PCS 3432 – LABORATÓRIO DE PROCESSADORES

EXPERIÊNCIA 7:

INTERRUPÇÕES

Autores: Prof. Dr. André Riyuiti Hirakawa
Prof. Dr. Carlos Eduardo Cugnasca

2018 – v.01

1. OBJETIVO

Esta experiência visa a familiarização com os mecanismos típicos de interrupção suportados por diversos microprocessadores, bem como as suas aplicações. Foram previstas duas atividades: uma envolvendo a avaliação dos mecanismos de interrupção no simulador IDE68k, e outra na Placa Experimental SBC68k do microprocessador MC 68000.

2. TRANFERÊNCIA DE DADOS EM SISTEMAS MICROPROCESSADOS

Os sistemas baseados em microprocessadores geralmente requerem a interligação com diversos dispositivos periféricos, tais como portas de entrada e saída paralelas, canais de comunicação seriais, controladores de teclado e display, conversores A/D, temporizadores, etc. Muitos desses dispositivos requerem tratamento em intervalos de tempo não necessariamente periódicos por parte do microprocessador, exigindo algum mecanismo de atendimento às suas solicitações. Outros são utilizados em aplicações que exigem pronto atendimento do microprocessador, por exemplo, sistemas e equipamentos projetados para operar em **tempo real**.

2.1 Mecanismos de Transferência de Dados

Para implementar as principais funções de entrada e saída de um sistema baseado em microprocessador, encontram-se disponíveis circuitos integrados programáveis. Muitos desses componentes desempenham boa parte das funções requeridas para a transferência de dados, simplificando o projeto do hardware e do software que o controla e conferindo, assim, melhor desempenho ao conjunto. Algumas arquiteturas utilizam processadores de entrada e saída, distribuindo as tarefas e liberando o microprocessador principal para as tarefas de maior complexidade (por exemplo, periféricos de microcomputadores, como teclados e mouses, costumam utilizar um microcontrolador para o seu controle).

Para implementar a transferência de dados entre os microprocessadores e os dispositivos periféricos são utilizadas diversas técnicas que são escolhidas em função das particularidades de cada aplicação. Algumas das principais técnicas são relacionadas a seguir.

a) Transferência Incondicional

Consiste na execução da operação de entrada ou saída no instante em que o microprocessador puder ou desejar, sem a verificação da viabilidade ou não de executar tal tarefa. É utilizada quando o dispositivo periférico não requer tratamento especial. Exemplos: coleta de informações do estado de chaves modificadas com pouca frequência, como as de configuração, envio de programação ou comandos aos dispositivos periféricos, leitura de palavras de estado de dispositivos periféricos, envio de informações de sinalizações para LEDs, lâmpadas e *displays* em interfaces sem multiplexação.

b) Transferência Condicional

Também conhecida como ***Wait-for-Flag***, consiste na execução da operação de entrada ou saída condicionada à ocorrência de outro evento externo (Figura 1.a). O microprocessador normalmente executa um *loop* de programa, efetuando a leitura de informações sobre uma dada porta de entrada, que fornece a informação da ocorrência ou não de dado evento (***status*** do periférico, ou ***flag***). Caso esse evento tenha ocorrido, a transferência é efetuada, encerrando-se o *loop*. Esse método apresenta como inconvenientes:

- a ausência de outros processamentos durante a execução do *loop*;
- o consumo de tempo útil de trabalho;
- a maior dificuldade de tratamento de mais de um evento, principalmente no que diz respeito à atribuição de prioridades a eles;
- a maior dificuldade na elaboração do software.

Em algumas aplicações que não requerem pronto tratamento, é possível intercalar algum processamento no *loop*, realizado entre leituras de palavras de *status*. As maiores aplicações desse método correspondem a situações nas quais o microprocessador nada tem a fazer enquanto o dispositivo periférico não sinalizar a ocorrência de um evento, como comunicações paralelas com sincronismo e altas velocidades, recebimento de comandos em canais seriais, etc.

c) Transferência Programada ou Amostragem

Essa técnica é uma variação da transferência condicional, quando o evento externo ocorre em um certo intervalo de tempo, normalmente grande com relação à velocidade do microprocessador. Nesse caso, as transferências são efetuadas periodicamente, com o microprocessador controlando o intervalo de tempo (Figura 1.b). Exemplos: aquisição de dados de conversores A/D, saídas pulsadas, varredura de teclados, multiplexação de *displays*, etc. Como inconveniente tem-se a necessidade de ajustar o intervalo de tempo por software, muitas vezes em função do tempo de execução do programa, e que nem sempre garante precisão desejada para aplicações de controle. Para se resolver esse problema, procura-se garantir uma maior precisão do

intervalo de amostragem t_a, por meio do uso de interrupções periódicas, determinadas por um oscilador preciso. Trata-se de uma técnica muito utilizada em equipamentos destinados a supervisão e controle de processos, uma vez que a digitalização dos algoritmos de controle obriga a retirada de amostras dos sinais e atuação em intervalos de tempo iguais.

