

Linguagens de montagem

Capítulo 8 - Entrada e Saída

Ricardo Anido
Instituto de Computação
Unicamp

Arquitetura *E/S mapeada na memória*:

- ▶ As instruções LD, LDB, ST e STB podem ser utilizadas para comunicação entre o processador e os dispositivos.
- ▶ Uma parte do espaço de endereçamento da memória é reservada para E/S, de modo que um dispositivo de E/S, e não a memória, responda a requisições de acesso do processador no endereço atribuído ao dispositivo.
- ▶ Suponha que o endereço 0x90000000 seja atribuído a um dispositivo. Então, quando uma instrução LDB especificando o endereço 0x90000000 é executada, ao invés de um byte ser lido da memória, o processador lê um byte do dispositivo.

Arquitetura de *E/S isolada*:

- ▶ O processador inclui em seu repertório instruções dedicadas de E/S para transferir dados entre o processador e dispositivos de E/S.
- ▶ Essas instruções dedicadas usam um espaço de endereçamento diferente do espaço de endereçamento da memória.

- ▶ Ambas as arquiteturas têm vantagens e desvantagens.
- ▶ A E/S mapeada na memória exige menos hardware, tornando a produção do processador mais simples e menos custosa.
- ▶ A E/S isolada permite que todo o espaço de endereçamento seja ocupado pela memória, o que pode ser importante se o espaço de endereçamento não é muito grande.

Instruções de E/S

- ▶ O LEG possui instruções dedicadas de E/S, e um espaço de endereçamento de E/S que permite endereçar 256 dispositivos distintos.
- ▶ Na arquitetura E/S isolada, endereços de E/S são comumente chamados de *portas*.
- ▶ As portas no LEG têm endereços entre 0 e 255, de forma que podem ser codificadas em um byte.
- ▶ As instruções de E/S do LEG podem transferir bytes ou palavras entre o processador e dispositivos de E/S.

Instruções de E/S

IN			
Carrega registrador com palavra de E/S			
Syntax	Operação	Flags	Codificação
$\text{in } rd, \text{expr8}$	$rd \leftarrow e_s32(imd8)$	—	<div><div>310</div><div><div>0x60</div><div><i>imd8</i></div><div><i>rd</i></div><div>—</div></div></div>
$\text{in } rd, rf$	$rd \leftarrow e_s32(rf)$	—	<div><div>310</div><div><div>0x61</div><div>—</div><div><i>rd</i></div><div><i>rf</i></div></div></div>

Instruções de E/S

INB			
Carrega registrador com byte de E/S			
Syntax	Operação	Flags	Codificação
<code>inb <i>rd</i>, <i>expr8</i></code>	$rd \leftarrow e_s8(imd8)$	—	<div><div>310</div><div><div>0x62</div><div><i>imd8</i></div><div><i>rd</i></div><div>—</div></div></div>
<code>inb <i>rd</i>, <i>rf</i></code>	$rd \leftarrow e_s8(rf)$	—	<div><div>310</div><div><div>0x63</div><div>—</div><div><i>rd</i></div><div><i>rf</i></div></div></div>

Instruções de E/S

OUT

Escreve palavra de registrador em E/S

Syntax	Operação	Flags	Codificação
<code>out <i>expr8</i>, <i>rf</i></code>	$e_s32(imd8) \leftarrow rf$	–	<div><div>310</div><div><div>0x64</div><div><i>imd8</i></div><div>–</div><div><i>rf</i></div></div></div>
<code>out <i>rd</i>, <i>rf</i></code>	$e_s32(rd) \leftarrow rf$	–	<div><div>310</div><div><div>0x65</div><div>–</div><div><i>rd</i></div><div><i>rf</i></div></div></div>

