Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia de Sistemas Eletrônicos - PSI

PSI-2553- Projeto de Sistemas Integrados

Experiência 3: Utilizando Interrupção no Plasma (Teoria)

M.S. / W.J.C / M.A.R.J (17)

1 Objetivos

Esta experiência visa a exploração dos conceitos de comunicação entre processadores e periféricos em sistemas digitais, particularmente por interrupção.

2 Parte Teórica

2.1 Tipos de Interrupção

Como visto nas aulas de teoria, a interrupção é uma forma tradicional de o processador atender as necessidades de comunicação de dados com um periférico. A interrupção é necessária quando o processador é mestre na comunicação e o periférico atua como escravo. Neste caso, somente o mestre pode iniciar a comunicação, portanto a interrupção passa a ser uma forma de o periférico avisar que necessita de algum serviço.

A interrupção pode ser de dois tipos, fixa ou vetorizada:

1) Na interrupção fixa, o processador ao atender a interrupção, sai da sua execução normal de programa e o contador de programa, PC, salta para um endereço fixo de memória, onde reside a sub-rotina de serviço de interrupção, ISR. Finalizada a ISR, o PC retoma o endereço de instrução anterior à interrupção e o processador volta a executar o programa principal. Este tipo de interrupção torna o hardware simples, porém é bastante limitado. Em geral é adequado para sistemas embutidos, pois na maioria dos casos, os periféricos são pré-definidos e pouca, ou nenhuma atualização, é permitida.
2) A interrupção vetorizada prevê que o periférico passe ao processador o endereço da ISR, o qual pode ser modificado. Esta forma é adequada para sistemas onde novos ou uma diversidade de periféricos podem ser conectados ao processador. Desta forma, cada periférico terá uma ISR diferente, localizada em posição de memória específica.

Para mais detalhes, consulte o livro-texto da disciplina.

2.2 Interrupção no Processador Plasma

O processador Plasma utiliza a abordagem de interrupção fixa. Como, então, tratar diversos periféricos, que é um caso normal de sistemas microprocessados? Para resolver esta dificuldade, utiliza-se de alguns registradores especiais. Nas próximas seções, iremos ver como o hardware do Plasma foi concebido para receber a interrupção de vários periféricos e como ele interage com o software.

2.2.1 Endereços de Registradores de Periféricos e Interrupção

O mapa de espaço de memória é definido no arquivo topo (plasma.vhd) com endereçamento de 32 bits. Esta definição inclui as implementações internas de periféricos e memórias que podem usar espaço menor, ou seja, apenas parte dos bits de endereçamento. Por exemplo, apesar de não haver restrições por parte de outros submódulos do Plasma, o bloco de memória RAM (ram.vhd) é de 8KB, com endereçamento de 13 bits (os 13 bits menos significativos dos 32 bits de endereçamento).

Em relação aos periféricos, o Plasma original já estabelece no seu espaço de endereçamento um conjunto de registradores associados aos periféricos, cf. indicado na Figura 1 e na Tabela 1. Observe pela figura que a interrupção é gerada quando o resultado da operação AND entre os conteúdos dos registradores IRQ_MASK e IRQ_STATUS apresenta pelo menos 1 bit igual a '1', significando que, pelo menos um periférico (dentre os habilitados atualmente pelo IRQ_MASK) está solicitando a interrupção.

