

Organização Básica de computadores e linguagem de montagem

Prof. Edson Borin

2º Semestre de 2015

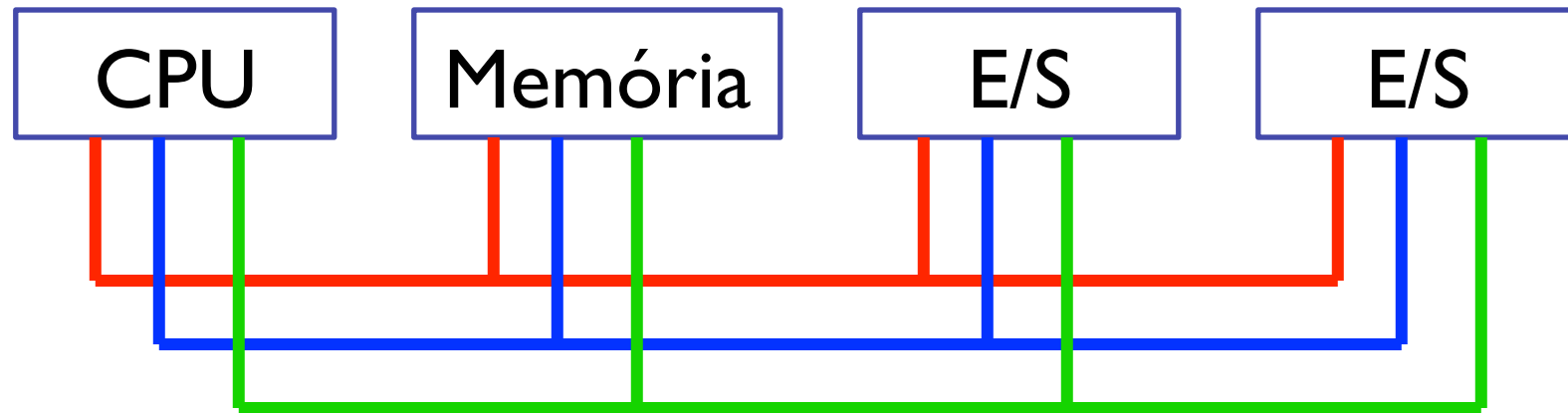


Barramentos

Barramentos

- Caminhos de comunicação entre dois ou mais dispositivos
- Diversas linhas de comunicação que podem ser classificadas em:
 - Linhas (ou barramento) de controle
 - Linhas (ou barramento) de endereço
 - Linhas (ou barramento) de dados
- Exemplos de barramento
 - PCI: desenvolvido originalmente pela Intel. Atualmente é um padrão público
 - AMBA: desenvolvido pela ARM

