**Microcontroladores vs Microprocessadores**

|  |  |
| --- | --- |
| **Microprocessador** | **Microcontrolador** |
| * Circuito integrado com um ou mais CPU * Não tem memória interna (Além do banco de registos * Os barramentos estão disponíveis no exterior * Para obter um sistema complexo é preciso acrescentar RAM, ROM e periféricos * Para operar a altas frequências * Sistemas computacionais de uso geral | * Circuito integrado, inclui CPU, RAM, ROM, periféricos * Frequência de funcionamento normalmente baixa * Baixo consumo de energia * Disponibiliza uma grande variedade de periféricos e interfaces * Rapidez de resposta a tempo real * Utilizado para tarefas especificas |

**Sistema embebido**

Um sistema embebido é um sistema computacional especializado, tendo os seus próprios requisitos e executa apenas tarefas pré definidas.

Tem recursos limitados, em relação a um sistema computacional de uso geral. O que faz com que tenha um preço reduzido.

Pode ser implemento com base num microcontrolador ou fazer parte de um sistema computacional

**Microcontrolador**

Dispositivo programável que integra num único circuito integrado, 3 componentes fundamentais:

* Unidade de Processamento
* Memória (volátil ou não volátil
* Postos de I/O

Pode ainda ter periféricos, tais como:

* Timers
* Conversor A/D
* Serial I/O

Existem barramentos (dados, endereços e controlo) interligam entre si estes dispositivos não estando normalmente disponíveis no exterior. Externamente existem pinos que podem ser programáveis para diferentes funções

**Processo de desenvolvimento de aplicações para uC**

Existe o computador host e o computador target, sendo o host o que cria o programa e o host o que vai executar o código

Edição do programa numa linguagem de alto nível, (C ou Assembly)

Geração de código usando um cross-assembler/cross-compiler

Cross compiler é um compilador que gera o código no computador host e que depois cria um executável para o computador target

Transferência para a memória do microcontrolador (geralmente parra a não volátil) do código criado pelo cross-compiler/cross-assembler

Teste e depuração (debug)

**Transferência de programas para o microcontrolador**

Para fazer mos a transferência de um programa executável para o microcontrolador, podemos utilizar vários métodos:

* Programa monitor
  + É um programa que reside de forma permanente na memória não volátil do microcontrolador
  + Implementa outras funções no debug de novos programas
* Bootloader
  + É um programa que reside de forma permanente na memória permanente na memória do controlador e que disponibiliza apenas funções básicas de transferência e execução de um programa
* In-Circuit Debugger
  + Um dispositivo hardware controlado por software pelo host que permite a transferência de um programa num controlador

**Tecnologias de memória não volátil**

**ROM** – programada durante o processo de fabrico

**PROM** – Programmable Read Only Memory: programável uma única vez

**EPROM –** Erasable PROM: escrita em segundos, apagamento em minutos

**EEPROM –** Eletrically Erasable PROM -Efectuados no próprio circuito em que a memória está integrada, apagamento feito byte a byte, escrita mais lenta do que a leitura

**Flash EEPROM**  - Tecnologia mais rápida do que a EEPROM, pois o reset é feito por blocos (de 4kB por exemplo), o reset e a escrita são feitos no próprio circuito em que a memória está integrada, escrita mais lenta que a leitura

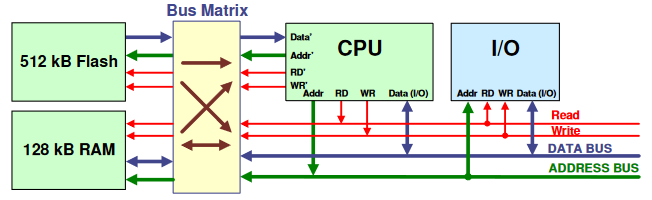
Uma imagem com texto

Descrição gerada automaticamente

Devido à elevada multiplexagem nos pinos, cada um dos 64 pinos pode ter 9 funções, que dependem da sua programção.

