Instituto Superior Técnico

Arquitectura de Computadores

Dicas e Truques do Assembly do P3

Prof. Renato Nunes

Versão 2.0 03/03/2008

1. Introdução

Este documento contém vários pedaços de código, instruções assembly individuais e notas diversas, chamando a atenção para aspectos relevantes da programação assembly do processador P3, inicializações que não devem ser esquecidas e aspectos específicos associados ao uso de interrupções e de periféricos do P3.

Este documento complementa a demais documentação disponível sobre o simulador do processador P3 e sobre os trabalhos de laboratório. São bem vindas todas as sugestões com vista à sua melhoria.

2. Inicialização do apontador para a pilha (SP)

Um programa que use subrotinas, que use interrupções ou use a pilha (stack) directamente, necessita de inicializar o apontador para o topo da pilha (SP - *Stack Pointer*). SP deve apontar para uma zona de memória livre. Recordar que a pilha "anda para trás", i.e., a colocação de dados na pilha **decrementa** o valor de SP, enquanto o retirar de dados da pilha aumenta o valor de SP. O SP deve por isso ser inicializado com um valor elevado, afastado da zona memória usada pelo programa e pelos dados. Um bom valor é a posição imediatamente antes da Tabela de Interrupções (localizada em FE00h).

```
TopoPilha  EQU  FDFFh

MOV R1,TopoPilha
MOV SP,R1
```

3. Interrupções

O uso de interrupções obriga a que sejam realizadas as seguintes acções (que serão detalhadas adiante):

1. Preenchimento da Tabela de Interrupções
2. Definição da Máscara de Interrupções
3. Activar o *Enable* das interrupções

Nota: Não esquecer de inicializar o SP.

3.1 Preenchimento da Tabela de Interrupções

A Tabela de Interrupções corresponde a uma zona de memória onde devem ser colocados os endereços das subrotinas que tratam as várias interrupções. Existe uma posição de memória por cada interrupção possível (0 a 255). A tabela tem o seu início no endereço **FE00**h e ocupa 256 palavras.

3.1.1 Uso de instruções assembly

A tabela de interrupções pode ser inicializada usando instruções assembly. Apresenta-se em seguida um exemplo relativo à interrupção 15 (interrupção do temporizador).

```
INT15  EQU  FE0Fh     ; FE0Fh = FE00h + Fh   (Fh = 15)
                      ; Notar que nada obriga a usar o nome INT15
                      ;  mas é mais sugestivo...


MOV R1, MyIntSub  ;R1 = endereço da rotina que trata a interrupção
MOV M[INT15],R1   ;Guarda o endereço na posição adequada da Tabela
                   de Interrupções

 ...
 ...

MyIntSub: ...     ; rotina de tratamento da interrupção
 ...
 ...
 RTI              ; RTI e não RET...
```

3.1.2 Uso de directivas assembly (pseudo-instruções)

Uma outra alternativa, que poupa a escrita de código e evita o consequente gasto de memória com essas instruções, consiste em recorrer a directivas do assembler (pseudo-instruções):

```
ORIG  FE0Fh           ; FE0Fh = FE00h + Fh   (Fh = 15)
INT15 WORD MyIntSub


MyIntSub: ...
 ...
 ...
 RTI
```

As directivas indicadas efectuam a declaração de uma variável na posição desejada da memória e inicializa essa posição com o endereço da rotina de tratamento de interrupção (no caso do exemplo esse endereço está associado à etiqueta MyIntSub).

Nota: A conclusão de uma subrotina de tratamento de interrupção tem de ocorrer usando a instrução **RTI**. Lembrar que, quando ocorre uma interrupção, é colocado na pilha o registo de estado (RE) e o endereço de retorno (PC). A instrução RTI repõe o valor do registo de estado e o endereço de retorno. Este comportamento é diferente do que se verifica quando é chamada um subrotina comum, que deve terminar por RET.

