU: Acesso Direto à Memória

Controlador de DMA. Exemplo

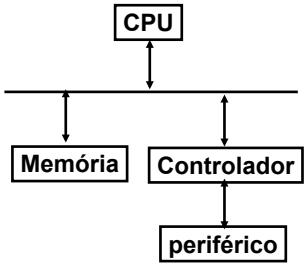
- Registos de um controlador genérico:
  - Endereço – endereço de memória
  - Contador – número de bytes a transferir
  - Canal DMA – *(DMA channel)* identifica o periférico alvo
  - Controlo – comandos, por exemplo: sentido da transferência



Operação com DMA

- O CPU programa o controlador de DMA com a transferência pretendida:
  - Leitura da memória ou escrita em memória
  - Endereço (canal de DMA) do controlador do periférico
  - Endereço do bloco de memória central, onde estão/para onde vão, os dados a transferir
  - Número de bytes a transferir
- O CPU volta ao processamento
- O controlador de DMA trata da transferência com o controlador do periférico
- O controlador de DMA pode "tomar conta" do bus (bus master) e aceder à memória e ao periférico
  - Compete com o CPU pelo uso do BUS
- O controlador de DMA envia uma interrupção quando termina toda a transferência (ou em caso de erro)

Transferência de dados baseada em interrupções sem DMA



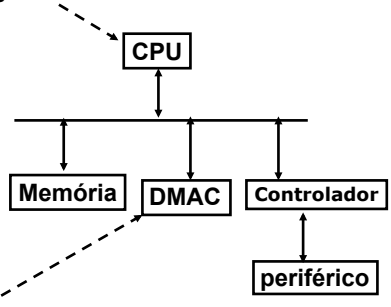
Os processos são parados (o CPU) durante a transferência (CPU a 1GHz e rot. tratamento com 100 instruções a 1 inst/Hz, 1000bytes):

1000 interrupções ➔ 1000 x a rotina de serviço custa ao CPU : 1 000*100 = 100 000 Hz

No melhor dos casos transfere $10^9/100 = 10^7 = 10$ Mbytes/s (mas depende também da velocidade do bus, periférico e memória)

Direct Memory Access

O CPU envia a direcção, end. inicial e nº de bytes ao controlador de DMA (DMAC). A seguir dá comando de "start".



DMAC faz a transferência e interrompe só no fim

Suponhamos que a preparação do DMA também "custa" 100 instr. A transferência de 1000bytes custa ao CPU: 100+100 = 200 instr.

A taxa máxima de transferência depende do bus, memória, do controlador de DMA e do periférico.
O CPU está livre para executar os programas.