OUTB

Escreve byte de registrador em E/S

Syntax	Operação	Flags	Codificação
<code>outb <i>expr8</i>, <i>rf</i></code>	$e_s8(imd8) \leftarrow rf$	–	<div> <div>31</div> <div>0</div> <div>0x66</div> <div><i>imd8</i></div> <div>–</div> <div><i>rf</i></div> </div>
<code>outb <i>rd</i>, <i>rf</i></code>	$e_s8(rd) \leftarrow rf$	–	<div> <div>31</div> <div>0</div> <div>0x67</div> <div>–</div> <div><i>rd</i></div> <div><i>rf</i></div> </div>

- ▶ A instrução OUT copia um registrador *rf* para uma porta de E/S
- ▶ A instrução OUTB copia o byte menos significativo de um registrador *rf* para uma porta de E/S.
- ▶ A porta de E/S onde o dado deve ser escrito pode ser codificada ou como um valor imediato no campo *imd8* ou em um registrador *rd*.

Instruções de E/S

- ▶ Como no caso de acessos à memória, em uma instrução de E/S o processador espera um tempo fixo, o *tempo de preparação*, para o dispositivo processar a operação requerida (leitura ou escrita) e os barramentos estabilizarem.
- ▶ Por exemplo, em uma instrução INB o processador
 - ▶ coloca no barramento de endereço o valor da porta especificada na instrução (em *imd8* ou *rf*)
 - ▶ coloca no barramento de controle $mem/\overline{io} = 0$ e $rd/\overline{wr} = 1$
 - ▶ espera o tempo de preparação especificado
 - ▶ copia o valor dos oito bits menos significativos do barramento de dados para o byte menos significativo do registrador *rd*, zerando os outros bits.

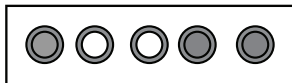
Exemplo de instruções de E/S

```
                                |@ exemplos de uso de E/S
                                |
00030000 [60 20 04 00] |      in  r4,0x20      @ leitura de uma palavra
                                |                        @ de E/S, end. imediato
0003000c [63 00 06 07] |      inb r6,r7        @ leitura de um byte
                                |                        @ de E/S, end. indireto
                                |                        @ por registrador
00030004 [65 00 01 02] |      out r1,r2       @ escrita de uma palavra
                                |                        @ em E/S, end. indireto
                                |                        @ por registrador
00030008 [66 30 00 05] |      outb 0x30,r5    @ escrita de um byte
                                |                        @ em E/S, end. imediato
```

Exemplo de dispositivo: painel de leds

- ▶ vamos considerar um dispositivo muito simples: um *painel de leds*, composto de um conjunto de leds.
- ▶ O painel de leds é um dispositivo de escrita apenas (ou seja, não pode ser lido), com uma porta de dados.
- ▶ Os leds podem ser acesos ou apagados através da escrita de um valor na porta de dados do painel de leds.
- ▶ Cada bit do valor escrito é associado a um led do painel, sendo que o led mais à direita no painel corresponde ao bit menos significativo do valor escrito.

Exemplo de dispositivo: painel de leds



Problema

Escreva um programa para mostrar continuamente os valores de 0 a 255, no formato binário, em sequência, usando um painel de oito leds. Considere que a porta de dados do painel de leds é 0x90.

Solução

@ programa para contar em binário, mostrando valor em painel de leds
@ executa continuamente, contando de 0 a 255

@ define constantes

```
LEDS      .equ 0x90      @ porta de dados do painel de leds
INTERVALO .equ 0xffff    @ valor para contador temporizador
```

```
.org 0x100
```

conta_leds:

```
set  r1,INTERVALO  @ valor inicial do contador
set  r0,1           @ valor inicial para leds
outb LEDS,r0       @ escreve valor inicial
```

loop:

```
sub  r1,1           @ decrementa contador temporizador
jnz  loop           @ espera contador temporizador chegar a zero
add  r0,1           @ adiciona 1 ao valor a ser mostrado
outb LEDS,r0       @ escreve valor nos leds
set  r1,INTERVALO  @ reinicializa contador
jmp  loop           @ e continua
```