3.2 Definição da Máscara de Interrupções

É possível, para as interrupções 0 a 15, controlar individualmente se são aceites ou ignoradas. Isso é possível preenchendo adequadamente a Máscara de Interrupções que corresponde à palavra de memória com o endereço **FFFA**h.
Inicialmente esta máscara encontra-se a 0, impedindo que qualquer das interrupções 0 a 15 sejam aceites. Para aceitar uma dada interrupção é necessário colocar o respectivo bit a 1 (bit 0 a bit 15). Apresentam-se em seguida alguns exemplos:

```
InterrupMask  EQU  FFFAh    ; endereço da Máscara de Interrupções

Int2_mask  EQU  0004h       ; 0000 0000 0000 0100 b
Int15_mask EQU  8000h       ; 1000 0000 0000 0000 b
MyInt_mask EQU  8005h       ; 1000 0000 0000 0101 b
                            ; esta última máscara permite aceitar
                            ; as interrupções 0, 2 e 15

...

MOV R1,Int2_mask
MOV M[InterruptMask],R1     ; Permite a interrupção 2

...

MOV R1,MyInt_mask
MOV M[InterruptMask],R1     ; Permite as interrupções 0, 2 e 15
```

3.3 Activar o *Enable* das interrupções

Existe um mecanismo global de aceitação ou rejeição de todas as interrupções. Isso é controlado pelo bit de estado E (Enable interrupt). Este bit contém inicialmente o valor 0 impedindo que o P3 aceite qualquer interrupção. Para colocar esse bit a 1 e permitir que o P3 possa tratar interrupções, o programador tem de usar explicitamente a instrução **ENI** (*enable interrupt*).

```
...

ENI  ; a partir de agora serão aceites interrupções

...
```

3.4 Informação adicional sobre as interrupções

Em certas circunstâncias pode interessar inibir as interrupções. Isso pode ser feito evocando a instrução DSI (*disable interrupt*), que coloca a 0 o bit de estado E.

Notar que se ocorrer uma interrupção estando o bit E a 0, a respectiva subrotina de tratamento da interrupção não será chamada, mas o pedido não se perde (fica registado internamente na flag I). Assim, nesta situação, assim que for executada a instrução ENI, será imediatamente tratada a interrupção pendente.

Relembra-se também que o bit E é colocado a 0 quando ocorre a chamada de uma subrotina de tratamento de interrupção. Deste modo garante-se que essa subrotina será executada sem interrupções. Se durante o seu tempo de execução ocorrer algum evento que gere nova interrupção, isso será memorizado mas a evocação da respectiva subrotina de tratamento de interrupção só ocorrerá após a sua conclusão, ou seja, após a execução da instrução RTI.

A razão para o comportamento descrito deve-se a que a evocação de uma interrupção coloca o registo de estado a 0 (RE = 0) e, consequentemente, o bit E ficará a 0. No entanto, o valor anterior do RE foi salvaguardado na pilha (e tinha o bit E a 1). Assim, a execução de RTI, que recupera o valor que estava na pilha, repõe o bit E a 1.

Caso se deseje, pode-se usar a instrução ENI no código da subrotina de tratamento de interrupção, permitindo assim que ela possa ser interrompida. No entanto, esta possibilidade deve ser usada com cuidado para evitar comportamentos indesejados ou indevidos.

Recorda-se que as interrupções 0 a 15 estão sujeitas a um controlo adicional, através da máscara de interrupções (ver ponto 3.2). As interrupções 16 a 255 estão apenas sujeitas ao controlo do bit de estado E.

É possível evocar qualquer interrupção explicitamente usando a instrução INT n (por exemplo, INT 122). Uma vez que se trata de uma chamada explícita da interrupção (através de uma instrução colocada propositadamente no código pelo programador), o bit E não tem efeito. Para o caso das interrupções 0 a 15 (INT 0 a INT 15), a máscara de interrupções também não tem efeito.

Relembra-se, a concluir, que as interrupções são usadas normalmente para tratar eventos detectados ao nível do hardware (por exemplo, activação de um pino do processador, temporizador que expirou, foi premida uma tecla). O temporizador está associado à interrupção **15**, enquanto as teclas (botões de pressão) estão associadas às interrupções 0 a 14 (existem 15 teclas).

Nota: Os interruptores de alavanca **não** geram interrupções. Os interruptores de alavanca têm de ser lidos explicitamente pelo programa.