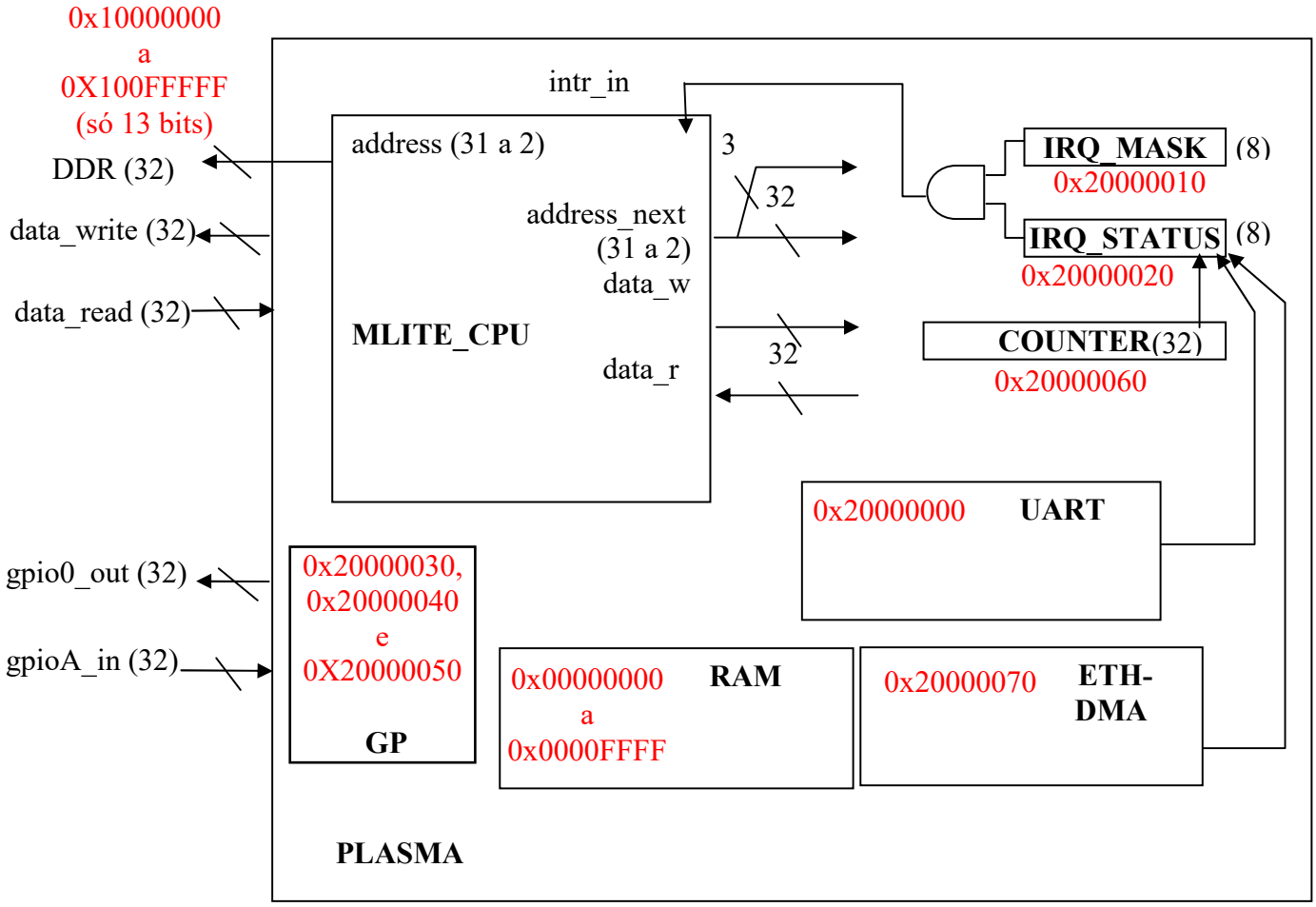

Figura 1. O Plasma e o endereçamento de seus periféricos e memória

Tabela 1. Endereçamento de Registradores dos Periféricos

Endereço	Registrador de Periférico(8/32 bits)
0x20000000	Uart
0x20000010	IRQ Mask
0x20000020	IRQ Status
0x20000030	GPIO Out (Set bits)
0x20000040	GPIO Out (Clear bits)
0x20000050	GPIO In
0x20000060	Counter
0x20000070	Ethernet transmit count

Os endereços acima estão embutidos no código VHDL do Plasma, i.e., estão definidos por hardware, e para alterações ou inclusões de novos registradores/periféricos, mudanças no código são necessárias. Os registradores da Figura 1que são associados a periféricos funcionam como

posições de memória para onde dados relativos a estes periféricos são enviados pelo processador ou retirados por este.

