Sistemas Embebidos

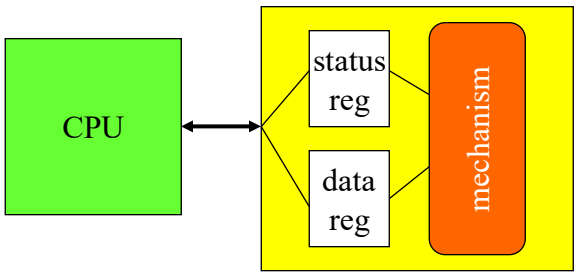
CPUs

Sumário - CPUs

- Input e output
- Interrupts
- Modo Supervisor, exceções, *traps*
- Co-processadores

Dispositivos de I/O

- Normalmente inclui alguns componentes não digitais
- Interface digital típico do CPU:

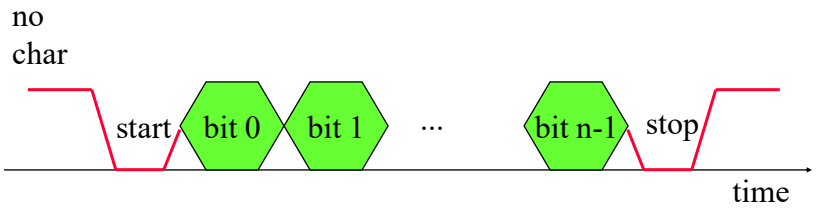


Aplicação: 8251 *UART*

- Universal Asynchronous Receiver Transmitter (*UART*) : fornece comunicação em série.
- Funções do 8251 estão integradas no chip de interface padrão dos *PCs* (portas de serie)
- Permite que sejam programados vários parâmetros de comunicação.

Comunicação em série

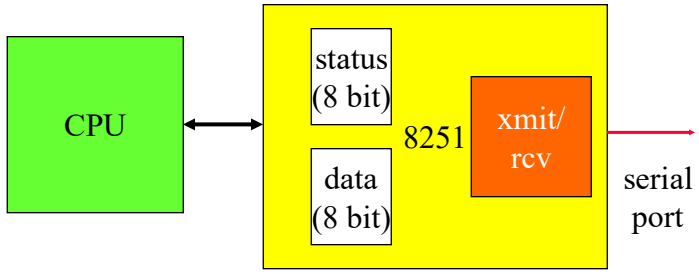
- Carateres são transmitidos separadamente
- Todos começam com um bit "0" e terminam com um bit "1":



Parâmetros de comunicação em série

- *Baud* (bit) *rate* (nº mudanças de sinal por segundo) (velocidade da comunicação)
- Número de bits por caratere.
- Com ou sem paridade (verificação de erros)
- Paridade par ou ímpar.
- Comprimento do bit de paragem (1, 1.5, 2 bits). (sinaliza o fim da comunicação)

Interface CPU 8251



Programação *I/O*

- I/O pode suportar dois tipos de instruções:
  - Instruções de propósito especial I/O;
  - Instruções load/store memória mapeada.
- *Intel x86* fornece instruções **`in, out`**. Maioria dos outros *CPUs* utilizam memória mapeada I/O.
- Instruções I/O não impedem o uso de memória mapeada I/O.

Memoria mapeada I/O ARM

- Define location for device:

```
DEV1 EQU 0x1000
```

- Read/write code:

```
LDR r1,#DEV1 ; set up device adrs
LDR r0,[r1] ; read DEV1
LDR r0,#8 ; set up value to write
STR r0,[r1] ; write value to device
```


Peek e poke

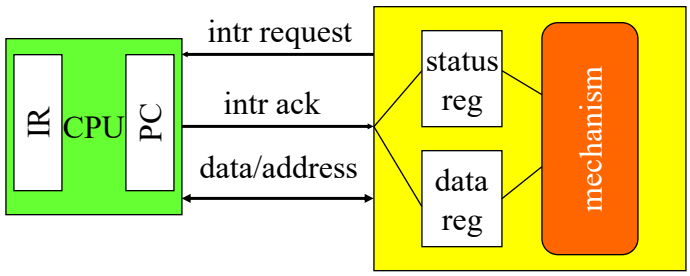
- Traditional HLL interfaces:

```
int peek(char *location) {
  return *location; }

void poke(char *location, char
  newval) {
  (*location) = newval; }
```

Interface de Interrupções *I/O*

- As interrupções permitem que um dispositivo altere o fluxo de controlo do CPU
  - Provocam a chamada de uma sub-rotina *(ISR)* para gerir o dispositivo



ISR = interrupt service routine

11

Comportamento de Interrupção

- Baseado no mecanismo de chamada de uma sub-rotina
- A interrupção força a próxima instrução a ser uma chamada a uma sub-rotina para uma localização predeterminada
  - O endereço de retorno é gravado para permitir a retoma ao *foreground program*.