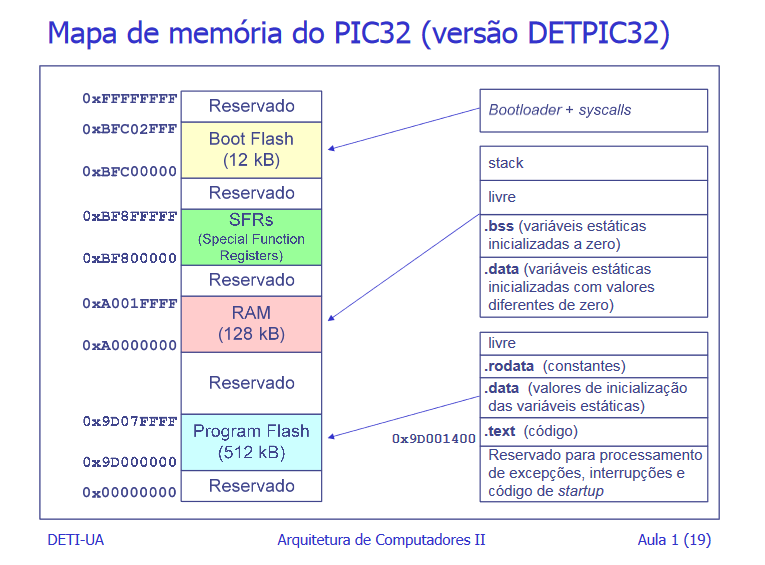
No PIC32 podemos ver um **Bus Matrix** que é um comutador de alta frequência, que funciona ao mesmo tempo que o CPU que tem como função estabelecer ligações ponto a ponto entre os módulos do microcontrolador, em particular, entre o CPU e a memória RAM ou entre o CPU e a memória Flash

O peripheral bus também liga ao Bux Matrix e pode ser configurado para trabalhar a uma frequência igual ou inferior à do CPU

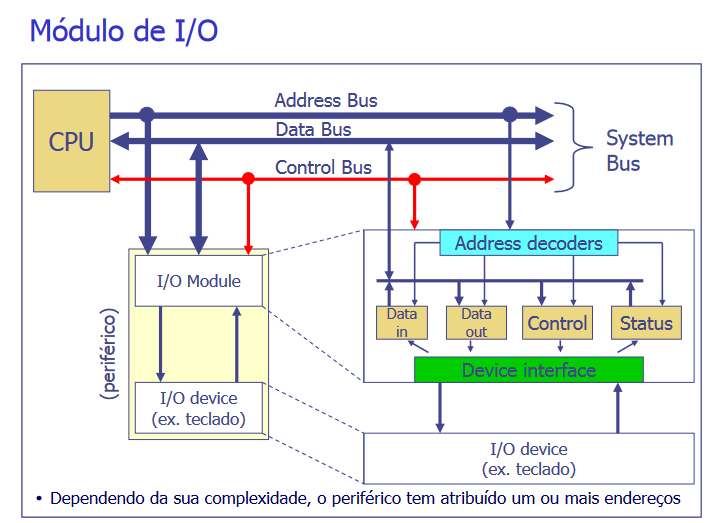


Com o Bus Matrix, o espaço de endereçamento aparece segundo o programador com um espaço unificado (instruções e dados no mesmo sitio). Assim, o CPU pode executar programas da Flash e da RAM. Para o programador, apesar da PIC ser baseado na arquitetura de Harvard, o PIC32 funciona como uma arquitetura de von Neumann:

Um único espaço de endereçamento onde residem dados e instruções



**Módulo de I/O**

****O módulo de I/O pode assim ser visto como um módulo de compatibilização entre as caraterísticas de funcionamento do sistema computacional e o sistema físico

A nível do hardware, o periférico liga—se ao sistema através de barramentos do mesmo modo que os outros dispositivos. Já a nível do software, o periférico liga-se a registos, disponibilizando ao programador a sua informação escondendo a complexidade e os detalhes da implementação do dispositivo periférico.

Assim, o processador apenas vê um modelo simplificado do periférico, escondendo os detalhes do funcionamento interno. Passando apenas a ser visto como o registo de dados, de controlo e de status.

**Data Register (Read/Write)** -> O processador coloca a memória para o periférico e onde lê os dados do periférico.

**Status Register (Read Only)** -> Registos que englobam um conjunto de bits que dão informação sobre o estado do periférico (Write only ou Read/Write) -> Registos onde o CPU escreve infomação sobre o modo de operação do periférico (comandos)

**Control Register (Write only ou Read/Write)** -> Registos onde o CPU escreve informação sobre o mode de operação do periférico (comandos)

É comum, um registo ter a capacidade de ser Control e Status, nesse caso, um dos bits está associado a cada um deles