Os registradores especiais IRQ_Status e IRQ_Mask são, na prática, de 8 bits; a rigor, o registrador pode ter 32 bits, mas só os 8 menos significativos são úteis. Cada bit está associado a algum periférico que solicitou a interrupção e às suas diferentes tarefas, como mostrado na Tabela 2. Por exemplo, ao se endereçar a UART (0x20000000) par comunicação de dados, pode ocorrer ou uma leitura de dados pelo processador (com a ativação do bit 0 dos registradores IRQ_Status ou (_Mask)) ou uma escrita (com a ativação do bit 1). Os endereços e posições de bit são também pré-definidos em hardware.

Tabela 2. IRQ bits

Posição do Bit	Registrador/Periférico
0	UartDataAvailable
1	^UartWriteBusy
2	^Counte
3	Counter
4	EthernetReceive
5	EthernetSendDone
6	^GPIO
7	GPIO

Detalhando os registradores IRQ_Status e IRQ_Mask:

- IRQ Mask: o registrador que contém o mapeamento das requisições permitidas de cada periférico ou registrador de periférico. Estas condições são definidas pelo usuário, via software, isto é, o código de programa do usuário se encarrega de escrever este mapa no registrador IRQ_Mask, de endereço 0x20000010 (nos 8 bits menos significativos), como ilustrado na Figura 2. Desta forma, de acordo com a porção do programa, certos periféricos poderão estar impedidos de realizarem interrupção.

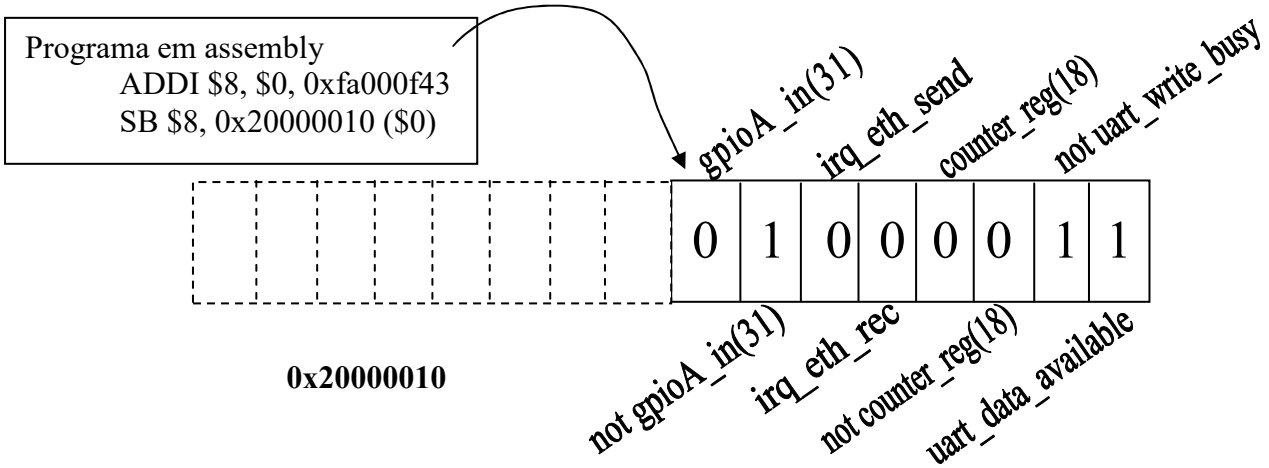

Figura 2. Programando o registrador IRQ_Mask- os bits ‘1’indicam habilitação

- IRQ Status: o registrador que contém o estado atual das requisições em hardware dos periféricos ou registradores de periférico. Importante: este registrador é alterado via hardware por requisições de periféricos; pode e deve ser lido através das instruções assembly (software) pelo seu endereço. O periférico que solicita uma interrupção durante a operação do circuito escreve no registrador, em posição pré-definida em hardware. A Figura 3 ilustra como o registrador IRQ_Status, de endereço 0x20000020 (os 8 bits menos significativos), está conectado aos periféricos. Neste pequeno exemplo, o registrador apresenta a situação em que apenas o periférico UART faz a solicitação, para escrita apenas (pelo segundo bit, UartWriteBusy, no caso da Tabela 2). Uma ISR correspondente a estes periférico/tarefa tratará deste caso.

