Linguagens de montagem
Capítulo 8 - Entrada e Saída

Ricardo Anido
Instituto de Computação
Unicamp

Arquitetura de E/S

Arquitetura *E/S mapeada na memória*:

- As instruções LDR, LDRB, STR e STRB 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

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.

Arquitetura de E/S

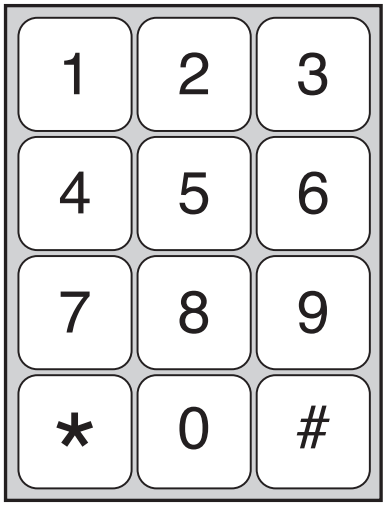
- Ambas as arquiteturas têm vantagens e desvantagens.
- A E/S mapeada na memória exige menos hardware, tornando a produção do processador mais simples e menos custosa.
- A E/S isolada permite que todo o espaço de endereçamento seja ocupado pela memória, o que pode ser importante se o espaço de endereçamento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

Instruções de E/S

- O Pentium possui instruções dedicadas de E/S, e um espaço de endereçamento de E/S que permite endereçar 4096 dispositivos distintos (endereços entre 0 e $0xFFF$).
- Na arquitetura E/S isolada, endereços de E/S são comumente chamados de *portas*.
- As instruções de E/S do Pentium podem transferir bytes ou palavras entre o processador e dispositivos de E/S.

Sincronização

- Exemplos muito simples não necessitam *sincronização*
- Considere agora um teclado:



- Como saber se uma tecla foi pressionada?

Sincronização

- 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
@ ********
     .equ   KEYBD_DATA,0x90040  @ porta de dados
     .equ   KEYBD_STAT,0x90041  @ porta de estado
     .equ   KEYBD_READY,1       @ bit READY
     .equ   KEYBD_OVRN,2        @ bit OVRN
le_tecla:
     ldr   r0,=KEYBD_STAT       @ lê porta de estado
     ldrb  r0,[r0]
     tst   r0,#KEYBD_READY      @ dado pronto para ser lido?
     beq   le_tecla             @ espera que dado esteja pronto
     tst   r0,#KEYBD_OVRN       @ houve erro?
     bne   le_tecla_erro        @ sim, desvia para tratar
     ldr   r0,=KEYBD_DATA       @ le porta de dados
     ldrb  r0,[r0]
     bx    lr                   @ e retorna
@ 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 de tratamento de 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 preservar o endereço da instrução B 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 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.
- No ARM, há dois tipos de interrupção externa: IRQ e FIQ.

Interrupções - Determinando o dispositivo que necessita de atenção

- Um dispositivo requisita uma interrupção sinalizando o controlador de interrupções através do fio *int_irq* ou do fio *int_fiq*.
- 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.

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 insterrupção mais prioritária (IRQ ou FIQ).
- 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 ARM os elementos do vetor de interrupções são em geral instruções de desvio os índices são os tipos de interrupções.
- 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.

Exceções

- 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.

Exceções

- 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 exceções

- Interrupções e exceções no ARM são associadas aos modos de operação do processador.
- O ARM possui duas interrupções externas: FIQ (interrupção rápida, do inglês *Fast Interrupt Request*) e IRQ (interrupção normal, do inglês *Interrupt Request*), e quatro tipos de exceções: *Reset*, *Data abort*, *Pre-fetch abort* e *Undefined instruction*.

Interrupções e exceções

- As diferentes interrupções e exceções têm prioridades, indicadas por números de 1 a 6 (quanto menor o número maior a prioridade).
- Se um tipo mais prioritário de interrupção ou exceção está sendo atendido, os eventos de outros tipos ficam pendentes.
- Para as interrupções externas FIQ e IRQ o registrador CPSR mantém dois bits de controle, respectivamente bits F e I, que indicam se a interrupção deve ser tratada. Se o bit de controle correspondente no registrador CPSR é igual a zero, a interrupção está habilitada e é atendida; caso o bit correspondente seja igual a um (interrupção desabilitada), a interrupção fica pendente.

