1 Interrupções no MIPS

O 'processador' do MIPS executa as "instruções de usuário" dos programas, que são as instruções que estudamos até agora.

O *Control Processor 0* (CP0) é o 'co-processador' que executa as instruções que controlam o ambiente em que os programas de usuário executam. Estas "instruções de controle" são usadas para implementar sistemas operacionais, através de trocas – bem-comportadas – de execução em *modo usuário* para execução *modo sistema*, além do tratamento de todas as condições excepcionais que possam ocorrer durante a execução de um programa. Por ora, nos interessam as interrupções.

Os sinais de interrupção, gerados pelos dispositivos de Entrada e Saída (ES), são amostrados pelo registrador de segmento EXCP_EX_MM, que "une e separa" os segmentos de execução e memória, como mostra a Figura 1.

Quando uma interrupçào é detectada, o CP0 verifica se o evento sinalizado pode ser atendido – quem decide pela viabilidade do atendimento é o projetista do SO. Se o evento pode ser tratado, o estado de execução do processador é alterado para "modo sistema", o endereço da instrução que seria executada, nào fosse a ocorrência da interrupção, é salvado, e a execução desvia para o código que trata as interrupçòes.

A instrução que seria executada, é chamada de "causadora", e seu endereço é salvado no registrador EPC (*Exception PC*).

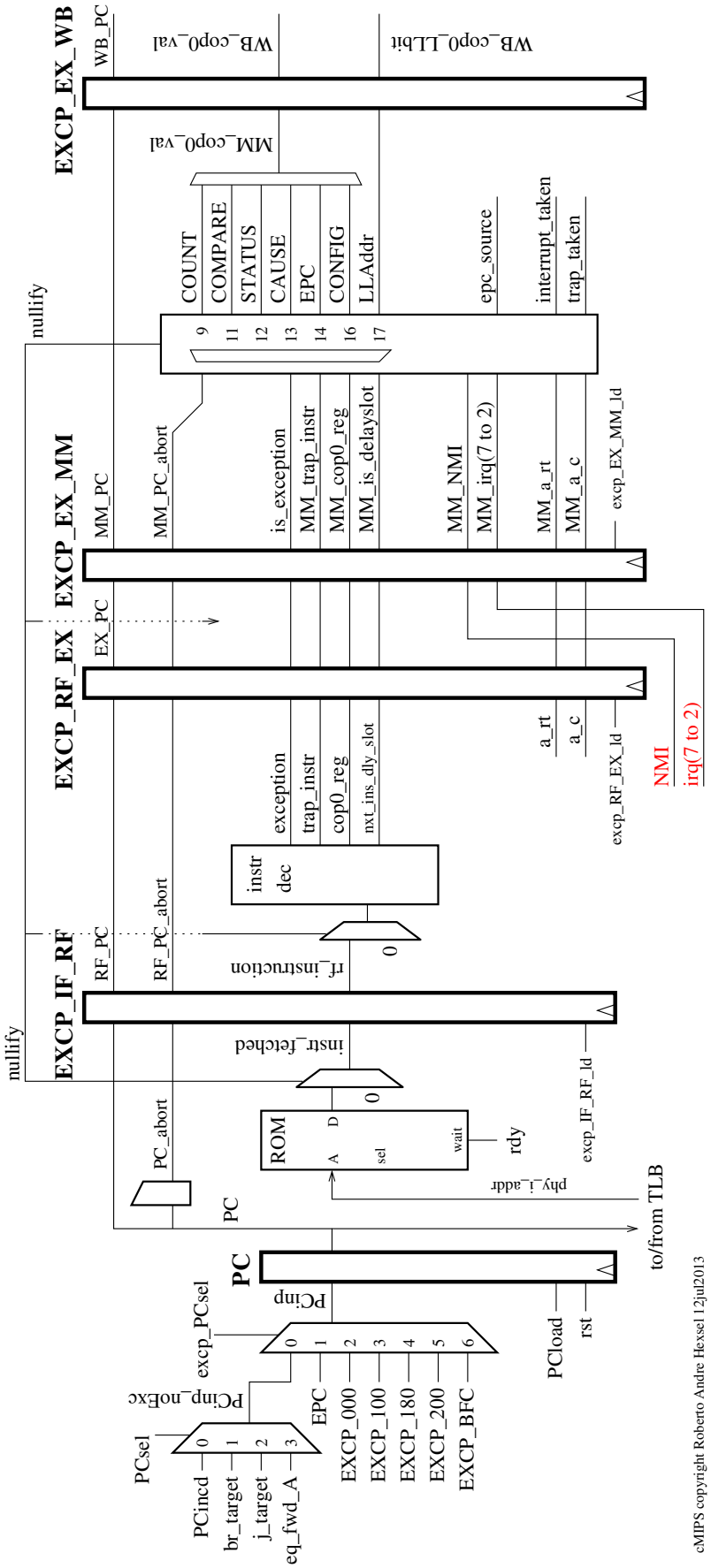


Figura 1: Diagrama de blocos do processador de controle (CP0).