**Protocolo de comunicação**

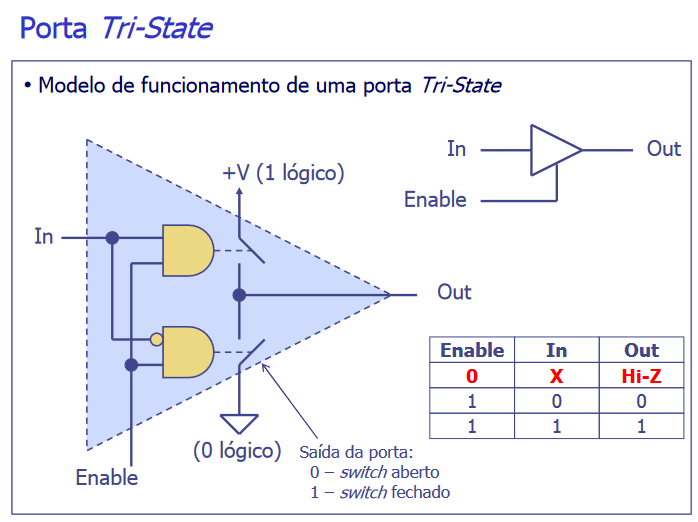
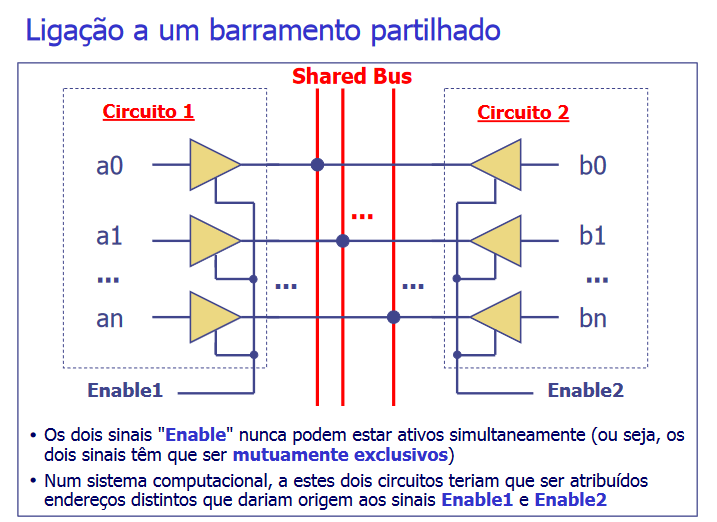
**Write** :== CPU 🡪 Periférico **Read** CPU 🡨 Periférico

Um operação de acesso do CPU a um periférico envolve:

- Usar o barramento de endereços para especificar o endereço do sipositivo a aceder

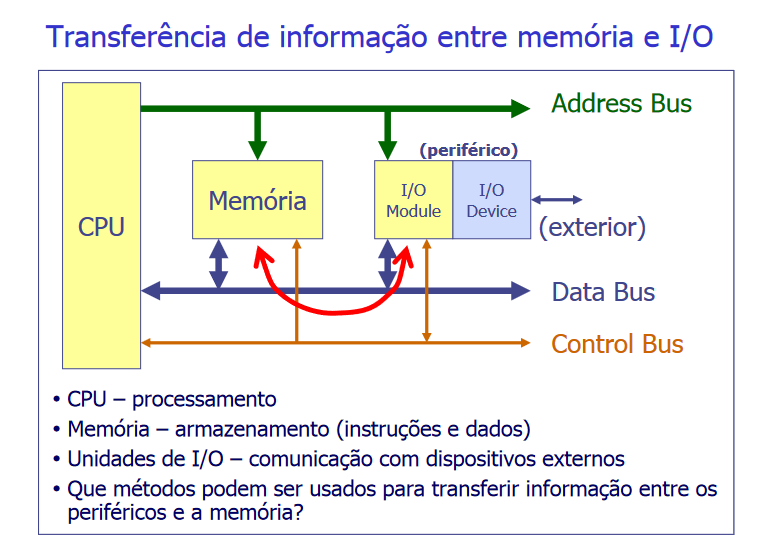
- Ler o valor de registo de controlo para saber se o processador vai ler ou escrever

- O barramento de dados assegura a transferência de dados



´

Aula 5, 6 e 7



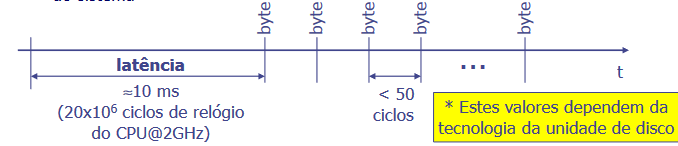
Transferência de informação entre memória e I/O