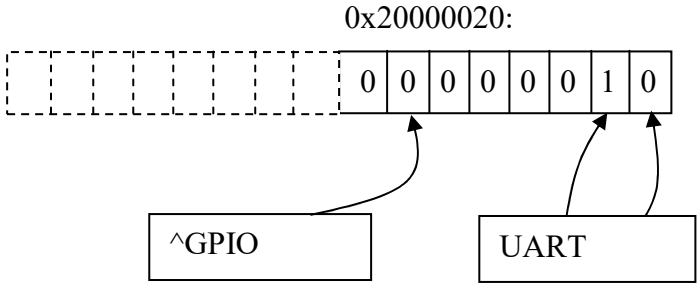

Figura 3. O registrador IRQ_Status

2.2.2 Registradores do Coprocessador

No MIPS um módulo coprocessador é especialmente especificado para lidar com as tarefas de interrupção (e outras exceções). O coprocessador contém uma série de registradores próprios, que não devem ser confundidos com os outros 32 do processador, de propósito geral. Estes registradores podem ser descritos por seus nomes, como $status ou $epc. Eles são acessados exclusivamente através das instruções MFC0 e MTC0, abaixo exemplificados:

- MTC0 $epc, $s1 (escreve no registrador epc do coprocessador o valor de $s1 do processador)

- MFC0 $s1, $status (armazena em $s1 do processador o valor do registrador de status do coprocessador)

As características destes dois registradores encontrados na atual distribuição do Plasma são:

- EPC ($epc): é usado durante a interrupção por hardware[1], quando recebe o valor corrente de PC +4. Neste mesmo momento, o registrador o PC recebe o endereço **fixo** 0x3c, definido por hardware (no reg_bank.vhd do códigos VDL do Plasma).
- STATUS ($status): é um registrador para supervisão do sistema. Nesta versão do hardware do Plasma, apenas o seu bit menos significativo tem significado de habilitação/desabilitação da interrupção (observe que esta habilitação ou desabilitação é geral, afetando a interrupção por parte de todos os periféricos. Quando status(0)=´0´, a interrupção está desabilitada ; quando status(0)=´1´, ocorre a habilitação.

[1] Se fosse uma interrupção por software, poder-se-ia utilizar a instrução jal (jump and link) para o armazenamento do valor PC+4 no registrador $ra.

2.2.3 Interrupção por Hardware e o seu tratamento no Software

Três aspectos principais devem ser tratados por software quando realizamos interrupção: o ajuste inicial dos registradores, o procedimento para tratamento de interrupção e a estrutura da rotina interrupção em si. Para dar suporte a estas tarefas um arquivo fornecido com a distribuição do Plasma deve ser utilizado: o boot.c. Este arquivo deve ser compilado junto ao arquivo do usuário que contém o programa principal (main) e ao programa com as tarefas de interrupção.

O programa boot.c apresenta as seguintes funções (detalhes serão vistos na parte prática):

- inicializar os registradores- todos os 32 registradores são inicializados com valores *default*, como, por exemplo, valores de endereço de memória onde é o início das áreas de dados de uso geral ($GP=$28) ou do *stack pointer* (apontador de pilha) ($SP=$29). O compilador cruzado MIPS realiza esta tarefa, seguindo um mapa de endereçamento pré-definido para a memória. Este mapa é definido tanto no hardware como tabelado no software..
- iniciar o programa principal: saltar para a posição de memória onde reside a primeira instrução do programa principal e iniciar a sua execução.
- Inserir blocos de instruções/rotinas de uso geral: sem dúvida, o mais importante é a instrução que se inicia no endereço 0x3c, endereço da **interrupção fixa**. Neste local encontra-se o código do ISR principal, um conjunto de instruções que faz o seguinte:
    - guarda todos os valores dos registradores na pilha (em posições de memória controlados pelo valor de $sp)
    - chama a rotina de interrupção, efetivamente preparado pelo do usuário, denominado OS_InterruptServiceRoutine, para tratar das tarefas do(s) periférico(s) que solictou(aram) a interrupção.
    - após o retorno do OS_InterruptServiceRoutine, retorna todos os valores da pilha para os registradores
    - sai do ISR principal e retorna ao programa principal