Uso de dispositivos de E/S com o simulador

- ▶ O simulador permite o uso de *dispositivos de E/S* como painel de leds, botões, mostradores de sete segmentos, teclado e outros.
- ▶ Os dispositivos devem ser declarados em um *arquivo de descrição de dispositivos* que deve ser carregado com a opção “-d” do simulador

Uso de dispositivos de E/S com o simulador

Leds podem ser de quatro cores, vermelho, azul, amarelo e verde. O formato da descrição de um painel de leds é

```
%leds NOME_PAINEL  
LEDS PORTA_DADOS
```

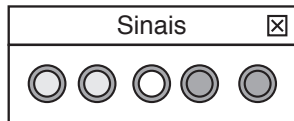
onde

- ▶ `NOME_PAINEL` é um nome que será usado na janela do painel.
- ▶ `LEDS` é uma sequência de letras que especifica os leds do painel. Cada letra corresponde a um led, e pode ser `r` (vermelho), `b` (azul) `g` (verde) ou `y` (amarelo). Cada led é associado a um bit na palavra de dados. O led descrito pela letra mais à esquerda em `LEDS` corresponde ao led mais significativo do valor, que aparecerá mais à esquerda no painel de leds.
- ▶ `PORTA_DADOS` é o endereço da porta de dados do painel de leds, de escrita apenas. A escrita de um valor na porta de dados liga ou desliga os leds correspondentes.

Uso de dispositivos com o simulador

```
%leds Sinais  
rrygg 0x41
```

(a) Arquivo de configuração



(b) Painel criado

Uso de dispositivos com o simulador

Suponha que o arquivo `conta_leds.s` contenha o programa `conta_leds`, e o arquivo `dispositivos.txt` contenha a seguinte descrição de dispositivos:

```
%leds CONTADOR  
rrrrrrrr 0x90
```

Então, use o montador `lasm` para montar o programa e o simulador `legsim` para executar, usando a opção `-d` para instanciar o painel de leds

```
$ lasm -l conta_leds conta_leds.s  
$ legsim -d dispositivos.txt -l conta_leds
```

E inicie a simulação, digitando `"g conta_leds"`.

Sincronização

- ▶ Exemplos, até aqui, muito simples e não necessitam *sincronização*
- ▶ Considere agora um teclado:



- ▶ Como saber se uma tecla foi pressionada?

- ▶ Teclado usa duas portas, estado e dado: leitura na porta de estado indica o *estado* (tem dado para ser lido?), leitura na porta de dados retorna a tecla pressionada
- ▶ Solução ruim: executar um loop testando estado, até que tecla seja pressionada (!!?)

Sincronização

```
@ *****
@ le_tecla
@ *****
    KEYBD_DATA .equ 0x40 @ porta de dados
    KEYBD_STAT .equ 0x41 @ porta de estado
    KEYBD_READY .equ 1   @ bit READY
    KEYBD_OVRN .equ 2    @ bit OVRN

le_tecla:
    set    r1,KEYBD_READY

le_tecla1:
    inb    r0,KEYBD_STAT @ lê porta de estado
    tst    r0,r1         @ dado pronto para ser lido?
    jz     le_tecla1     @ espera que dado esteja pronto
    set    r1,KEYBD_OVRN
    tst    r0,r1         @ houve erro?
    jnz    le_tecla_erro @ sim, desvia para tratar
    inb    r0,KEYBD_DATA @ le porta de dados
    ret

@ aqui trata erro (não mostrado)
le_tecla_erro:
```

Interrupções

- ▶ Maneira melhor de sincronizar processador e dispositivos de E/S: inverter a iniciativa da comunicação.
- ▶ Dispositivo toma a iniciativa de informar ao processador que tem um dado pronto.
- ▶ Enquanto não houver dado disponível para ser lido, o processador pode ser utilizado para executar outras tarefas.
- ▶ Quando um dispositivo tem um dado disponível, ele *interrompe* a tarefa que está sendo executada pelo processador.
- ▶ O processador acessa o dispositivo de forma a tratar a razão da interrupção, e após volta a executar a tarefa que havia sido interrompida.