Interrupções e exceções

Interrupção/Exceção	Prioridade	Número
Reset	1	0
Data Abort	2	4
FIQ	3	7
IRQ	4	6
Prefetch Abort	5	3
SVC	6	2
Undefined Instruction	6	1

Modos de Operação

Nome	Descrição
User	Modo usuário, único modo sem nenhum privilégio de execução.
FIQ	Entra neste modo quando uma interrupção do tipo FIQ é aceita.
Interrupt	Entra neste modo quando uma interrupção do tipo IRQ é aceita.
Supervisor	Modo supervisor, entra neste modo quando o processador inicia, reinicia ou executa de chamada ao sistema.
Abort	Modo aborto de acesso, entra neste modo quando uma exceção de acesso a memória é disparada (acesso desalinhado de palavra, por exemplo).
Undefined	Modo instrução indefinida, entra neste modo quando uma exceção de instrução indefinida é disparada.
System	Modo sistema, único modo em que o processador entra por programa.

Registradores

- O processador ARM tem 37 registradores, mas apenas 17 (ou 18, em alguns modos de operação) são acessíveis a cada momento.
- 13 são registradores de propósito geral (`r0` a `r12`).
- Os outros quatro registradores têm funções específicas:
  - `sp` (*stack pointer*), apontador de pilha, similar ao registrador homônimo do LEG, também acessado pelo nome `r13`.
  - `lr` (*link register*), registrador de ligação, também acessado pelo nome `r14`. Recebe o endereço de retorno em chamadas de procedimento.
  - `pc` (*program counter*), contador de programa, também acessado pelo nome `r15`.
  - `CPSR` registrador de estado corrente do programa.

Registradores

Registradores de propósito geral e contador de programa

User/System		FIQ		Supervisor		Abort		IRQ		Undefined	
r0	0	r0	0	r0	0	r0	0	r0	0	r0	0
r1	1	r1	1	r1	1	r1	1	r1	1	r1	1
r2	2	r2	2	r2	2	r2	2	r2	2	r2	2
r3	3	r3	3	r3	3	r3	3	r3	3	r3	3
r4	4	r4	4	r4	4	r4	4	r4	4	r4	4
r5	5	r5	5	r5	5	r5	5	r5	5	r5	5
r6	6	r6	6	r6	6	r6	6	r6	6	r6	6
r7	7	r7	7	r7	7	r7	7	r7	7	r7	7
r8	8	r8	17	r8	8	r8	8	r8	8	r8	8
r9	9	r9	18	r9	9	r9	9	r9	9	r9	9
r10	10	r10	19	r10	10	r10	10	r10	10	r10	10
r11	11	r11	20	r11	11	r11	11	r11	11	r11	11
r12	12	r12	21	r12	12	r12	12	r12	12	r12	12
r13/SP	13	r13/SP	22	r13/SP	24	r13/SP	26	r13/SP	28	r13/SP	30
r14/LR	14	r14/LR	23	r14/LR	25	r14/LR	27	r14/LR	29	r14/LR	31
r15/PC	15	r15/PC	15	r15/PC	15	r15/PC	15	r15/PC	15	r15/PC	15

Registradores de estado

CPSR	16	CPSR	16	CPSR	16	CPSR	16	CPSR	16	CPSR	16
		SPSR	32	SPSR	33	SPSR	34	SPSR	35	SPSR	36

Mecanismo de interrupção

- muda o modo de operação para o modo correspondente ao evento;
- copia o registrador CPSR no registrador SPSR do modo correspondente;
- armazena o endereço de retorno da interrupção no registrador `lr` (`r14`) do modo correspondente;
- desvia para o endereço *número_da_interrupção* $\times$ 4 (elemento do vetor de interrupção correspondente ao tipo da interrupção ou exceção.