Barramentos

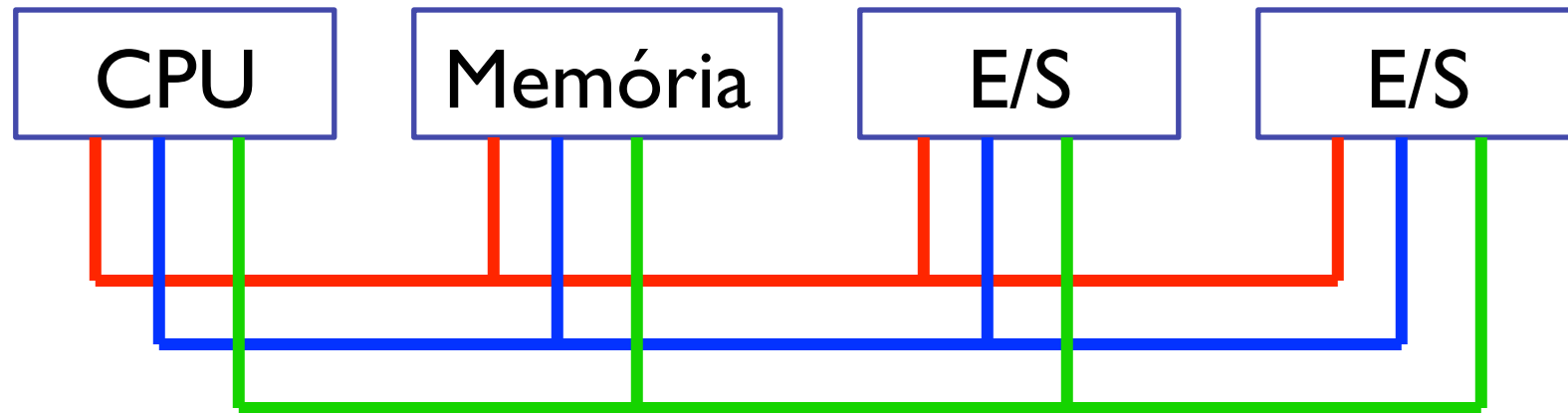


Linhas de dados

Linhas de endereço

Linhas de controle

Barramentos



Linhas de dados

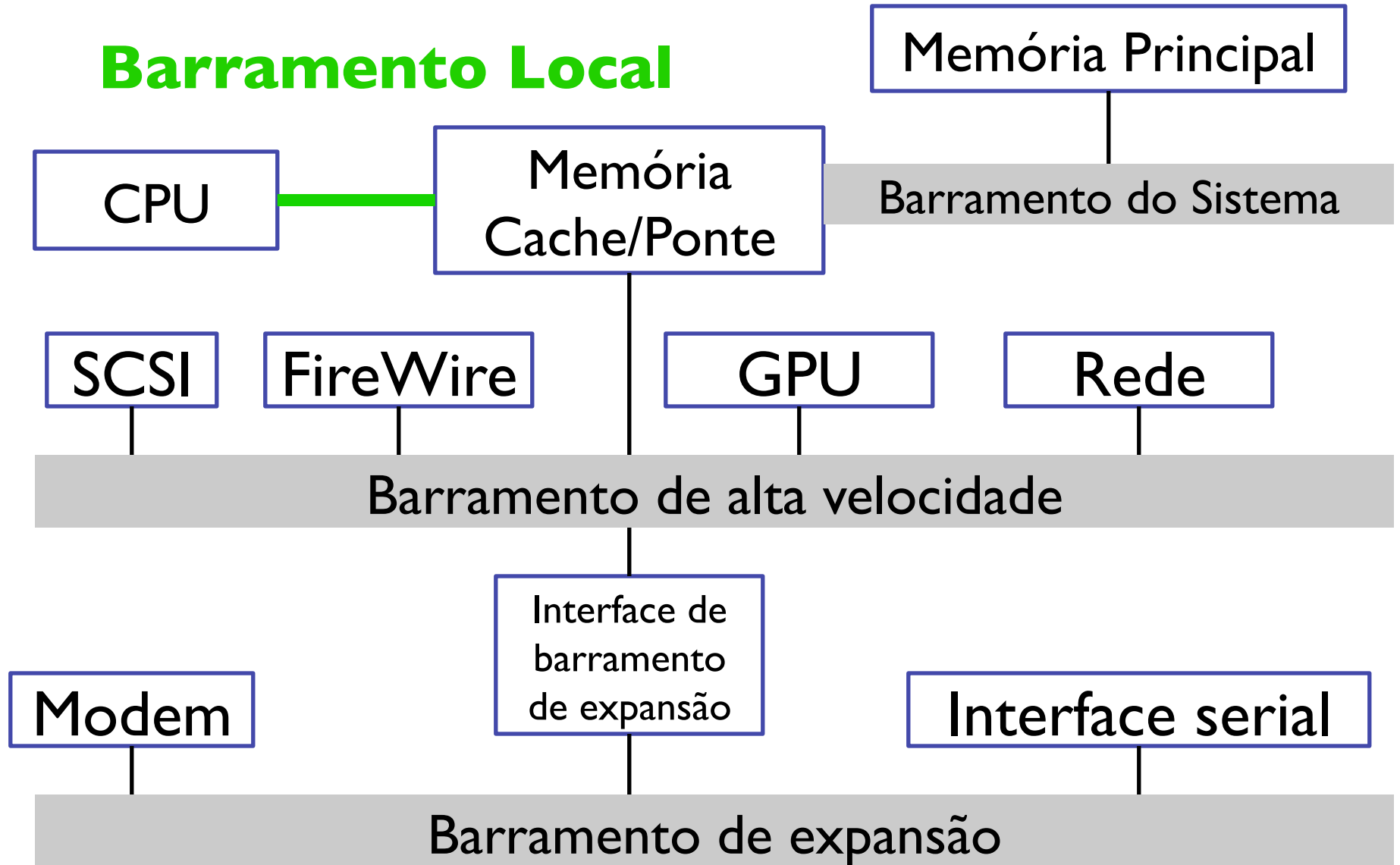
Linhas de endereço

Linhas de controle

- Todos os dispositivos compartilham o mesmo barramento
- Problema: todos têm que operar na mesma velocidade

Barramentos Modernos

Barramento Local



Barramentos

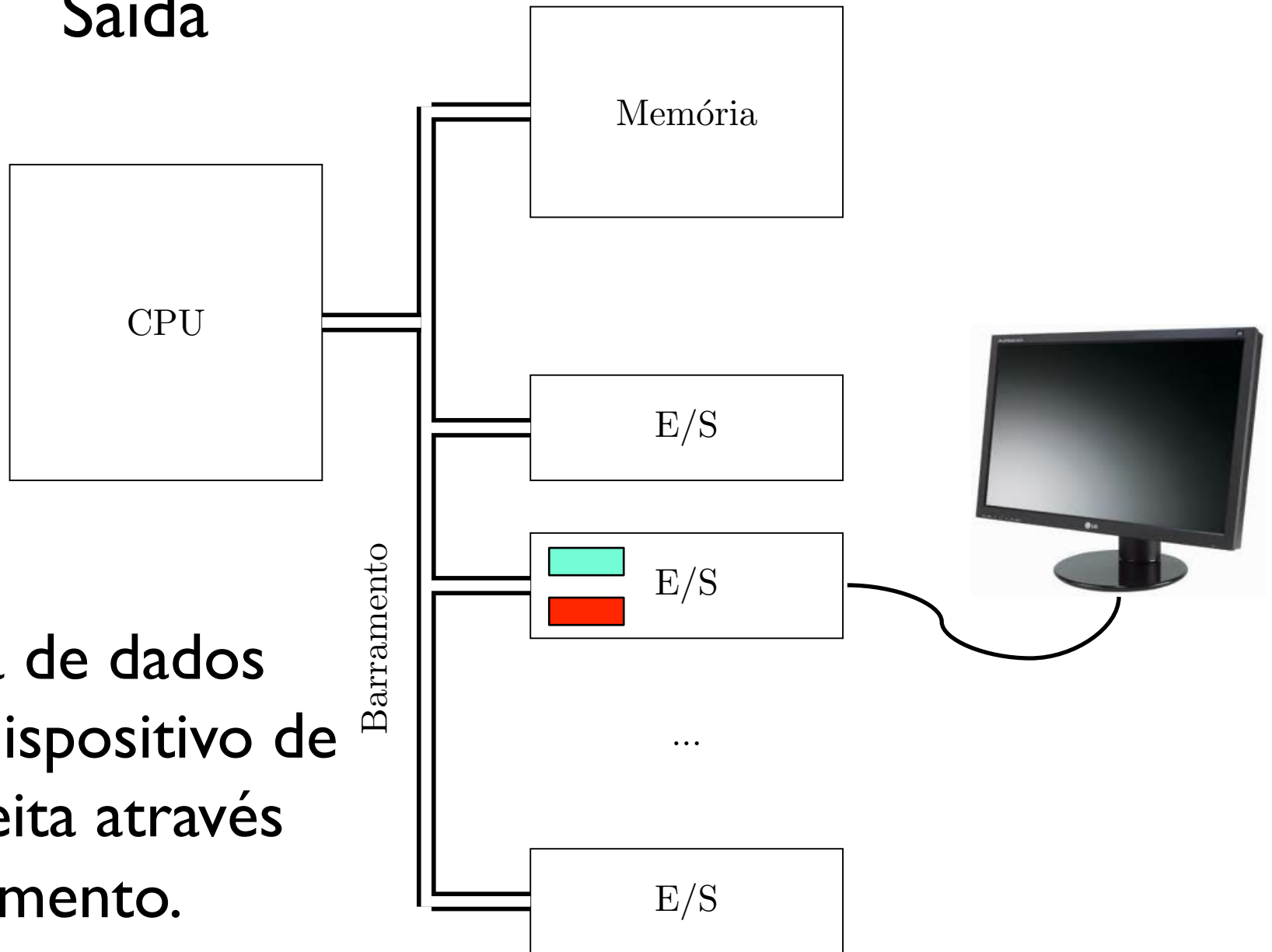
- Leitura recomendada: Capítulo 3.4 do livro do Stallings.