12

Interface físico interrupção

- O *CPU* e o dispositivo estão conectados pelo barramento do *CPU*
- Negociação do *CPU* e dispositivo (*handshake*):
  - dispositivo alega pedido de interrupção;
  - *CPU* reconhece a interrupção (*acknowledge*) quando puder lidar com ela.

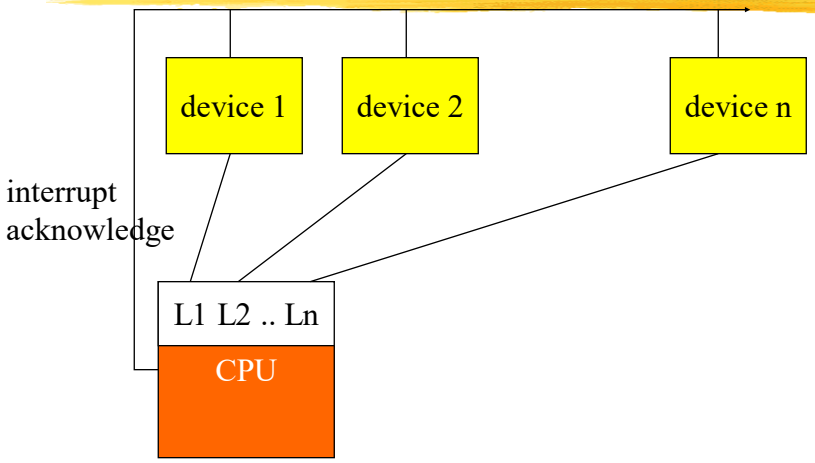
13

Prioridades e vetores

- Dois mecanismos permitem tornar as interrupções mais específicas:
  - Prioridades determinam quais as interrupções que acedem ao *CPU* mais rapidamente
  - Vetores determinam que código é chamado para cada tipo de interrupção
- Os mecanismos são ortogonais: a maioria do *CPUs* fornecem ambos

14

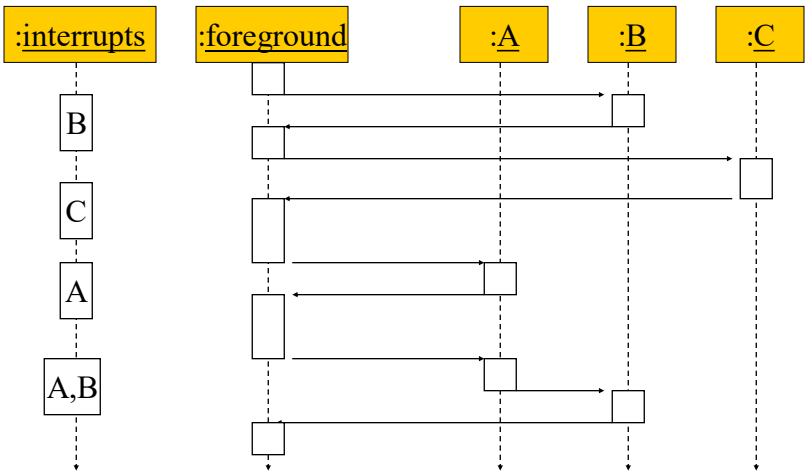
Interrupções prioritárias



Prioridades de Interrupções

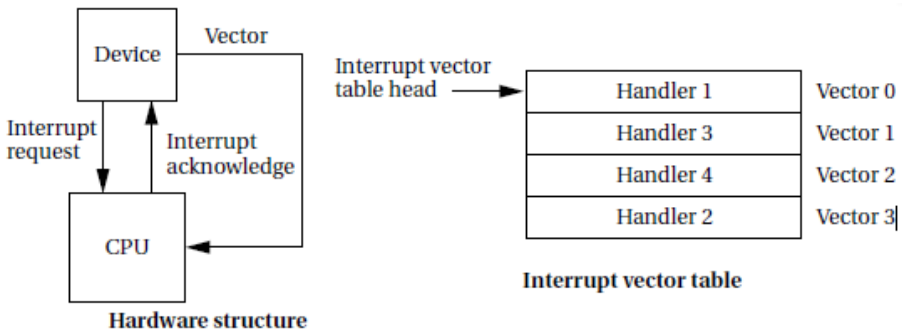
- *Masking*: interrupções com prioridade inferior à interrupção atual não são reconhecidas enquanto a interrupção não estiver completa
- *Non-maskable interrupt* (*NMI*): prioridade de maior valor, nunca é mascarada.
  - Frequentemente utilizada para *power-down*