Um dos aspectos fundamentais no funcionamento da interrupção é o código da ISR principal (que consiste de algumas poucas linhas de código *assembly*, começando no endereço 0x3c) que está na Figura 4. Neste trecho de código do boot.asm há uma chamada à sub-rotina OS_InterruptServiceRoutine.

```
lui   $a2,  0x2000
lw    $a0,  0x20($6)  #IRQ_STATUS
lw    $a2,  0x10($2)  #IRQ_MASK
and   $a0,  $a0, $a2
jal   OS_InterruptServiceRoutine
```

Figura 4. Trecho de código de boot.asm

Primeiramente, vamos nos focar na instrução “jal” (jump and link); a utilidade dos outros ficará aparente mais tarde. Com a compilação, será criado um jump para o endereço físico relativo ao ponto onde for alocada a sub-rotina de interrupção do usuário criar, de nome OS_InterruptServiceRoutine. Há dois casos:

- Se o código de OS_InterruptServiceRoutine estiver também em *assembly*, basta seguir com o rótulo (label).
- Se for em programa C, cria-se um procedimento como na Figura 5. O endereço do PC antes do salto será armazenado em $ra.

```
//Endereços de hardware
#define UART              0x20000000
//Bit do IRQ
#define IRQ_UART_READ_AVAILABLE  0x01

#define MemoryRead(A) (*(volatile unsigned int*)(A))
#define MemoryWrite(A,V) *(volatile unsigned int*)(A)=(V)

void OS_InterruptServiceRoutine(unsigned int status)
{
    char c = '0';
    if( status & IRQ_UART_READ_AVAILABLE)
    {
        c = (char) MemoryRead(UART);
        MemoryWrite(UART, c);
    }
}
int main()
{
    ….
    for(;;)
        ;
}
```

Figura 5. Sub-rotina de interrupção e programa principal

Observe que este salto para o OS_InterruptServiceRoutine é genérico e até então não se sabe qual é o periférico que solicitou a interrupção. Pelo exemplo da Figura 5, percebe-se que variáveis podem ser definidas com o endereço de registradores relacionados aos periféricos (caso do UART). O que a sub-rotina de interrupção do usuário faz, de início, é exatamente verificar qual o periférico que solicitou a interrupção. A forma encontrada é pela comparação do valor de status com a variável IRQ_UART_READ_AVAILABLE (= 0x01, veja os #define do programa). Caso status também tiver o mesmo valor, então o periférico é o UART, e a tarefa prevista da sub-rotina é executada (é o trecho de código entre as chaves mais internas).

Entretanto, qual é o valor de status?

Se observarmos o código do boot.asm na figura 4, antes da chamada da sub-rotina de interrupção, o registrador $a0 adquire valor do registrador IRQ_Status, enquanto o $2, o do registrador IRQ_Mask. É feito um AND entre eles que é colocado em $a0 (argumento de sub-rotina), significando que o valor de *status* será $0x01 (pelo exemplo) se o registrador IRQ_Status indicar que a UART está requisitando uma interrupção.

2.3 Novo Periférico com Interrupção

Introduzimos aqui o módulo denominado Perif_Exp3, que servirá de exemplo, nesta experiência, de um componente periférico do sistema Plasma que requisita interrupção. O esquema básico do módulo é apresentado na Figura 6. Procura-se replicar o periférico utilizado na teoria para

a ilustração do conceito de interrupção (Fig.6.11 do livro-texto). Pode-se observar as seguintes características.

1) O módulo funciona realmente como I/O, tendo interface com o meio externo através dos sinais:

- *data_in_ext*: entrada de dado externo de 32 bits;
- *data_out_ext*: saída de dado externo de 32 bits;
- *ext*: flag do meio externo, indicando ao periférico que há dado externo disponível através do sinal *data_in_ext*.

2) Tão logo o módulo recebe o dado acima, faz a requisição de interrupção para a CPU:

- Ao capturar o dado externo através de *data_in_ext*, o dado é armazenado no registrador Perif_Exp3 (de endereço 0x20000070). Observar que usamos o endereço originalmente dedicado ao ETH-DMA, o qual será desabilitado;
- O sinal *irq_perif_exp3* é ativado solicitando a interrupção ao processador.