Vetor de interrupções

Normalmente, cada entrada no vetor de interrupções contém uma instrução de desvio incondicional para a rotina de interrupção correspondente.

```
   .org 0
vetor_int:
   b      trata_reset            @ tipo 0, Reset
   ldr    pc,=trata_undef        @ tipo 1, Undefined Instruction
   b      trata_svc              @ tipo 2, SVC
   ldr    pc,=trata_prefetch     @ tipo 3, Prefetch abort
   ldr    pc,=trata_abort        @ tipo 4, Data abort
   .skip  4                      @ tipo 5, reservado
   ldr    pc,=trata_irq          @ tipo 6, IRQ
trata_fiq:                       @ tipo 7, FIQ
   ...                           @ tratador FIQ pode ser colocado aqui
```

Retorno de interrupção

Para retornar do tratamento de uma interrupção ou exceção deve ser utilizada a instrução especial MOVS:

```
movs   pc,lr
```

que restaura o registrador de estado, copiando o registrador de estado SPSR para o registrador de estado CPSR do modo em que o processador estava quando a interrupção foi aceita, e desvia para o endereço de retorno armazenado no registrador `lr`, voltando a executar o código interrompido.

Mudança de modo de execução

- O processador, ao iniciar, está no modo de execução Supervisor.
- Para permitir que o programador prepare cada modo de execução (Supervisor, IRQ, FIQ, etc) que pretende utilizar (preparar a pilha para o modo, por exemplo), o ARM inclui uma instrução que permite a mudança de um modo de execução para outro: MSR (move para registrador de estado).
- O modo de execução corrente é armazenado nos bits 4 a 0 do CPSR; assim, alterando o valor do CPSR é possível alterar o modo de execução corrente.
- Note ainda que a instrução MSR não pode ser executada no modo Usuário.

Mudança de modo de execução

Modos de operação definidos pelos bits de controle M[4:0] do CPSR

M[4:0]	Modo de Operação
10000	*User*
10001	*FIQ*
10010	*IRQ*
10011	*Supervisor*
10111	*Abort*
11011	*Undefined*
11111	*System*

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 armazenar o registrador de estado CPSR no novo modo de operação.
- 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.
- as interrupções serão ativadas automaticamente ao final da rotina de interrupção, assim que a instrução MOVS for executada, pois esta instrução restaura o registrador de estado CPSR (e se a interrupção foi aceita é porque as interrupções estavam habilitadas, ou seja o bit ativa/inibe a interrupção estava ligado)

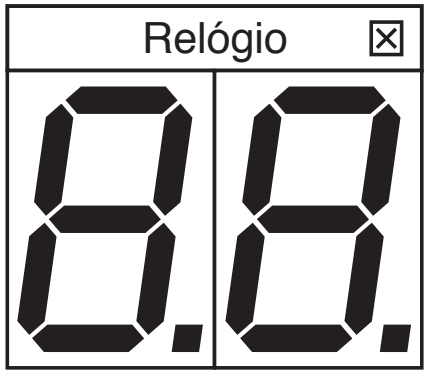
Problema

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
```

Solução

```
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
```

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 dos modos especiais (que não o modo usuário). 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.
- Nos outros modos, todas as instruções são permitidas.

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

- Quando um programa de usuário está sendo executado (no modo usuário) e uma interrupção é aceita, o mecanismo de interrupção faz com que o processador mude para o modo associado ao tipo de interrupção (IRQ ou FIQ).
- Assim, quando o processador executa uma rotina de tratamento de interrupção (ou de exceção), todas as instruções estão disponíveis.

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

```
@ ************************************
@ 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
    .equ  WRITE, 4          @ tipo de chamada ao sistema
    .equ  CONSOLE,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
escreve_cadeia1:
    ldrb  r3,[r2],#1        @ procura final da cadeia
    cmp   r3,#0             @ que é indicado por byte 0
    bne   escreve_cadeia1 @ continua laço se não encontrou final
    mov   r7,#WRITE         @ tipo de serviço é write; r1 já tem endereço
    mov   r0,#CONSOLE       @ dispositivo queremos acessar em r0
    sub   r2,r1             @ número de bytes a serem escritos em r2
    swi   0x55              @ executa chamada a sistema
    bx    lr
```