Interrupções

- ▶ Conjunto de ações, em hardware e em software, para implementar essa abordagem é conhecido como *mecanismo de interrupção*.
- ▶ O trecho de código associado ao tratamento de uma interrupção específica é chamado de *rotina interrupção* para aquele dispositivo.
- ▶ O mecanismo de interrupção envolve hardware dedicado tanto no processador como nos dispositivos. Ele deve garantir que:
 1. não haja nenhuma interferência na tarefa que é interrompida, seja antes, durante ou após o tratamento da interrupção
 2. o processador possa determinar qual dispositivo necessita de atenção
 3. o processador possa determinar que trecho de código deve ser executado para atender o dispositivo que necessita de atenção

Interrupções

- ▶ Conjunto de ações, em hardware e em software, para implementar essa abordagem é conhecido como *mecanismo de interrupção*.
- ▶ O trecho de código associado ao tratamento de uma interrupção específica é chamado de *rotina interrupção* para aquele dispositivo.
- ▶ O mecanismo de interrupção envolve hardware dedicado tanto no processador como nos dispositivos. Ele deve garantir que:
 1. não haja nenhuma interferência na tarefa que é interrompida, seja antes, durante ou após o tratamento da interrupção
 2. o processador possa determinar qual dispositivo necessita de atenção
 3. o processador possa determinar que trecho de código deve ser executado para atender o dispositivo que necessita de atenção

Interrupções - Ausência de interferência

- ▶ Suponha que o processador esteja executando um programa P
- ▶ ao término da execução de uma instrução A , logo antes de iniciar a execução da instrução seguinte, B , o processador verifica que há uma solicitação de interrupção pendente.
- ▶ Processador inicia o tratamento da interrupção, executa a rotina de interrupção correspondente, e após terminar o tratamento da interrupção o processador retoma a execução do programa P .
- ▶ Tratamento da interrupção ocorre como se o programa que está executando fizesse uma invocação (involuntária) da rotina de interrupção para o dispositivo.
- ▶ O mecanismo de interrupção portanto deve empilhar o endereço da instrução B na pilha quando a interrupção é aceita (endereço de retorno).
- ▶ Ao final da rotina de interrupção o endereço de retorno deve ser usado para voltar a executar o programa P .

Interrupções - Ausência de interferência

- ▶ Considere agora que A é uma instrução aritmética e B é uma instrução de desvio condicional que depende do resultado de A .
- ▶ Se a rotina de interrupção executa alguma instrução aritmética ou lógica (o que normalmente ocorre), os bits de estado do processador podem ser alterados!
- ▶ Mecanismo de interrupção deve preservar também o registrador de bits de estado antes do início do tratamento da interrupção, e este deve ser restaurado ao final do tratamento da interrupção, antes da execução de B .
- ▶ Registradores usados pela rotina de interrupção também devem ser salvos na pilha (pelo programador).

Interrupções - Determinando o dispositivo que necessita de atenção

- ▶ O processador inclui um hardware adicional, chamado de *controlador de interrupções*, que é o responsável por determinar qual o dispositivo que necessita de atenção.
- ▶ Cada dispositivo é ligado ao controlador de interrupções do processador através de um fio, *int*, do barramento de controle.
- ▶ Cada uma das interrupções é identificada por um número inteiro, chamado de *tipo* da interrupção. Cada tipo de interrupção é associado a um nível de prioridade.