1. **Efetuar o pedido** à unidade de disco
2. **Esp**erar que a unidade de disco tenha a informaçãodisponivel na sua memória interna
3. Transferir a informação da memória da unidade de disco para a memória do sistema

Latência -> Tempo que decorre desde o pedido de informação até à disponibilização do primeiro byte de informação

Taxa de transferência de pico (burst) -> Número máximo de bytes transferidos por segundo, após ocorrido o período de latência

Taxa de transferência média -> Numero total de bytes / tempo total (incluindo latência)



Existem várias técnicas de transferência de informação, entre as quais:

1. O CPU faz parte da transferência
   1. E/S programada
      1. O CPU toma a iniciativa – aguarda se necessário, inicia e controla a transferência de informação (POLLING)
   2. E/S por interrupção
      1. O periférico sinaliza o CPU de que está pronto para trocar informação (leitura ou escrita). O CPU inicia e controla a transferência
2. O CPU não faz parte da transferência
   1. E/S por acesso direto à memória
      1. Um dispositivo externo ao CPU (DMA) assegura a transferência de informação diretamente entre a memória e o periférico; O CPU não toma parte da transferência
      2. O CPU apenas configura inicialmente o periférico e o DMA; No final o DMA sinaliza o CPU que a transferência terminou

**E/S Programada**

nChar = 0;

do {

do{

Read “Status Register” of the Keyboard // Aguarda enquanto não houver nenhuma entrada

{ while(Key not pressed)

Read(); // O CPU vai ler a entrada ( atráves) do Data Register

Write(); // O CPU vai escrever na memória a entrada

nChar = nChar +1;

}while(nChar <N)

Portanto, o programa bloqueia até haver uma entrada e só avança quando for premida uma tecla

Durante esse tempo, o CPU não executa outra ação

**E/S por interrupção**

Para evitar o que acontece na E/S Programada (o facto de o programa ficar mais lento pela pausa de que o processador terá que fazer enquanto espera por uma nova entrada) existe o E/S por Interrupção

Para isso, o periférico sinaliza o CPU de que este já pode ler a informação da entrada, evitando a pausa.

Depois de sinalizada, o CPU para de executar o programa que estava a fazer para ler a entrada. A esta rotina se chama de “**Interrupt handler**”

Exemplo: leitura de dados de um periférico  
• CPU envia pedido de informação ao periférico (escrita num registo  
de controlo do periférico)  
• CPU continua a execução do programa (com outras tarefas)  
• Quando tiver informação disponível, o periférico gera um pedido de  
interrupção ao CPU  
• CPU atende a interrupção:  
• Suspende a execução do programa corrente  
• Salta para a rotina de atendimento à interrupção (  
interrupt  
handler) que transfere a informação  
• Retoma a execução do programa suspenso

Não existe qualquer ciclo de espera, pois o periférico gera o pedido quando já está pronto a transmitir

O programa em execução pode ser interrompido a qualquer momento.

A Rotina de Serviço à Interrupção tem que salvaguardar o contexto do programa (Registos Internos) antes de executar qualquer ação

A palavra chave “**interrupt**” distingue a função do tipo RSI de uma função normal

**Exceções :**  A instrução que a gera, não a termina

**Interrupções :**  A unidade de controlo apenas verifica se existe algum pedido de interrupção pendente antes de iniciar o fetch de uma nova instrução

**Secção Critica :**  Uma sequência de instruções em que a ocorrência de uma interrupção interferia de forma indesejada no funcionamento do sistema

**RSI :** Rotina de Serviço à Interrupção

Em termos gerais, o processamento de uma interrupção é efetuado pelo CPU nos seguintes passos:

1. Identificação da fonte de interrupção e obtenção do endereço da RSI
2. Salvaguarda do contexto atual do CPU
3. Desativação das interrupções
4. Carregamento noo PC do endereço da RSI ( salto para a primeirsa instrução da RSI)
5. Execução da instrução do retorno da RSI

**RSI -:**

1. Salvaguarda do contexto do programa que foi interrompido
2. ..Ações da interrupção..
3. Reposição do contexto do programa interrompido
4. Conclusão da RSI com a instrução de retorno (eret)

**Latência da interrupção :**  Define-se como o tempo que decorre entre a ocorrência do evento que desencadeia a interrupção até à execução da primeira execução útil da RSI

**Fiz até slide 22**