Entrada e Saída

Entrada e Saída

- Dispositivos de E/S (Entrada/Saída) ou I/O (Input/Output) permitem a entrada e saída de dados do processador.
- Ex: Teclado, Mouse, Monitor, Impressora, Placa de rede, disco rígido, unidade de CD-ROM.
- Como funciona?

Saída



A escrita de dados em um dispositivo de saída é feita através do barramento.

Saída

- Como o programa realiza uma saída?

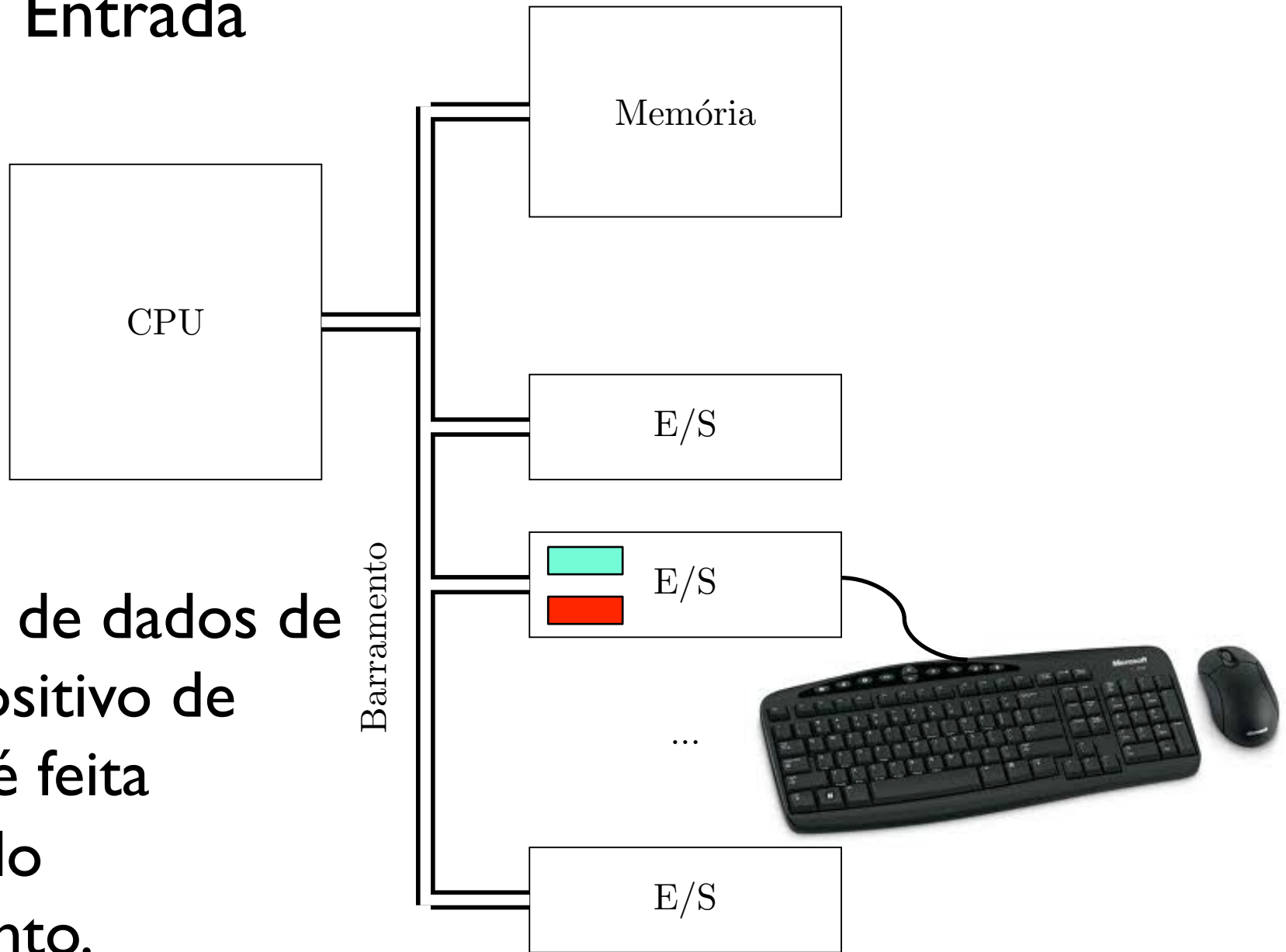
Saída

- Como o programa realiza uma saída?
- Escreve em uma *porta*, que está associada a um dispositivo de saída.

Saída

- Como o programa realiza uma saída?
- Escreve em uma *porta*, que está associada a um dispositivo de saída.
- 2 opções comuns:
 - Instrução especial para saída. Ex:
`out 0x10, r1`
 - Instrução de *store* em uma faixa de endereços reservada.
Ex:
`ldr r0, =0x80000`
`str r1, [r0]`
- Como o processador sabe se é uma saída ou acesso à memória?

Entrada



A leitura de dados de um dispositivo de entrada é feita através do barramento.

Entrada

- Como o programa realiza uma entrada?
- Lê de uma *porta*, que está associada a um dispositivo de entrada.
- 2 opções comuns:
 - Instrução especial para entrada. Ex:
`in r1, 0x10`
 - Instrução de *load* em uma faixa de endereços reservada.
Ex:
`ldr r0, =0x80000`
`ldr r1, [r0]`
- Como o processador sabe se é uma entrada ou acesso à memória?