1.1 Registradores do Control Processor 0 – CP0

Os títulos das próximas seções indicam o número do registrador no CP0, e a página de sua definição completa, em

MIPS32 Architecture for Programmers, Volume III: The MIPS32 Privileged Resource Architecture, MIPS Technologies, Rev. 2.50, 2005.

1.1.1 Status (CP0-12, pg.79)

O registrador Status define o modo de execução do processador, e quais são as interrupções habilitadas.

Este registrador é atualizado por *hardware* e alguns bits podem ser alterados por *software*.

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04	03	02	01	00
x		$IM_7..IM_0$	0	UM	0	ERL	EXL	IE

$IM_7..IM_0$ *interrupt mask*; se bit IM_i=1 então a interrupção de nível i está habilitada;

UM *user mode*, UM=1 indica modo usuário, UM=0 indica modo sistema;

ERL *at error level*, ERL=1 quando ocorre um dentre Reset, SoftReset, NMI ou CacheError;

EXL *at exception level*, EXL=1 quando ocorre uma exceção que não seja Reset, SoftReset, NMI ou CacheError;

IE *interrupt enable*, IE=1 indica que as interrupções estão habilitadas.

Os campos *IM, UM, IE* podem ser alterados por *software*.

1.1.2 Cause (CP0-13, pg.92)

O registrador Cause indica a causa da excessão mais recente.

Este registrador é atualizado por *hardware* e alguns bits podem ser alterados por *software*.

31	30	29	28	27	26	25	24	23	22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07	06 05 04 03 02	01 00
BD	TI	0	0	DC	0	0	0	IV	x	$IP_7..IP_0$	0	ExcCode	0

BD *in branch delay*, BD=1 indica que a última exceção ocorreu num *delay slot*;

TI *timer interrupt*, TI=1 indica que há uma interrupção do temporizador pendente;

DC *disable Count register*, DC=1 desabilita o contador interno; DC=0 habilita o contador. Usado para reduzir dissipação de energia em aplicações de baixo consumo;

IV *interrupt uses general exception vector*, IV=0 para tratamento de interrupções como se fossem exceções (0x180) ; IV=1 para o tratamento das interrupções ser desviado para o vetor de interrupções (0x200). Este (IV=1) é o modo usado no *software* do cMIPS;

$IP_7..IP_0$ interrupção pendente; se bit IP_j=1 então a interrupção de nível j está pendente;

ExcCode *exception code*, identifica a exceção, veja os códigos na Tabela 1.

Os campos *TI, DC* podem ser alterados por *software*.

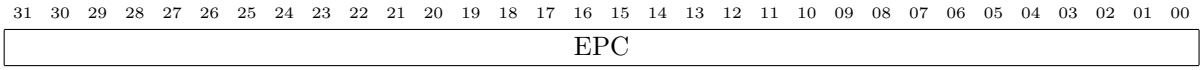
Tabela 1: Códigos de excessão, bits 6 a 2 do registrador Cause.

binário	dec	mnem	causa da excessão
00000	0	Int	interrupção
00001	1	Mod	modificação de página (modificação na TLB)
00010	2	TLBL	excessão da TLB (*load* ou busca)
00011	3	TLBS	excessão da TLB (*store*)
00100	4	AdEL	erro de endereçamento (*load* ou busca)
00101	5	AdES	erro de endereçamento (*store*)
00110	6	IBE	erro no barramento de instruções
00111	7	DBE	erro no barramento de dados
01000	8	Sys	*syscall*
01001	9	Bp	*breakpoint*
01010	10	RI	instrução reservada (opcode inválido)
01100	12	Ov	*overflow*
01101	13	Tr	*trap*
11111	31	–	nenhuma excessão pendente

1.1.3 EPC (CP0-14, pg.97)

Contém o endereço no qual o processamento reinicia após o tratamento de uma interrupção ou excessão. O conteúdo do EPC é atualizado pelo processador na ocorrência de uma excessão (ou interrupção) se Status.EXL=0.

O conteúdo de EPC pode ser alterado e/ou consultado por *software*.



1.2 Instruções de controle

São duas as instruções usadas para acessar os registradores do CP0: `mfc0` (*move from coprocessor 0*), e `mtc0` (*move to coprocessor 0*). As duas tem dois argumentos, um dos 32 registradores de uso geral, e o número de um registrador do CP0.

```
mfc0 r5, c0_status     # r5 <- Status
mtc0 r6, c0_cause      # Cause <- r6
```

A instrução `eret` (*exception return*) copia o conteúdo do EPC para o PC, e faz STATUS_EXL←0.

1.3 Mascaramento de interrupções

São oito as interrupções disponíveis no cMIPS. As interrupções de nível 7 a 2 são as cinco interrupções por *hardware* e que podem ser ligadas a dispositivos externos.

As interrupções são apelidadas de irq, de *Interrupt ReQuest*. A irq7 é associada ao contador interno, a irq6 é ligada à UART, e a irq5 é ligada ao contador externo. As interrupções irq1 e irq0 são interrupções "por *software*" e podem ser disparadas por escritas no registrador Cause.