Exemplo: *I/O* Prioritários

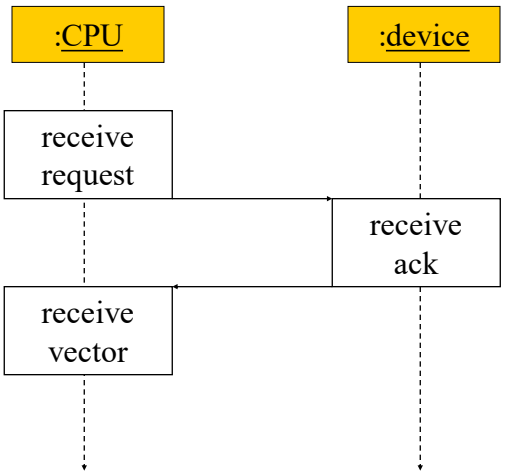


Vectores de Interrupt

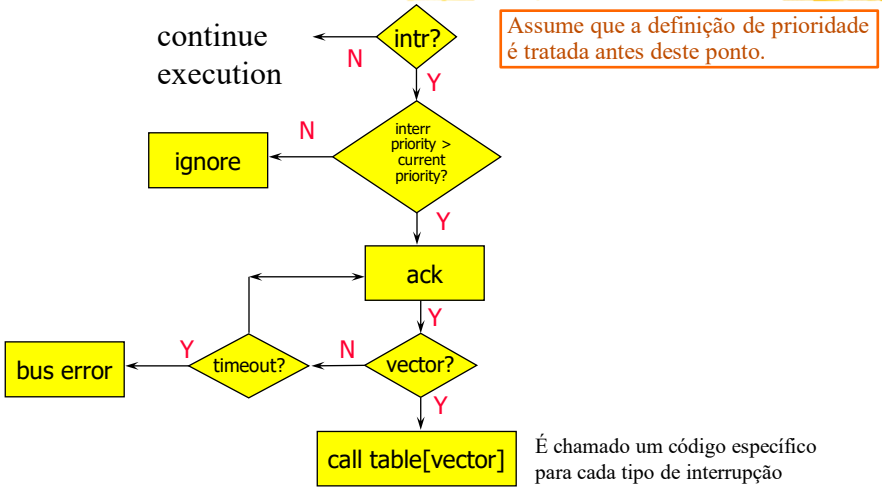
- Permite que diferentes dispositivos sejam manipulados por códigos diferentes.
- Tabela de vetores de Interrupts :



Atribuição de vetores de Interrupts



Mecanismo de interupções genérico



Sequência de Interrupção

1. CPU afirma o pedido (*ACK*)
2. Dispositivo envia o vetor
3. CPU chama o *handler*
4. Software processa o pedido
5. CPU restaura o estado ao programa *foreground*

Fontes da sobrecarga do interrupt

- Tempo de execução do *Handler*.
- Sobrecarga do mecanismo do *interrupt*.
- *Save* e *restore* dos registos
- Penalizações relacionadas com *pipeline*.
- Penalizações relacionadas com *cache*.

Interrupções do ARM

- ARM7 suporta 2 tipos de interrupções:
  - *Fast interrupt requests (FIQs)* – Interrupts mais rápidos, permitem atalhar algumas etapas do *handler*
  - *Interrupt requests (IRQs)*
- A tabela de interrupções inicia-se na localização 0

Procedimento de interrupções do *ARM*

- Ações do CPU:
  - Salva o PC. Copia CPSR para o SPSR.
  - Força os bits no CPSR a guardar a interrupção
  - Força o PC para o vetor
- Responsabilidades do *handler* :
  - Restaura o PC correto.
  - Restaura CPSR do SPSR.
  - Limpa (*interrupt disable flags)*.

SPSR - *saved program status register*
CPSR - *Current program status register*
PC *- program counter*

Latência das interrupções do *ARM*

- O pior caso de latência para resposta a uma interrupção é de 27 ciclos:
  - 2 ciclos para sincronizar o pedido externo
  - Até 20 ciclos para completar a instrução em curso
  - 3 ciclos para abortar dados
  - 2 ciclos para entrar no estado de *handling* da interrupção

Modo Supervisor

- Proporciona uma forma de proteção de memória entre programas
  - Impede a corrupção de memória
- Necessidade de modo supervisor para gerir os vários programas

Modo Supervisor do ARM

- Uso da instrução *SWI* (*software interrupt*) para entrada no modo supervisor, ex:

```
SWI CODE_1
```

- Define *PC* em 0x08
- O argumento para o *SWI* é passado com o código do modo supervisor
- Salva *CPSR* no *SPSR*

27

Exceção

- Exceção: erro detetado internamente
- As exceções são síncronas com instruções mas imprevisíveis
- O mecanismo de exceções está no topo do mecanismo de interrupções
- As exceções são habitualmente priorizadas e vetorizadas
- Um exemplo simples é a divisão por zero

28

Trap

- ***Trap*** (interrupção de software): exceção gerada por uma instrução/processo
  - Exemplos: Chamada do modo supervisor, acesso à memória inválido, etc.
- O ARM utilizada a instrução *SWI* para as ***traps***

29

Co-processador

- Co-processador: unidade de funcionamento invocada por uma instrução
  - Ex: co-processador de vírgula flutuante, *ALU*
- O ARM permite até 16 co-processadores
  - O co-processador de vírgula flutuante utiliza as unidades 1, 2

30

Sistemas Embebidos

Ficha de trabalho nº3

31