Interrupções - Determinando o dispositivo que necessita de atenção

- ▶ Um dispositivo requisita uma interrupção sinalizando o controlador de interrupções através do fio *int*.
- ▶ Se há mais de uma interrupção pendente, o controlador de interrupções é responsável por informar ao processador o tipo da interrupção mais prioritária que está pendente.
- ▶ O LEG reconhece 256 interrupções distintas, com tipos de 0 a 255 e níveis fixos de interrupção.
- ▶ No LEG, quanto menor o tipo, maior a prioridade da interrupção.

Interrupções - Determinando o dispositivo que necessita de atenção

- ▶ Ao término de cada instrução executada, o processador consulta o controlador de interrupções para saber se existe uma requisição de interrupção pendente.
- ▶ Se existe, o controlador de interrupção informa o tipo da instrução mais prioritária.
- ▶ O processador então usa o tipo da interrupção para determinar o endereço da rotina de interrupção correspondente.

Interrupções - Determinando o endereço do tratador da interrupção

- ▶ o processador utiliza um vetor de endereços, chamado de *vetor de interrupções*.
- ▶ No LEG os elementos do vetor de interrupções são endereços, e os índices são os tipos de interrupções: elemento 0 do vetor contém o endereço da rotina de interrupção para o tipo 0, o elemento 1 contém o endereço da rotina de interrupção para o tipo 1, e assim por diante.
- ▶ No LEG, o vetor de interrupções tem endereço fixo, a partir da posição 0 da memória.

Interrupções - Determinando o endereço do tratador da interrupção

- ▶ Como o LEG tem 256 tipos de interrupções, e cada elemento do vetor ocupa quatro bytes, o vetor de interrupções ocupa os primeiros 1024 bytes da memória.
- ▶ O programador deve garantir que, antes que uma interrupção ocorra, o endereço do tratador dessa interrupção esteja corretamente instalado no vetor de interrupções, e o registrador apontador de pilha esteja posicionado em uma região de memória válida.

O mecanismo de interrupção

- ▶ consulta o controlador de interrupções para determinar o tipo T da interrupção mais prioritária;
- ▶ empilha o endereço da instrução que seria executada após a instrução corrente se não houvesse interrupção (esse é o endereço de retorno da interrupção);
- ▶ empilha o registrador de bits de estado;
- ▶ faz um acesso à memória para determinar o endereço do tratador da interrupção T , no endereço $T \times 4$ (o vetor de interrupções inicia na posição 0 da memória e cada endereço é uma palavra de 32 bits) e coloca o valor lido no registrador ip .
- ▶ retoma execução a partir do endereço ip .

Retorno de interrupção

- ▶ Para efetuar o retorno de uma rotina de interrupção é utilizada uma instrução específica, IRET (retorno de interrupção).
- ▶ A instrução IRET retira da pilha os valores do endereço de retorno e dos bits de estado, restaurando respectivamente o apontador de instruções e o registrador de bits de estado.

Retorno de interrupção

Retorno de interrupção							
Sintaxe	Operação	Flags	Codificação				
iret	$ip \leftarrow mem[sp]$ $sp \leftarrow sp + 4$ $flags \leftarrow mem[sp]$ $sp \leftarrow sp + 4$	todas	<div>310</div> <table><tr><td>0x58</td><td>—</td><td>—</td><td>—</td></tr></table>	0x58	—	—	—
0x58	—	—	—				

Habilitando e desabilitando interrupções

CLI			
Desabilita interrupções			
Sintaxe	Operação	Flags	Codificação
cli	–	$I \leftarrow 0$	<div>310</div> <div>0x72–––</div>

STI			
Habilita interrupções			
Sintaxe	Operação	Flags	Codificação
sti	–	$I \leftarrow 1$	<div>310</div> <div>0x73–––</div>

Interrupções

- ▶ O processador, ao ser iniciado, tem as interrupções inibidas.
- ▶ o processador também desliga o bit de estado de interrupção automaticamente quando uma solicitação de interrupção é aceita. O bit de estado I é desligado logo após o processador empilhar o registrador de estado.
- ▶ uma nova interrupção não será aceita enquanto a rotina de interrupção para a interrupção que acabou de ser aceita está sendo executada, a menos que o programador explicitamente habilite novamente as interrupções, dentro da rotina de interrupção, através de uma instrução STI; e
- ▶ as interrupções serão ativadas automaticamente ao final da rotina de interrupção, assim que a instrução IRET for executada, pois esta instrução restaura o registrador de bits de condição (e se a interrupção foi aceita é porque as interrupções estavam habilitadas, ou seja o bit I estava ligado).