Exemplos no ARM

- Entrada:

```
ldr    r0, =0x53FA0008
```

```
ldr    r1, [r0]
```

- Saída

```
ldr    r0, =0x53FA0000
```

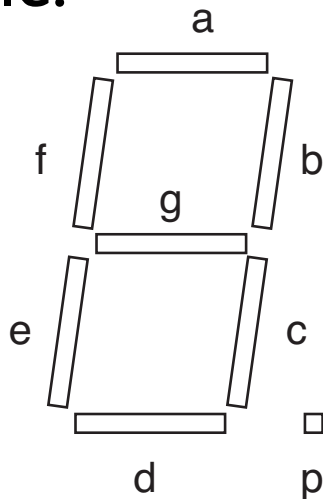
```
str    r1, [r0]
```

Exemplo - Elevador

- Dois dispositivos. 1 de entrada e 1 de saída
- Entrada:
 - sensor de andar. Conectado à porta 0x20.
 - quando acessado, responde com um *byte* indicando o andar atual (de 0 a 9)

Exemplo - Elevador

- Saída
 - mostrador digital: Conectado à porta 0x30
 - dispositivo com 7 segmentos (a,b,...,g) e um ponto luminoso que ficam ligados ou desligados de acordo com o dado no registrador de controle.
 - a saída corresponde em escrever um *byte* no registrador de controle.



7	6	5	4	3	2	1	0
p	a	b	c	d	e	f	g

Exemplo - Elevador

@ Procedimento atualiza andar

@ Lê o andar do sensor e atualiza o valor do mostrador

```
.equ  SENSOR_PORT, 0x20
```

```
.equ  DISPLAY_PORT, 0x30
```

atualiza_andar:

```
    ldr    r1, =SENSOR_PORT
```

```
    ldrb   r1, [r1]           @ lê o valor do sensor
```

```
    ldr    r0, =tab_digitos   @ converte valor para
```

```
    ldrb   r0, [r0, r1]       @ byte de controle
```

```
    ldr    r1, =DISPLAY_PORT  @ escreve byte de controle
```

```
    strb   r0, [r1]           @ no mostrador
```

```
    mov    pc, lr
```

```
tab_digitos: .byte 7e,30,6d,79,33,5b,5f,70,7f,7b
```

Problemas com a abordagem anterior

- Suponha que o elevador suba 8 andares.
- Quando devemos chamar o procedimento `AtualizaAndar`?

Outro Exemplo - Teclado

- Teclado:
 - Dispositivo de Entrada
 - Duas portas: dados (0x40) e estado (0x41)
 - Dado lido representa caractere
- E se o teclado for apertado múltiplas vezes?
- Como saber se o dado que está lá já foi lido?

1	2	3
4	5	6
7	8	9
*	0	#

Outro Exemplo - Teclado

- Teclado:

- um *bit* de estado indica se o dado atual não foi lido pelo processador ainda (READY). *Bit 0*

- outro *bit* de estado indica se mais de um botão já foi apertado antes do processador ler o dado, ou seja, houve dado perdido (OVRN). *Bit 1*

1	2	3
4	5	6
7	8	9
*	0	#

Outro Exemplo - Teclado (*Busy waiting*)

- Rotina le_tecla
- Lê palavra de controle (end. 0x40)
- Se o dispositivo não tiver dado (*bit 0 - READY*)
 - Tenta novamente (*Busy waiting*)
- Se o dispositivo tiver dado
 - Verifica se houve perda de dado (*bit 1 - OVRN*)
 - Se houve perda de dado
 - Trata o erro
 - Senão
 - Lê dado do dispositivo (end. 0x41) e retorna

Outro Exemplo - Teclado (*Busy waiting*)

@ Procedimento lê tecla

@ Lê a o valor da tecla que foi pressionada

```
.equ    KB_DATA,    0x40
```

```
.equ    KB_STAT,    0x41
```

```
.equ    KB_READY,   0x01
```

```
.equ    KB_OVRN,    0x02
```

le_tecla:

```
    ldr    r1, =KB_STAT
```

```
    ldrb   r1, [r1]           @ lê o estado do teclado
```

```
    tst    r1, #KB_READY     @ testa se tem dado pronto
```

```
    beq    le_tecla          @ se não tiver, tenta novamente
```

```
    tst    r1, #KB_OVRN      @ perdeu dado?
```

```
    bne    lt_ovrn           @ se sim, trata erro
```

```
    ldr    r1, =KB_DATA      @ senão, lê dado
```

```
    ldrb   r0, [r1]          @ no mostrador
```

```
    mov    pc, lr
```

lt_ovrn: @ trata erro aqui

Problemas com a abordagem anterior

- Suponha que o usuário demore para apertar algo.
- O que o processador faz?

Problemas com a abordagem anterior

- Suponha que o usuário demore para apertar algo.
- O que o processador faz?
- Como melhorar?
 - Verifique o teclado de tempos em tempos e faça algum trabalho útil no intervalo entre as verificações.

Problemas com a abordagem anterior

- Suponha que o usuário demore para apertar algo.
- O que o processador faz?
- Como melhorar?
 - Verifique o teclado de tempos em tempos e faça algum trabalho útil no intervalo entre as verificações.
 - Ainda há o risco do usuário pressionar a tecla múltiplas vezes antes do programa verificar se alguma tecla foi pressionada.
 - Talvez o usuário não seja tão rápido para causar este problema, mas e se for uma placa de rede.

Problemas com a abordagem anterior

- Suponha que o usuário demore para apertar algo.
- O que o processador faz?
- Como melhorar?
 - Verifique o teclado periodicamente para algum trabalho útil no teclado.
 - Ainda há o risco de o usuário pressionar algumas teclas múltiplas vezes antes de pressionar alguma tecla.
- Talvez o usuário não seja tão rápido para causar este problema, mas e se for uma placa de rede.

Interrupção: O dispositivo avisa o processador quando acontecer alguma coisa!

Interrupção

- Iniciativa de comunicação é do periférico
- Exemplo:
 - Quando o dado está disponível, o teclado “interrompe” o processador.
 - O processador pára o que está fazendo para atender o teclado
 - Após o processamento da leitura, o processador continua com o que estava fazendo.

Interrupção

- **O processador pára o que está fazendo para atender o teclado.**
- O que acontece com o programa que o processador estava executando???