3) Na rotina OS_InterruptServiceRoutine, o seguinte conjunto de tarefas é definido:

- Ler o dado do periférico para um registrador do Plasma e, deste, fazer um *store* para uma posição da memória interna RAM; observar que a primeira operação, com o periférico, envolve a ativação do sinal *r_e* (read enable), pois é equivalente a ler um dado de algum endereço qualquer da memória;
- Alterar e, depois, fazer um *load* o dado de posição da memória interna RAM para um registrador do Plasma e, deste, para o periférico; observar que esta última operação, com o periférico, ativará o sinal *w_e* (write_enable), pois é equivalente a escrever um dado do registrador para algum endereço qualquer da memória.

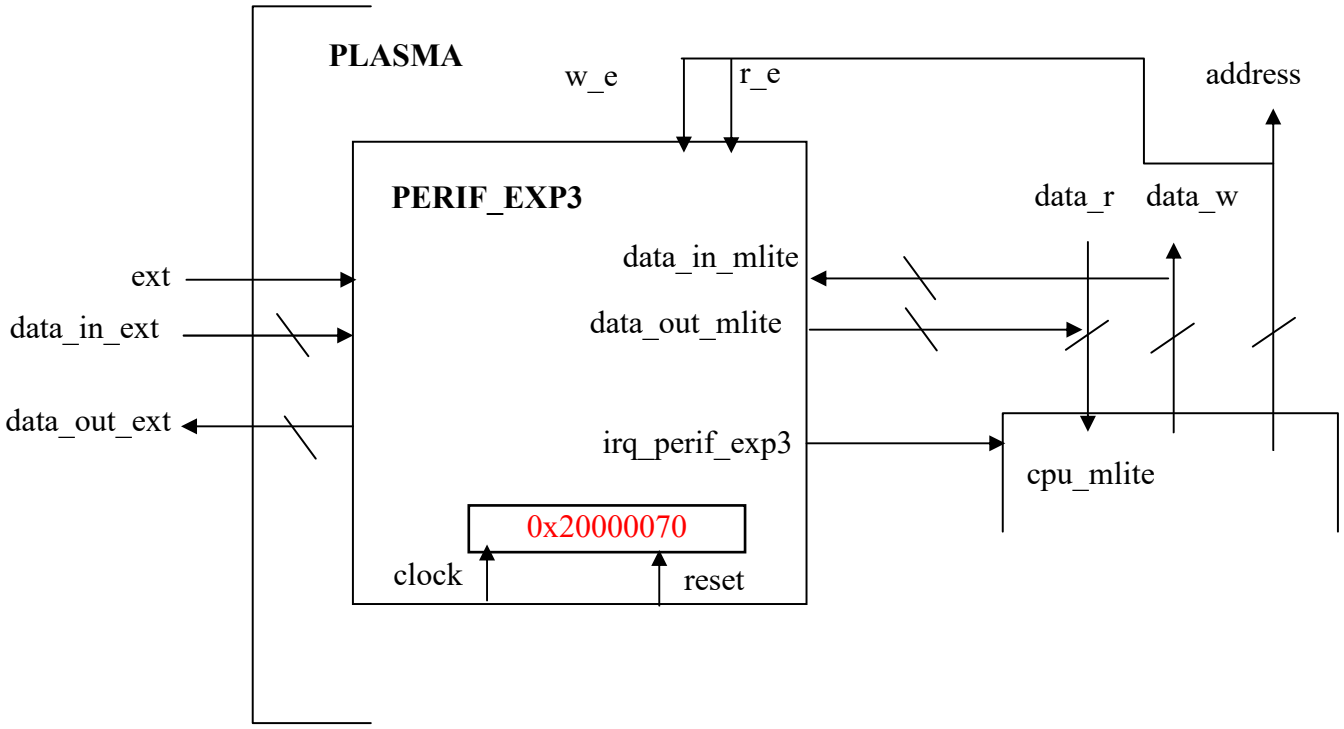

Figura 6. Esquema com o Módulo Perif_Exp3