O uso de interrupções para tratar eventos que não se sabe quando vão ocorrer é muito mais eficiente e mais rápido do que o método alternativo (designado de *polling*) em que, periodicamente, é necessário executar instruções (desperdiçando capacidade de processamento) para verificar explicitamente a ocorrência de determinado evento.

Caso, as interrupções estejam *enabled*, o processador desencadeia automaticamente a evocação da subrotina que irá tratar a ocorrência (no ponto 3.1 foi explicado como se define qual a subrotina que trata uma dada interrupção em particular). A chamada dessa subrotina verifica-se assim que o evento ocorre, mas sem interromper a

instrução em curso. A verificação se ocorreu um pedido de interrupção é feita após o fim de uma instrução e antes do início da próxima.

4. Como utilizar o temporizador do P3

O P3 possui um temporizador (*timer*) para permitir contabilizar de forma precisa a passagem do tempo. Na prática corresponde a um contador que é decrementado a intervalos de tempo regulares (100ms).

O uso do temporizador é controlado por duas posições de memória. Uma, com o endereço **FFF6**h, corresponde ao temporizador propriamente dito, contendo o número de unidades de tempo a decrementar. A outra, com o endereço **FFF7**h permite parar o funcionamento do contador (= 0) ou permite que vá decrementando (= 1). Inicialmente a posição de memória FFF7h está a 0 sendo necessário escrever lá o valor 1 para que a contagem tenha início. Notar também que, quando o temporizador atinge o valor 0 (fim de contagem), a posição de memória FFF7h é automaticamente colocada a 0, parando a contagem.

Para iniciar nova contagem será pois necessário colocar novo valor em FFF6h e colocar 1 em FFF7h.

O temporizador está associado à interrupção 15, a qual é activada quando o contador atinge o valor 0 (lembrar que a máscara de interrupções necessita estar devidamente programada - ver ponto 3.2, e o bit de estado E necessita estar a 1 - ver ponto 3.3).

5. Exemplo de utilização do temporizador do P3

Apresenta-se em seguida um exemplo completo da utilização do temporizador do P3 que está associado à interrupção 15.

```
TopoPilha  EQU  FDFFh       ; endereço do topo da pilha
InterrupMask  EQU  FFFAh    ; endereço da Máscara de Interrupções
TimerValue   EQU FFF6h      ; endereço do Temporizador
TimerControl EQU FFF7h      , endereço do controlo do temporizador

Int15_mask EQU  8000h       ; 1000 0000 0000 0000 b

TimeLong  EQU  0050h
EnableTimer EQU 0001h


ORIG  FE0Fh           ; FE0Fh = FE00h + Fh   (Fh = 15)
INT15 WORD TimerSub   ; Preenchimento da posição 15 da Tabela de
                      ;  Interrupções

ORIG  0000h
MOV R1,TopoPilha
MOV SP,R1             ; inicializa SP

MOV R1,Int15_mask
MOV M[InterruptMask],R1     ; Permite a interrupção 15 (timer)

MOV R1,TimeLong
MOV M[TimerValue],R1        ; definir valor de contagem do timer
MOV R1,EnableTimer
MOV M[TimerControl],R1      ; inicia contagem
```

```
   ENI               ; aceita interrupções

    ...
    ...


   ; esta subrotina é chamada quando o temporizador chegou ao fim (0)
   ; vai ser necessário definir novo valor de contagem e permitir
   ; que essa contagem comece ( M[FFF7h] = 1 )
TimerSub: PUSH R1
   MOV R1,TimeLong
   MOV M[TimerValue],R1        ; definir valor de contagem do timer
   MOV R1,EnableTimer
   MOV M[TimerControl],R1      ; inicia contagem

    ...
                    ; operações adicionais …
    ...

   POP R1
   RTI


   ...
```

6. Teste de bits

Em várias circunstâncias torna-se necessário verificar o estado de um certo bit. Um exemplo para esta necessidade ocorre quando se lêem os interruptores de alavanca (cujo estado está disponível na posição de memória **FFF9**h).