Exemplo: Interrupção

@ programa faça algo útil 1000 vezes

main:

 mov r4, #1000

loop:

 bl algo_util

 sub r4, r4, #1

 cmp r4, #0

 bne loop

 ...

Exemplo: Interrupção

@ programa faça algo útil 1000 vezes

main:

mov r4, #1000

loop:

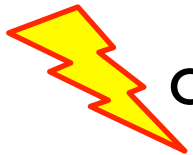
bl algo_util

sub r4, r4, #1

cmp r4, #0

bne loop

...



Interrupção

Interrupção

- **O processador pára o que está fazendo para atender o teclado.**
- O que acontece com o programa que o processador estava executando???
- Antes de tratar a interrupção, é importante salvar todo o “contexto” do programa que está executando
 - Registradores,
 - *flags*,
 - Manter a pilha consistente...

Exemplo: Interrupção

@ programa faça algo útil 1000 vezes

main:

mov r4, #1000

loop:

bl algo_util

sub r4, r4, #1

cmp r4, #0

bne loop

...

trata_interrupcao:

@ salva contexto

@ trata a interrupção

@ restaura o contexto

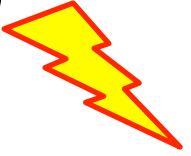
Exemplo: Interrupção

@ programa faça algo útil 1000 vezes

main:

mov r4, #1000

loop:

①  bl algo_util
sub r4, r4, #1
cmp r4, #0
bne loop
...

trata_interrupcao:

@ salva contexto

@ trata a interrupção

@ restaura o contexto

① Interrupção acontece



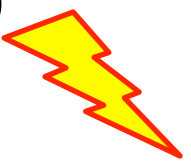

Exemplo: Interrupção

@ programa faça algo útil 1000 vezes

main:

mov r4, #1000

loop:

①  bl algo_util
sub r4, r4, #1
cmp r4, #0
②  bne loop
...

trata_interrupcao:

@ salva contexto

@ trata a interrupção

@ restaura o contexto

① Interrupção acontece

② Fluxo de controle é desviado

Exemplo: Interrupção

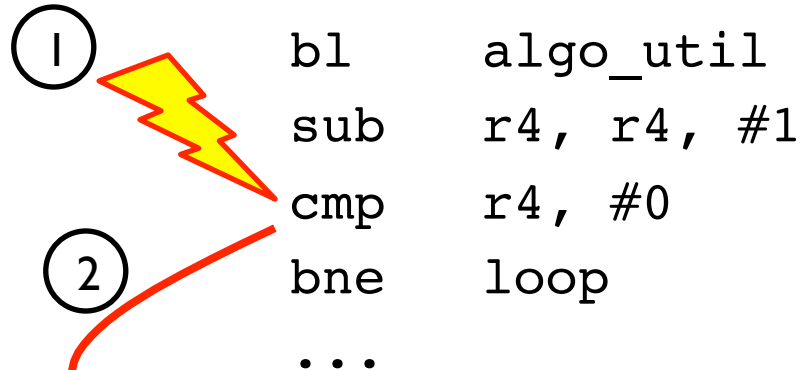
@ programa faça algo útil 1000 vezes

main:

mov r4, #1000

loop:

① bl algo_util
sub r4, r4, #1
cmp r4, #0
② bne loop
...



trata_interrupcao:

③ @ salva contexto
@ trata a interrupção
@ restaura o contexto

① Interrupção acontece

② Fluxo de controle é desviado

③ A interrupção é tratada

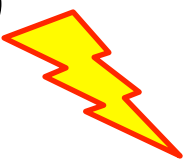
Exemplo: Interrupção

@ programa faça algo útil 1000 vezes

main:

mov r4, #1000

loop:

①  bl algo_util
sub r4, r4, #1
cmp r4, #0
② bne loop
...

trata_interrupcao:

③ @ salva contexto
@ trata a interrupção
④ @ restaura o contexto

① Interrupção acontece

② Fluxo de controle é desviado

③ A interrupção é tratada

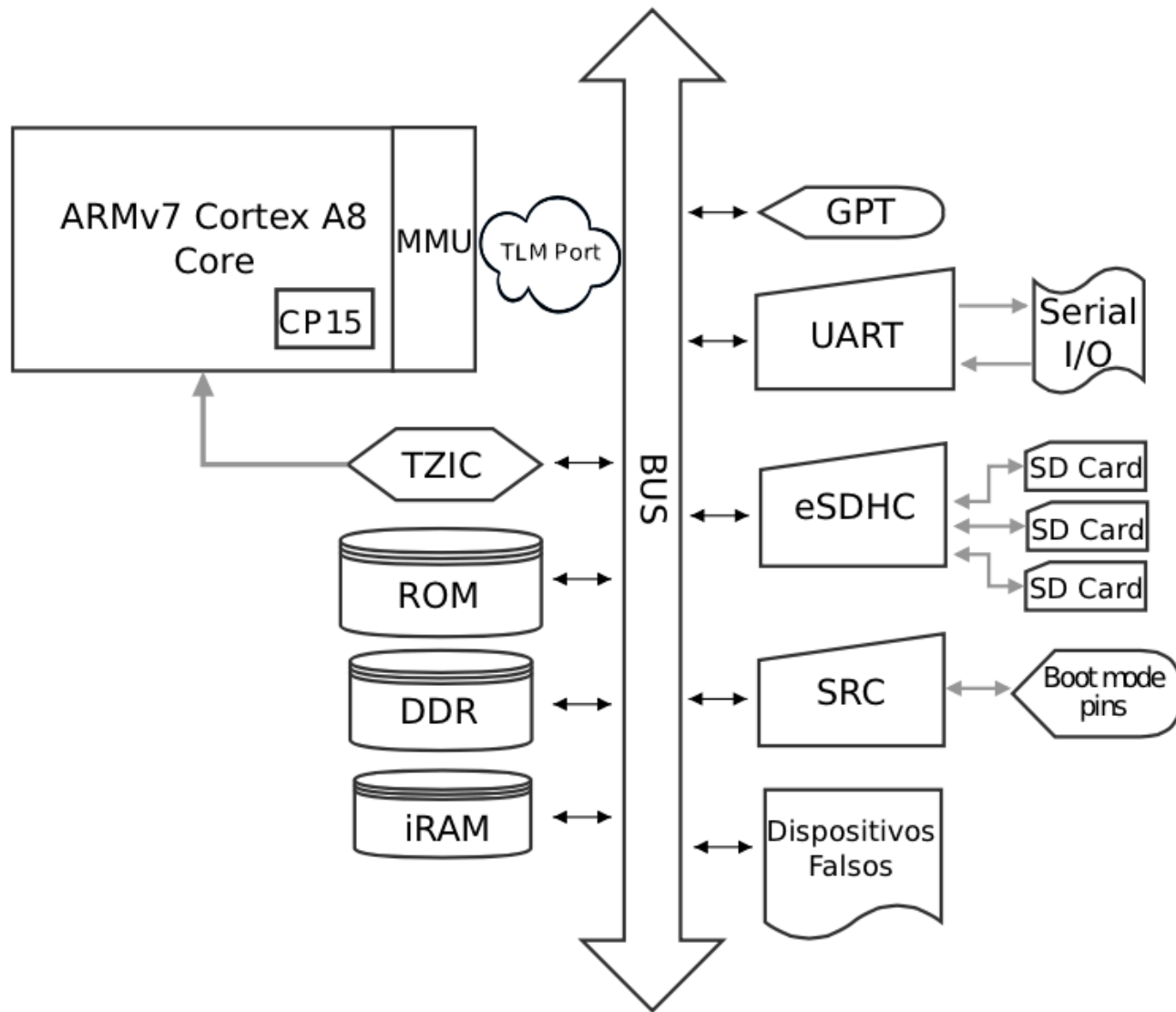
④ Contexto é recuperado

Organização Básica de computadores e linguagem de montagem

Prof. Edson Borin

2º Semestre de 2015

Visão geral de uma plataforma computacional



Interrupção

- Diversos dispositivos de E/S.
- Como chamar o tradador de interrupção correto?

Interrupção

- Diversos dispositivos de E/S.
- Como chamar o tradador de interrupção correto?
- Diversas opções:
 - a) Um único tratador que verifica qual dispositivo interrompeu e chama o procedimento adequado.
 - b) Múltiplos tratadores, um para cada dispositivo.
 - Vetor de interrupções!
 - Armazena o endereço das rotinas de tratamento.

Vetor de Interrupções

- É um **espaço de memória que contém o endereço ou o instruções** das rotinas de tratamento de interrupções
 - **Endereço:** o processador carrega da memória o endereço da rotina e desvia o fluxo de execução para o endereço carregado.
 - **Instruções:** o processador desvia o fluxo de execução para o espaço que contém o código.

Vetor de Interrupções

- No ARM, o vetor de interrupções contém a instrução inicial de cada rotina.
- Vetor inicia-se tipicamente no endereço 0x000

0x00000000	Reset
0x00000004	Undefined Instruction
0x00000008	Software Interrupt
0x0000000C	Prefetch Abort
0x00000010	Data Abort
0x00000014	Reserved
0x00000018	IRQ
0x0000001C	FIQ
	— — — — —

Vetor de Interrupções

- Só cabe 4 *bytes*, ou seja, 1 instrução. Logo a primeira instrução do tratador deve ser um salto para rotina que trata a interrupção.
- Opções:

b reset

ldr pc, =reset

mov pc, #0xEF000000

Vetor de Interrupções

```
.org 0x000
b trata_reset
.org 0x18
b trata_IRQ
...
.org 0x20
trata_IRQ:
    @ salva contexto
    @ trata a interrupção
    @ restaura o contexto
```

Vetor de Interrupções – Salvando o contexto

- Como fazemos para salvar o contexto?

Vetor de Interrupções – Salvando o contexto

- Como fazemos para salvar o contexto?
- O que acontece com o valor de PC do programa que estava sendo executado? Perdemos o valor?

Vetor de Interrupções – Salvando o contexto

- Como fazemos para salvar o contexto?
- O que acontece com o valor de PC do programa que estava sendo executado? Perdemos o valor?
 - **Resposta: o processador salva o valor de PC em um registrador LR especial, o LR_<mode>**
- mode é o modo de operação. Existem 7 modos de operação: *User, System, Supervisor, Abort, Undefined, Interrupt, Fast Interrupt*.

Vetor de Interrupções – Salvando o contexto

- Como fazemos para salvar o contexto?
- O que acontece com o valor de PC do programa que estava sendo executado? Perdemos o valor?
 - Resposta: o processador salva o valor de PC em um registrador LR especial, o LR_<mode>
- Mode é o modo de operação. Existem 7 modos de operação: *User, System, Supervisor, Abort, Undefined, Interrupt, Fast Interrupt*.
 - O modo de operação é selecionado de acordo com a interrupção**

Modos de operação

- Registadores visíveis nos diferentes modos de operação

User32 / System	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13 (sp)	r13_fiq	r13_svc	r13_abt	r13_irq	r13_undef
r14 (lr)	r14_fiq	r14_svc	r14_abt	r14_irq	r14_undef
r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)

