Entrada e Saída (E/S ou I/O) – Capítulo 8

Acesso Físico aos Periféricos

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.

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 com o aumento das taxas de transferência, temos problemas com:

- *Clock skew:* sinais paralelos sofrem atrasos diferentes (capacitâncias parasitas, reflexão de sinais, impedância dos condutores), o que causa transições em momentos ligeiramente diferentes para cada condutor.
- *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 Core i7 da Intel já estejam com 1155 pinos e vão a 2011, na arquitetura Sandy Bridge, 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 no nível de sistema (USB, SATA, PCIe) quanto no 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.

Possibilidades para o Programador

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 é retomado após o fim da 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.

Questionário – Estudo Dirigido

Não é necessário entregar. Tirem as dúvidas comigo.

1. Por que o acesso direto aos pinos é mais adequado no caso de microcontroladores de pequeno porte? No caso de sistemas mais complexos como, digamos, um telefone celular, o microcontrolador ainda deve usar esta estratégia de *hardware*?
2. Considere dois barramentos: um barramento que é paralelo de 8 bits sincronizado por um clock máximo de 10kHz e outro barramento que é serial assíncrono com palavras de 8 bits antecedidas por um *start bit* e concluídas por um *stop bit*, com cada bit transmitido em não menos de 1ms. Qual a taxa de transferência máxima dos barramentos, em kbytes/s? Suponha que o tempo usado para a rampa de subida e para a rampa de descida, somados, equivalha a 20% do período de clock ou bit. Qual dos dois barramentos exige maior cuidado com reflexão de sinais na linha?
3. Os PCs modernos utilizam tanto o acesso por barramento (por retrocompatibilidade) quanto o mapeamento em memória. Isso deve fazer alguma diferença para o programador que trabalha em alto nível? Por quê?
4. Quais as vantagens e desvantagens de se usar mapeamento em memória para o projetista de microprocessadores?
5. Exemplifique um caso em que o *polling* é a melhor estratégia de software a adotar.
6. Podemos dizer que o *polling* é síncrono e as interrupções são assíncronas em relação ao programa principal. Pensando nisso, se tivermos um caso em que o sinal de um dispositivo de I/O fica ativo por um tempo muito pequeno, qual dos dois é mais indicado e por quê?
7. Um aluno desocupado está tentando emular um HD IBM RAMAC 350 de 1956, que possui tempos de leitura de 1ms/byte. É vantajoso usar DMA, considerando que o custo adicional é zero pelo controlador DMA já estar presente no circuito?
8. [Opcional] Podemos mapear os periféricos em endereços da cache ao invés da RAM principal (ou seja, usando endereços virtuais ao invés de físicos). Que complicações você imagina que isso pode trazer?
9. Um processador AMD64 rodando um Windows 32 bits consegue enxergar apenas 3GB dos 4GB de RAM disponível, pois o 1GB superior (próximo ao fim da memória) é perdido para o mapeamento dos periféricos. Rodando a 64 bits, ele consegue usar os 4GB, remapeando periféricos e memória para outros endereços. Pergunta: quantos bits do *address bus* são utilizados, aparentemente, para descobrir se o endereço acessado é memória ou periférico?
10. Considere os modelos simplificados de processadores e I/O descritos a seguir. Escolha as estratégias de *software* e *hardware* mais adequadas para cada um dos casos.
    Assuma interrupções sem prioridades, com tempo de processamento para atendimento sendo desprezível e todos os periféricos com características idênticas. O programa principal exige 30% do tempo de processamento da máquina.
    a) 8 periféricos exigindo banda de 10kB/s com latência máxima aceitável de 3ms de leitura/escrita (ou seja: entre o momento “ideal” da operação e o real, não podemos estourar 3ms), clock do processador a 10MHz.
    b) 3 periféricos de banda 100kB/s, com latência que pode ser ignorada e clock a 50MHz.
    c) 40 periféricos exigindo banda de 10B/s (simples sensores/atuadores) com latência máxima de 1ms cada e clock de 10kHz.
    d) 3 periféricos com banda negligenciável, mas a latência de qualquer operação é, no mínimo, 3s (ou seja, entre a requisição e a execução, demoramos sempre mais de 3s), clock acima de 1MHz.