6.1 Teste de um bit individual

Uma forma possível para verificar o estado de um bit individual recorre ao uso de uma máscara e à realização da operação AND. Exemplo:

```
   Alavancas EQU FFF9h

   Alavanca2 EQU 0004h   ; 0000 0000 0000 0100 b


   MOV R1,M[Alavancas]
   AND R1,Alavanca2
   BR.Z Alavanca2_zero   ; salta se alavanca 2 estava a 0
   ; passa à instrução seguinte se a alavanca 2 estava a 1
```

Pode ser usada a instrução TEST em substituição do AND (`TEST R1,Alavanca2`). A diferença reside apenas no facto do AND afectar R1 e o TEST não.

6.2 Teste sistemático de bits

Existem situações em que se pretendem analisar todos os bits (ou vários bits) de uma palavra. Neste caso pode-se recorrer a um ciclo:

```
   Alavancas EQU FFF9h


   MOV R2,8          ; R2: número de bits a analisar

   MOV R1,M[Alavancas]   ; R1: estado dos interruptores de alavanca

Loop: RORC           ; o 1.bit a ser analisado é o menos significativo
                     ; notar que o bit a testar foi colocado na Carry
   BR.C Bit_a_um
   ; bit a zero
   ...
   ...

Bit_a_um: ...
   ...
   ...

   DEC R1
   BR.NZ Loop        ; continua ciclo

   ; fim do ciclo (foram analisados 8 bits…)
```

6.3 Como verificar se um número é par ou ímpar

Basta testar o bit menos significativo do número: se for 0 é par; se for 1 é ímpar.
Nos exemplos que se seguem assume-se que o número está em **R1**:

Exemplo 1: (o conteúdo de R1 é afectado)

```
   SHR R1,1          ; Altera R1
   BC Impar
   ; é Par
   ...
   ...

Impar: ...
   ...
   ...
```

Exemplo 2: (o conteúdo de R1 <u>não</u> é afectado)

```
   ROR R1,1
   ROL R1,1          ; R1 não é alterado
   BC Impar
   ; é Par
   ...
   ...

Impar: ...
   ...
   ...
```

7. Manipulação da pilha (stack)

Exemplos de instruções que manipulam o conteúdo da pilha.

```
PUSH R1           ;...

PUSH M[end]       ;Coloca no topo da pilha o conteúdo da posição
                   de memória com o endereço end

POP R0            ;Elimina valor do topo da pilha sem afectar
                   nenhum registo

MOV R1,M[SP+2]    ;Lê para R1 valor guardado na pilha. O valor é
                   lido do endereço SP+2
```

8. Deslocamentos e cálculo aritmético

Num deslocamento (shift) de bits à direita, cada bit passa a ocupar uma posição cujo valor é metade do seu valor original. Assim, deslocar um número à direita equivale a dividir esse número por 2.

Exemplos (deslocamento de 1 bit à direita):

```
00100  (4)  ->  00010 (2)
01010 (10)  ->  00101 (5)
```

Notar que se trata de uma divisão inteira, perdendo-se a parte fraccionária do resultado. Exemplo:

```
00101  (5)  ->  00010  (2)   (5/2=2,5 ->2)
```

Se o deslocamento à direita não envolver só um bit mas `n` bits, isso equivale a dividir o número por 2^n (exemplo: deslocar à direita 3 bits equivale a dividir por 8).

O que foi afirmado para os deslocamentos à direita também se verifica para os deslocamentos à esquerda mas em que a operação envolvida é agora a multiplicação.

Exemplos (deslocamento de 1 bit à esquerda):

```
00011 (3)  ->  00110  (6)
00101 (5)  ->  01010 (10)
```

Operação sobre valores representados em complemento para 2

Nos exemplos anteriores assumiu-se que os números envolvidos eram positivos. Será que a lógica descrita também se aplica a números positivos e negativos representados em complemento para 2? A resposta é sim no caso dos números positivos. No caso dos números negativos também se verifica se houver o cuidado de manter o bit de sinal do número (a divisão por 2 não afecta o sinal do número). Notar que o SHR comum desloca os bits à direita e introduz sempre o valor 0 no bit mais significativo. Para manter o bit de sinal será necessário usar a instrução de deslocamento aritmético (SHRA).

Exemplos do uso de SHRA de 1 bit (para simplificar ilustra-se com palavras de apenas 4 bits):

```
0010 (+2)  ->  0001 (+1)
1100 (-4)  ->  1110 (-2)
```