Program Status Registers

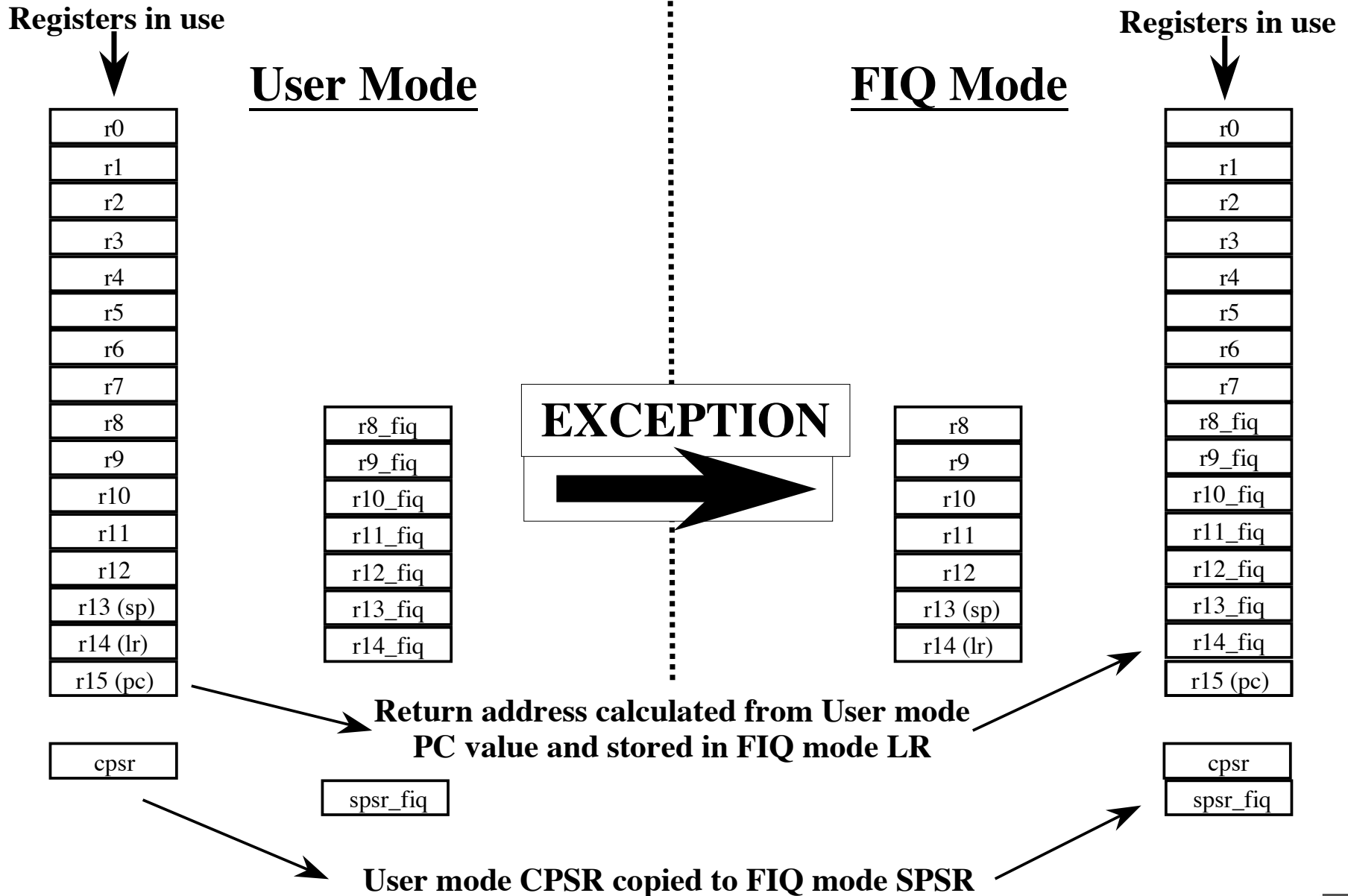
cpsr	cpsr	cpsr	cpsr	cpsr	cpsr
	spsr_fiq	spsr_svc	spsr_abt	spsr_irq	spsr_undef

Vetor de Interrupções – Salvando o contexto

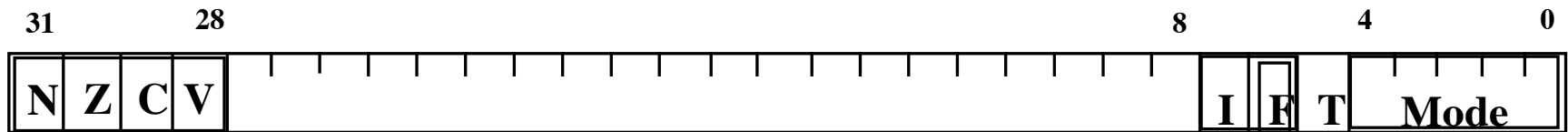
Durante uma interrupção o processador:

1. Copia o CPSR no registrador SPSR_<mode>
2. Seta os *bits* do CPSR que indicam o modo de operação
3. Interrupções IRQ são desabilitadas automaticamente. Interrupções FIQ são desabilitadas somente se a interrupção for do tipo FIQ ou RESET
4. O endereço de retorno é armazenado em LR_<mode>

Interrupção/Exceção



Interrupção/Exceção



Copies of the ALU status flags (latched if the instruction has the "S" bit set).

- * **Condition Code Flags**

N = **N**egative result from ALU flag.

Z = **Z**ero result from ALU flag.

C = ALU operation **C**arried out

V = ALU operation o**V**erflowed

- * **Mode Bits**

M[4:0] define the processor mode.

- * **Interrupt Disable bits.**

I = 1, disables the IRQ.

F = 1, disables the FIQ.

- * **T Bit (Architecture v4T only)**

T = 0, Processor in ARM state

T = 1, Processor in Thumb state

Vetor de Interrupções – Tratando *RESET*

- A interrupção de RESET não precisa salvar o contexto, afinal, foi uma operação de *reset*.
- No entanto, o tratador deve supor que a máquina acabou de ser ligada, e deve preparar o contexto para execução.
 - Configurar os vetores de exceção
 - Inicializar a MMU
 - Inicializar as pilhas e registradores.
 - Inicializar dispositivos de E/S críticos
 - Habilitar interrupções
 - Mudar o modo para execução de código do usuário.

Traps

ou

Interrupções por *software*

Traps / Interrupções por Software

Papéis do sistema operacional:

- Abstrair o funcionamento dos dispositivos de entrada e saída.
- Proteger o sistema contra a execução de código malicioso.
- Gerenciar o acesso aos dispositivos.
- Etc...

Traps / Interrupções por Software

- Ex: Abstrair o funcionamento dos dispositivos de entrada e saída.
 - Como escrever um dado em um arquivo no disco rígido da Seagate, ou da Hitachi?
 - E se o arquivo estiver em um *pen-drive*?

Traps / Interrupções por Software

- Ex: Abstrair o funcionamento dos dispositivos de entrada e saída.
 - Como escrever um dado em um arquivo no disco rígido da Seagate, ou da Hitachi?
 - E se o arquivo estiver em um *pen-drive*?
- O sistema operacional provê uma interface bem definida para acessar arquivos e abstrai os detalhes de acesso ao dispositivo.
 - O *driver* do dispositivo cuida dos detalhes!
- O programa pode acessar o dispositivo diretamente? Sem o auxílio do sistema operacional???