O programador pode habilitar individualmente cada interrupção ao escrever em Status.IM, ligando ou desligando as "máscaras de interrupção". O trecho de código abaixo habilita as interrupções irq7 e irq6 e desabilita todas as demais.

```
   li   r5, 0xc001        # bits IM7=1, IM6=1, IM5..0=0, IE=1
   mfc0 r6, c0_status     # r6 <- Status
   or   r6, r6, r5        # Status OR (IM7=1,IM6=1,IE=1)
   mtc0 r6, c0_status     # Status <- r6
```

Ao tratar uma interrupção, para determinar quais são as interrupções pendentes, e que *não estão mascaradas*, basta fazer um *and* de Status.IM com Cause.IP.

```
tratador:
   mfc0 k0, c0_cause      # k0 <- Cause.IP
   mfc0 k1, c0_status     # k1 <- Status.IM
   andi k1, k1, 0xff00    # mantém somente bits IM
   and  k1, k0, k1        # k1 <- IP AND IM

   ...  # trata interrupções da maior para a menor prioridade
```

1.4 Alterações no estado do processador

Suponha que uma interrupção é detectada após a decodificação da instrução no endereço 1023 e durante a busca da instrução no endereço 1024. As seguintes alterações no CP0 ocorrem automaticamente e no mesmo ciclo:

EPC $\leftarrow$ 1024	salva endereço da "causadora"
Status.EXL $\leftarrow$ 1	entra em *EXception Level*
Cause.ExcCode $\leftarrow$ 0	sinaliza interrupção (cfe. Tabela 1)
Cause.IP $\leftarrow$ irq_i irq_j ...	mostra interrupções pendentes
	anula instruções em BUSCA, DECOD, EXEC
PC $\leftarrow$ `&( tratador() )`	salta para tratador

As interrupções somente são aceitas se Status.EXL = 0 e Status.ERL = 0 e Status.IE = 1. Ao aceitar uma interrupção, o CP0 faz Status.EXL = 1, o que automaticamente desabilita outras interrupções. Além disso, o *software* decide pelo tratamento em função do mascaramento.

1.5 Código do tratador

Programa 1: Esqueleto/protótipo de um tratador de interrupção.

```
1   void tratador(void) {
2     volatile regCause Cause; // record descreve reg Cause
3
4     Cause = le_reg_cause();
5     // habilita_interrupcoes(); // SE permite aninhamento
6
7     // descobre causa da interrupção e trata evento
8     switch (Cause.IP) {
9        case IRQ7: // Count == Compare
10          // trata do evento mais prioritário
11       case IRQ6: // UART
12          // trata da segunda prioridade
13       default:   // contador externo
14          // trata dos outros eventos
15    }
16    // desabilita_interrupcoes(); // SE permite aninhamento
17    retorna_para_causadora();
18  }
```

Qual deve ser o ambiente de execução para suportar tratadores escritos em C?

Código completo está em `cMIPS/include/{start.s,handlers.s}`.

Programa 2: Tratador/despachante de interrupções – versão cMIPS.

```
1  excp_0200:              # interrupções separadas das excessões
2      mfc0 k1, c0_status
3      mfc0 k0, c0_cause
4      andi k1, k1, 0xff00 # keep only IP bits from Status
5      and  k0, k0, k1     #   and mask them off with IM bits
6      srl  k0, k0, 10     # keep 3 MS bits of IP (irq7..5)
7      lui  k1, %hi(handlers_tbl) #  + 8 By displ in j-table
8      ori  k1, k1, %lo(handlers_tbl)
9      add  k1, k1, k0     # add displacement to base of j-table
10     jr   k1             #   and jump (indirectly) to handler
11     nop
12
13 handlers_tbl:
14     j dismiss             # no request: 000
15     nop
16
17     j extCounter          # lowest priority, IRQ5: 001
18     nop
19
20     j UARTinterr          # mid priority, IRQ6: 010 = 01x
21     nop
22     j UARTinterr          # mid priority, IRQ6: 011
23     nop
24
25     j countCompare        # highest priority, IRQ7: 100 = 1xx
26     nop
27     j countCompare        # highest priority, IRQ7: 101
28     nop
29     j countCompare        # highest priority, IRQ7: 110
30     nop
31     j countCompare        # highest priority, IRQ7: 111
32     nop
33
34 dismiss: # No pending request, must have been noise, just return
35
36     eret                 # Return from interrupt
```

Programa 3: Tratador de interrupções simples, contador da Seção 7.1.

```
1      .set noat               # do not use r1=at
2  extCounter:                 # Handles IP5 = external counter
3      la   k0, HW_counter_addr
4      sw   r0, 0(k0)          # Reset counter, remove IRQ
5      li   k1, 0xc0000100     # Count 256 clock pulses & interr
6      sw   k1, 0(k0)          # Reload counter so it starts again
7      la   k0, count_acum
8      lw   k1, 0(k0)          # Increment time accumulator
9      addi k1, k1, 1
10     sw   k1, 0(k0)
11     eret                    # Return from interrupt
```