Problema

Escreva um programa para um sistema simples composto por um teclado e um mostrador de sete segmentos, para atualizar o mostrador a cada vez que uma tecla é pressionada, de forma que o mostrador indique o valor da tecla pressionada. O teclado deve utilizar o mecanismo de interrupção para sinalizar que uma tecla foi pressionada.

Definição de Mostrador para o simulador

```
%7segs Relógio  
0x21  
0x22
```

(a) Arquivo de configuração



(b) Painel criado

Solução

```
@ *****
@ mostra_tecla
@ *****
@ algumas constantes
    FIM_MEMORIA .equ 0x10000 @ para inicializar a pilha
    TRATADA      .equ 0x0f    @ indica que tecla já foi tratada
@ endereços de portas e bits de estado
    KEYBD_DATA   .equ 0x40    @ porta de dados
    KEYBD_STAT   .equ 0x41    @ porta de estado
    DISPLAY_DATA .equ 0x30    @ porta do mostrador
    KEYBD_READY  .equ 1       @ bit READY
    KEYBD_OVRN   .equ 2       @ bit OVRN
@ tipos das interrupções
    INT_KEYBD     .equ 0x10    @ tipo de interrupção do teclado
@ Vetor de interrupções. Apenas o tipo que interessa é inicializado
    .org INT_KEYBD*4          @ posição no vetor de interrupções onde
                                @ deve ser colocado o endereço da rotina
                                @ de interrupção. O montador se encarrega
                                @ de montar o valor correto.

.word trata_int_teclado
```

Solução

@ início do programa, após o vetor de interrupções

```
.org 0x400
```

mostra_tecla:

```
    set    sp,FIM_MEMORIA      @ prepara pilha
    sti                      @ habilita interrupções
    set    r0,0                @ no início, apaga o mostrador
    outb   DISPLAY_DATA,r0
    set    r0,TRATADA          @ no início, indica que
    stb     valor_tecla,r0      @ tecla foi tratada
```

mostra_tecla_espera:

```
    ldb    r0,valor_tecla
    cmp    r0,TRATADA          @ tecla nova pressionada?
    jz     mostra_tecla_espera @ se não, desvia e espera
    set    r1,tab_digitos      @ indexa valor lido no vetor de dígitos
    add    r1,r0                @ para determinar a configuração de bits
    ldb    r0,[r1]              @ a ser escrita no mostrador
    outb   DISPLAY_DATA,r0     @ envia para o mostrador
    set    r0,TRATADA          @ indica que tecla foi tratada
    stb     valor_tecla,r0
    jmp    mostra_tecla_espera @ e não tem mais nada a fazer
```

```
trata_int_teclado:           @ rotina de interrupção
    push    r0               @ preserva valor dos registradores
    inb     r0,KEYBD_DATA    @ lê porta de dados
    stb     valor_tecla,r0   @ armazena valor lido
    pop     r0               @ restaura registrador
    iret                    @ e retorna

@ variáveis
valor_tecla:
    .skip 1
tab_digitos:
    .byte   0x7e,0x30,0x6d,0x79,0x33,0x5b,0x5f,0x70,0x7f,0x7b,0x4f,0x4f
```

- ▶ existe outro tipo de evento, interno ao processador, que também pode ser tratado com o mecanismo de interrupção: *Exceção*,
- ▶ Exceção ocorre devido a algum problema na execução de uma instrução.
- ▶ Exemplo: divisão por zero.
- ▶ Outro exemplo: *instrução inválida*, disparada quando o processador detecta que o código da instrução a ser executada não corresponde a uma instrução válida.