Traps / Interrupções por Software

- Ex: Proteger o sistema contra a execução de código malicioso.
 - O que acontece se um programa executar o seguinte trecho de código:

```
trecho_malicioso:
```

```
    mrs r0, CPSR
```

```
    orr r0, r0, #0xC0
```

```
    msr CPSR, r0
```

```
laco:
```

```
    b laco
```

Traps / Interrupções por Software

- Precisamos de um meio de:
 - 1) proteger o sistema de código malicioso!
 - 2) permitir que o programa do usuário chame o sistema operacional para executar tarefas (E/S, etc...)
- Proteger o sistema de código malicioso:
 - Restringir o código de usuário à execução de instruções seguras. Não permitir a execução de instruções de entrada e saída, msr, e outras.
- Para chamar o sistema operacional:
 - Podemos usar a instrução `b` (ou `bl`)?

Traps / Interrupções por Software

- Vamos supor que nós restringimos o código do usuário.
 - Se nós chamarmos uma rotina do SO com a instrução `b`, o SO conseguirá executar instruções que inibem interrupções, realizam entrada e saída ou outras instruções protegidas?

Traps / Interrupções por Software

- Vamos supor que nós restringimos o código do usuário.
 - Se nós chamarmos uma rotina do SO com a instrução b, o SO conseguirá executar instruções que inibem interrupções, realizam entrada e saída ou outras instruções protegidas?
 - Não

Traps / Interrupções por Software

- Vamos supor que nós restringimos o código do usuário.
 - Se nós chamarmos uma rotina do SO com a instrução b, o SO conseguirá executar instruções que inibem interrupções, realizam entrada e saída ou outras instruções protegidas?
 - Não
- Solução: 2 modos de execução
 - Supervisor: todas as instruções estão disponíveis.
 - Usuário: apenas instruções seguras estão disponíveis.

Traps / Interrupções por Software

- Como ir para o modo supervisor e chamar o SO ao mesmo tempo?
 - *Traps*, ou interrupções por *Software*
- Uma interrupção por *Software* invoca uma função registrada no vetor de interrupções!
 - Ajuda a garantir que apenas o SO executará no modo superusuário
- ARM:
 - Instrução: `svc #0`

Traps / Interrupções por Software

- Processador começa (*boot*) no modo supervisor.
- O *kernel* do SO inicializa os vetores de interrupções, mapas de memória, etc.
- SO prepara a pilha de cada modo de execução
- SO inicia contexto do processo de usuário e muda para o modo usuário.

Traps / Interrupções por Software

- O programa está executando
 - Se o programa precisa chamar o sistema operacional, executa uma interrupção por *Software*
`svc #0`
 - O processador gera uma interrupção, chamando o tratador cadastrado no vetor de interrupções.
 - Neste momento o processador entra no modo supervisor.
 - O tratador do SO realiza a operação e retorna com `movs pc, lr`, trazendo a execução de volta para a aplicação no modo usuário e restaurando o CPSR

Exceções

Exceções

- Interrupções:
 - Eventos causados por dispositivos externos ao processador.
 - Ex: dispositivo de entrada e saída.
 - Estes eventos podem ocorrer a qualquer momento.
- Exceções:
 - Eventos causados pelo próprio processador.
 - Causados durante a execução de instruções.
 - Somente sob certas circunstâncias!
 - Ex: divisão por zero...

Exceções

- Exemplos de exceções:
 - Divisão por zero
 - Execução de instrução inexistente
 - Acesso a regiões de memória protegidas
 - Falta de página
- São eventos infrequentes: **Exceções à regra!**

Exemplo – Divisão por Zero

- O resultado de uma divisão por zero é indefinido.
- Como o processador deve tratar a divisão por zero?

Exemplo – Divisão por Zero

- O resultado de uma divisão por zero é indefinido.
- Como o processador deve tratar a divisão por zero?
 - Resposta: deixe o *software* (programador) tratar.

Exemplo – Divisão por Zero

- O resultado de uma divisão por zero é indefinido.
- Como o processador deve tratar a divisão por zero?
 - Resposta: deixe o *software* (programador) tratar.
- Como?

Exemplo – Divisão por Zero

- O resultado de uma divisão por zero é indefinido.
- Como o processador deve tratar a divisão por zero?
 - Resposta: deixe o *software* (programador) tratar.
- Como?
 - Opção 1: antes de dividir, compare o divisor com zero, se for igual, salte para uma rotina que trata a divisão por zero.
- OBS: este exemplo não se aplica ao ARM do simulador, que não possui instrução de divisão!

Exemplo – Divisão por Zero

```
...  
cmp r1, #0  
beq trata_div_zero  
div r2, r1  
...  
  
trata_div_zero:  
...
```

(a) Verificar o divisor antes.

Exemplo – Divisão por Zero

```
...  
cmp r1, #0  
beq trata_div_zero  
div r2, r1  
...  
  
trata_div_zero:  
...
```

(a) Verificar o divisor antes.

Não é eficiente
verificar se o valor do
divisor é zero toda vez
que realizarmos uma
divisão.

Exemplo – Divisão por Zero

Utilizamos o mecanismo de exceções, similar ao de interrupções!

```
beq trata_div_zero  
div r2, r1  
...
```

```
trata_div_zero:  
...
```

(a) Verificar o divisor antes.

```
.org 0x...  
b trata_div_zero  
  
...  
cmp r1, 0  
beq trata_div_zero  
div r2, r1  
...
```

```
trata_div_zero:  
...
```

(b) Usar o mecanismo de Exceções.

Exceções

- O tratamento de uma exceção é similar ao tratamento de uma interrupção.
 - O processador salva parte do contexto
 - Desvia a execução para o tratador da exceção
 - Tipicamente, o endereço do tratador é armazenado no vetor de interrupções.
 - O tratador da exceção salva o restante do contexto
 - Após tratar a exceção, o tratador “pode” recuperar o contexto e retornar ao programa ou abortar a execução do programa.