Entrada e Saída (E/S ou I/O)

Tráfego de Dados – Possibilidades ("Estratégias de Hardware")

a) Acesso Direto aos Pinos

Típico em microcontroladores de pequeno porte, um comando joga os dados diretamente em pinos específicos do CI. Ex.: MOV P1,#0FFh seta os pinos 1 a 8 de um 8051 para 5V; para jogar apenas o pino 1 para 0V, basta fazer CLR P1.0.

b) Acesso via Barramento

Um barramento *(bus)* é um conjunto de condutores onde determinada informação trafega, ou podemos dizer que é um conjunto de sinais com um propósito específico. Podemos ter barramentos seriais (mais comuns, usualmente mais rápidos) ou paralelos (mais simples). Ambos podem ser ponto-a-ponto (apenas dois elementos conectados) ou multiponto (todos os componentes ligados diretamente ao mesmo ponto).

Nos **barramentos paralelos**, podemos ter o Data Bus (dados), o Address Bus (endereços) e o Control Bus (todos os sinais restantes). Tanto a memória quanto os periféricos utilizam os mesmos barramentos; os acessos devem ser disciplinados via um protocolo que utiliza sinais de controle e endereços para diferenciar operações tanto para chips de memória quanto para periféricos específicos. Decodificadores são utilizados para este fim.

No obsoleto Z80, os comandos LD C,47; OUT (C),255 escrevem o dado 255 no periférico de endereço 47 (*Data*=255, *Addr*=47, os sinais *Write* e *IORequest* estão ativos). Na arquitetura x86 original, da Intel, as instruções MOV CX,378h; MOV AX,41h; OUT (CX),AX seguidas de um "*strobe*" irão escrever uma letra 'A' numa impressora ligada à porta paralela de um PC-XT.

Atualmente os **barramentos seriais** são os mais comuns, tanto para microcontroladores quanto para periféricos de alto desempenho. Neles, os dados, endereços e controle são enviados bit a bit, um de cada vez por uma única linha, de acordo com o protocolo utilizado. Eles superam os paralelos porque nestes, com o aumento das taxas de transferência, temos problemas com:

- *Clock skew:* sinais paralelos sofrem atrasos diferentes (por capacitâncias parasitas, reflexão de sinais, impedância dos condutores), o que causa transições em momentos ligeiramente diferentes para cada condutor. Isso prejudica especialmente a sincronização de clocks.
- *Crosstalk:* dois condutores em paralelo irão gerar interferência eletromagnética um no outro, mesmo havendo cuidados no isolamento.
- Área e isolamento: múltiplos condutores em paralelo ocupam muito espaço na pastilha de silício, na placa de circuito impresso e mesmo em cabos, dificultando o isolamento elétrico.
- Contagem de pinos: cada sinal exige um pino, o que demanda área de placa e encarece os chips, em especial para sistemas embarcados (embora os x86-64 da Intel e AMD possam ter cerca de 4000 pinos usando conectores LGA – *Land Grid Array*).

Note-se que os problemas de reflexão da linha são relacionados à *inclinação* das rampas de clock (e não exatamente à frequência de clock ou dados), que aumenta a potência dos harmônicos do sinal, que são refletidos e deterioram o sinal, caso não se coloque um terminador ao final da linha.

Com isso, os periféricos tendem a ser serializados, tanto a nível de sistema (USB, SATA, PCIe) quanto de placa (padrões I^2C, SPI) e até mesmo dentro de pastilhas com alta integração (System on a chip – SoC), podendo ser **síncronos** ou **assíncronos**.

c) Acesso com Periféricos Mapeados em Memória

Esta é a opção mais comum tanto para sistemas grandes, tais como um PC desktop ou notebook, quanto para processadores RISC. Para cada periférico no sistema é atribuída uma área de memória, de forma que escritas e leituras nestes endereços estão na verdade acionando o periférico ao invés dos pentes de memória.

Exs.: operações *lw* e *sw* em regiões definidas como de periféricos em MIPS são necessárias para qualquer operação de I/O; periféricos modernos no PC, como barramentos PCI e PCIe, são mapeados em memória.

Disciplinas de Transferência ("Estratégias de Software")

a) Polling

Neste caso, o programa rodando no processador tem a obrigação de periodicamente pedir ou escrever o valor no periférico. O instante de acesso é definido pelo programa, portanto, que tem que fazer uma "pesquisa" (*poll*) em todos os periféricos. A implementação é simples: usualmente temos um loop (*superloop* ou "lupão") que acessa sequencialmente, um de cada vez, os periféricos.

As desvantagens são evidentes, contudo: a latência no acesso ao periférico depende do tamanho do ciclo de execução do *software*, que é variável. A execução sequencial do programa torna bastante difícil a programação de certas situações comuns, e é difícil organizar um número grande de periféricos ligados ao mesmo processador.

Um exemplo: para ler um ADC, é preciso gerar um pulso de início de conversão e esperar um tempo fixo ou um sinal de término para só então ler os dados. Um programa que espera pelo término lendo o sinal está fazendo o que se chama de *busy waiting* ou *spinning*, e fica paralisado enquanto espera.

b) Interrupções[1]

Quando o sistema suporta interrupções (ou exceções, há diferentes definições de nomenclatura), o programa principal é pausado temporariamente para que o processador possa lidar com um periférico através de uma *rotina de tratamento de interrupção*[2] (ISR – *Interrupt Service Routine*). O programa principal é retomado no ponto em que parou logo após o fim desta rotina.

Quem define o momento de comunicação, neste caso, é o periférico, que requisita a atenção do processador gerando um sinal para este. Isto libera do processador a carga de ter de verificar um a um todos os dispositivos ligados a ele e naturalmente paraleliza o tratamento de I/O.

Se tivermos um sistema com baixa carga de informações sendo transferidas entre CPU e periféricos, podemos garantir uma baixa latência no atendimento a uma solicitação de I/O por parte de um periférico. Conflitos são resolvidos com a fixação de uma hierarquia de prioridades de atendimento.

***c) DMA* – Direct Memory Access**

É evidente que a transferência de uma massa muito grande de dados usando o método de interrupções iria praticamente paralisar o processador. Para evitar isso, em sistemas mais complexos é utilizado um CI controlador de acesso direto à memória, que irá fazer a transferência de dados diretamente entre o periférico e a memória, em ambos os sentidos.

A operação de DMA é sempre iniciada pelo processador, que irá dar instruções ao controlador de DMA. Nos momentos apropriados, o controlador irá *retirar o controle do barramento da CPU*, em alguns casos efetivamente paralisando o processador, e estabelecendo um canal de movimentação de dados direto entre o dispositivo e a RAM principal.

Note-se que a presença de **memória cache** no sistema faz com que o processador possa continuar executando normalmente o programa enquanto o DMA ocorre (desde que não ocorram falhas de cache, claro), mas cria um outro problema: os dados do periférico devem estar mapeados na RAM principal ou na cache?

É comum fazer a transferência de dados para a RAM principal (mapeamento por endereços reais), invalidando quaisquer blocos de cache que contenham cópias destes dados, mas outras alternativas também são possíveis (cf. p. 452 do livro).

1 Material de interrupções no MIPS pode ser visto no apêndice A.7 (no CD que acompanha a 3a edição do livro)

2 Há várias formas de identificar a fonte da interrupção e desviar para o endereço de programa adequado. A *vetorização* das interrupções significa que há uma tabela de endereços mapeando o dispositivo originador e o endereço da respectiva rotina de tratamento.