- ▶ Alguns tipos de interrupção são pré-definidos como sendo reservados para exceções,
- ▶ O hardware, no caso de ocorrência de uma exceção, gera uma interrupção do tipo correspondente.

Interrupções e o sistema operacional

- ▶ Sistema operacional deve garantir que execução de um aplicativo não afete a execução de outro aplicativo, nem afete a execução do próprio sistema operacional.
- ▶ Exemplo:

```
1 trecho_malicioso:
2     cli                @ desabilita interrupções
3 laco_malicioso:
4     jmp     laco_malicioso    @ e fica eternamente no laço
```

Interrupções e o sistema operacional

- ▶ Algumas instruções não podem ser executadas por programas de usuários.
- ▶ Essas instruções só podem ser executadas quando o processador está em um modo especial.
- ▶ O processador tem dois modos de operação: modo *usuário* e modo *supervisor*.
- ▶ as instruções são divididas em dois tipos: *normais* e *privilegiadas*.

Interrupções e o sistema operacional

- ▶ Quando o processador está no modo usuário, somente as instruções normais estão disponíveis para execução.
- ▶ Se uma instrução privilegiada é executada enquanto o processador está no modo usuário, uma exceção é gerada.
- ▶ No modo supervisor, todas as instruções são permitidas.

Interrupções e o sistema operacional

- ▶ No LEG, todas as instruções de E/S são privilegiadas.
- ▶ Um bit do registrador de estado, identificado por S, indica o modo corrente do processador: usuário se $S = 0$, supervisor se $S = 1$.

Interrupções e o sistema operacional

- ▶ Quando processador inicia sua operação, está no modo supervisor.
- ▶ O sistema operacional inicia sua operação: instala os tratadores de interrupções no vetor de interrupções, protege regiões da memória (como vetor de interrupções e variáveis internas) contra escrita, carrega o código do sistema operacional do disco, etc.
- ▶ Após todas as inicializações, e logo antes de executar um programa de usuário, o sistema coloca o processador em modo usuário.
- ▶ Dessa forma, quando o programa de usuário executa, instruções privilegiadas são proibidas.

Interrupções e o sistema operacional

- ▶ O mecanismo de interrupção leva em conta a existência do bit de estado S.
- ▶ Quando um programa de usuário está sendo executado e uma interrupção é aceita, o mecanismo de interrupção faz com que o processador mude para o modo supervisor após armazenar na pilha o registrador de estado.
- ▶ Assim, quando o processador executa uma rotina de interrupção, todas as instruções estão disponíveis.

O mecanismo de interrupção completo no LEG

- ▶ A cada instrução que termina de executar, o processador verifica o bit de estado I. Se $I = 1$, o processador consulta o controlador de interrupções para saber se há requisições pendentes. Se não há interrupções pendentes, ou se as interrupções não estão habilitadas (bit de estado $I = 0$), o processador prossegue executando a próxima instrução do programa corrente.
- ▶ Se $I = 1$ e há requisição de interrupção pendente, o processador inicia o mecanismo de interrupção, executando os passos de 1 a 8 a seguir.
- ▶ Se durante a execução de uma instrução uma exceção ocorre, o processador também executa os passos de 1 a 8, independentemente do valor corrente do bit de estado I.

O mecanismo de interrupção completo no LEG

1. empilha o endereço da próxima instrução a ser executada;
2. empilha o registrador de bits de estado;
3. no registrador de estado faz $I = 0$, e $S = 1$;
4. se é uma exceção, determina seu tipo T . Se é uma interrupção, consulta o controlador de interrupções para determinar o tipo T da interrupção mais prioritária;
5. faz um acesso à memória para determinar o endereço da rotina de interrupção do tipo T .
6. executa a rotina de interrupção. A última instrução executada por uma rotina de interrupção deve ser IRET.
7. retoma a execução do programa interrompido, a partir do endereço em ip .

Chamada ao Sistema Operacional

- ▶ Programas de usuários não podem acessar dispositivos de E/S diretamente, pois qualquer acesso causa uma exceção.
- ▶ A única maneira de um programa de usuário interagir com E/S é através de *serviços* oferecidos pelo sistema operacional.
- ▶ Exemplos: leitura e escrita de um dispositivo, como um disco rígido ou uma placa de rede.

Chamada ao Sistema Operacional

- ▶ Os sistemas operacionais oferecem interfaces bem definidas para acesso a seus serviços.
- ▶ Uma requisição de um serviço do sistema operacional é denominada de *chamada de sistema*, porque em geral a interface para o serviço é na forma de uma função ou procedimento.
- ▶ Como os programas de usuários executam em modo usuário, e o sistema operacional executa em modo supervisor, deve haver uma maneira de fazer com que o processador, ao iniciar a execução de um serviço em nome de um usuário, passe temporariamente para o modo supervisor

Chamada ao Sistema Operacional

- os processadores incluem em seu repertório de instruções uma instrução especial, que codifica como um valor imediato o tipo de interrupção que deve ser acionado e que, quando executada, dispara o mecanismo de interrupção.

Chamada ao sistema operacional							
Sintaxe	Operação	Flags	Codificação				
$\text{sys } \text{expr8}$	$\text{mem}[sp] \leftarrow \text{flags}$ $sp \leftarrow sp + 4$ $\text{mem}[sp] \leftarrow ip$ $sp \leftarrow sp + 4$ $I \leftarrow 0, S \leftarrow 1;$ $ip \leftarrow$ $\text{mem}[4 \times \text{imm8}]$	IS	<div>310</div> <table border="1"><tr><td>0x57</td><td>imd8</td><td>—</td><td>—</td></tr></table>	0x57	imd8	—	—
0x57	imd8	—	—				

Chamada ao Sistema Operacional

```
@ *****
@ Chamada ao sistema operacional: write
@ *****
@ Descrição
@ Chamada ao sistema operacional para escrever uma cadeia de bytes
@ em um dispositivo de E/S
@ Tipo da interrupção usada: 0x55
@ Parâmetros de entrada
@ r7 com tipo do serviço (0x04 para write)
@ r0 com número do dispositivo (1 escrever na saída padrão)
@ r1 com endereço do início da cadeia de bytes
@ r2 com número de bytes da cadeia
@ Retorno
@ se não houve erro:
@     r0 com zero
@ se houve erro:
@     r0 com número negativo que indica tipo de erro
```

Problema

Escreva um procedimento `escreve_cadeia` para escrever na saída padrão (console) uma cadeia de caracteres terminada por zero, usando a chamada ao sistema `write`. O endereço de início da cadeia de caracteres é dado no registrador `r0`.

Solução

```
@ *****
@ escreve_cadeia
@ *****
@ constantes
    WRITE      .equ 4          @ tipo de chamada ao sistema
    CONSOLE    .equ 1          @ descritor do dispositivo saída padrão
escreve_cadeia:
    mov     r1,r0              @ r1 tem início da cadeia
    mov     r2,r0              @ vamos usar r2 para procurar final
    sub     r2,1
escreve_cadeia1:
    add     r2,1
    ldb     r3,[r2]            @ procura final da cadeia
    cmp     r3,0               @ que é indicado por byte 0
    jnz     escreve_cadeia1    @ continua laço se não encontrou final
    set     r7,WRITE           @ tipo de serviço é write; r1 já tem endereço
    set     r0,CONSOLE         @ dispositivo queremos acessar em r0
    sub     r2,r1              @ número de bytes a serem escritos em r2
    sys     0x55               @ executa chamada a sistema
    ret
```