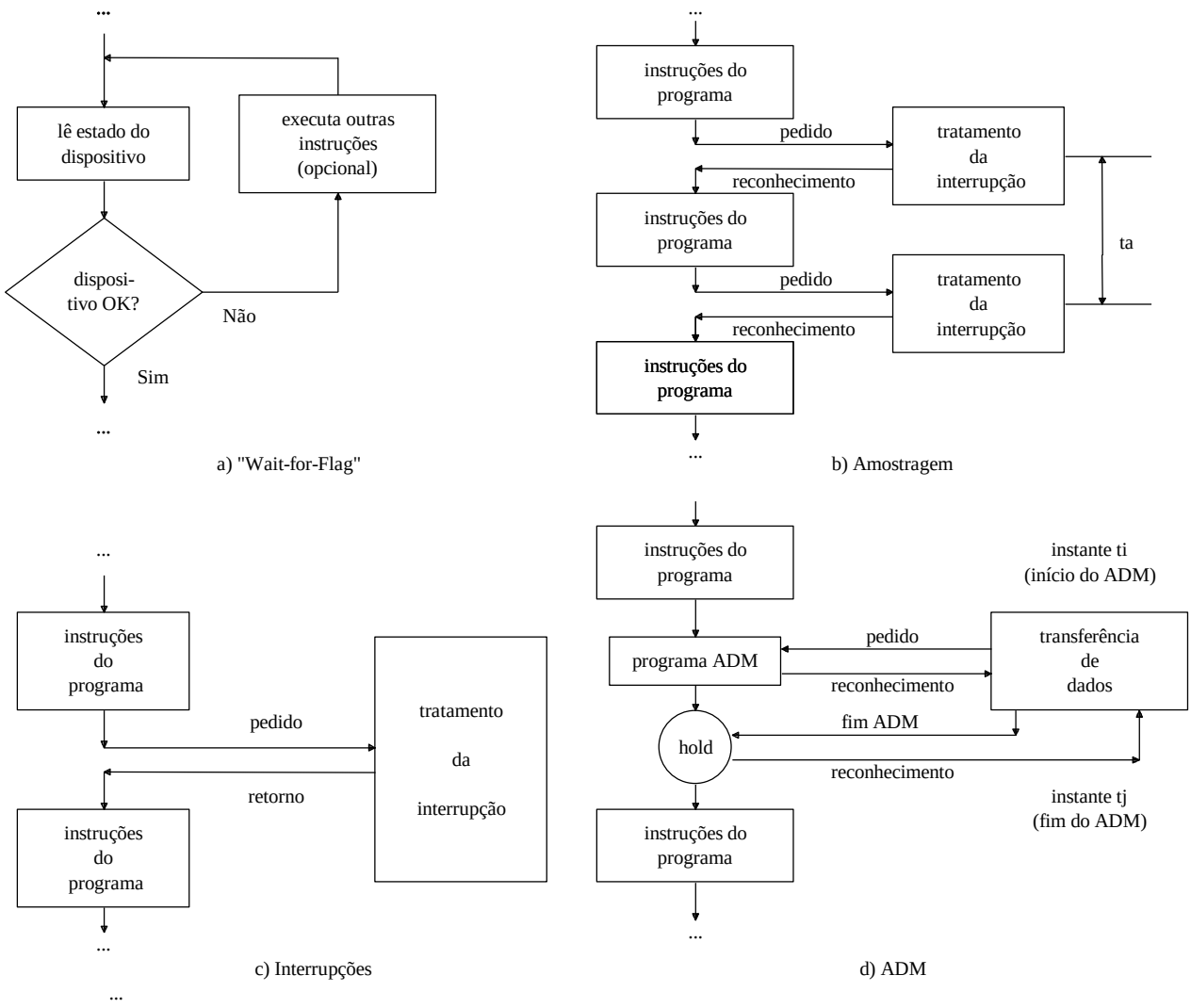
d) Interrupção

Para tornar mais eficiente o tratamento de dispositivos periféricos, que solicitam ações do microprocessador assincronamente ou periodicamente, existe a técnica baseada em um recurso sempre disponível em microprocessadores, denominado **interrupção**. Essa técnica requer sinais que todo microprocessador apresenta e, eventualmente, algum circuito externo adicional. Utilizando os sinais de entrada de interrupção do microprocessador, os dispositivos periféricos podem informar a ocorrência de um dado evento, que será tratado com maior facilidade e rapidez, através das **sub-rotinas de tratamento de interrupção**, sem muito comprometer o programa que estava em execução, ou seja as condições internas do microprocessador são preservadas para posterior continuação da execução do programa interrompido (Figura 1.c). Com esse método, o microprocessador não necessita consumir tempo de processamento para pesquisar a ocorrência de eventos externos, e pode garantir um tempo reduzido para iniciar o seu tratamento. Exemplos: sistemas com muitos dispositivos periféricos, uso de dispositivos periféricos cujos eventos envolvidos são sempre sinalizados assincronamente (canais de comunicação serial ou paralelos, temporizadores e contadores programáveis, controladores de periféricos), relógios de tempo real, sinais de emergência, sinais de alguns sensores pulsados, etc.

e) Transferência via Acesso Direto à Memória (ADM)

Essa técnica é utilizada normalmente quando as informações a serem transferidas encontram-se dispostas sequencialmente na memória, e a quantidade de informações é muito grande. Esse método requer um circuito controlador de ADM, programado pela UCP, e que efetua automaticamente a transferência de dados entre a memória e o dispositivo periférico, ou até mesmo outra memória, sem a presença do microprocessador (que fica bloqueado durante a operação de ADM, no estado conhecido como ***hold***), tornando-a mais rápida (Figura 1.d). São exemplos de dispositivos periféricos que comumente se utilizam dessa técnica: controladores de disco, placas especiais de microcomputadores, canais de comunicação de alta velocidade, etc.

Figura 1 – Principais Técnicas de Transferência de Dados de Entrada e Saída



Fonte: Autoria própria

3. INTERRUPÇÃO

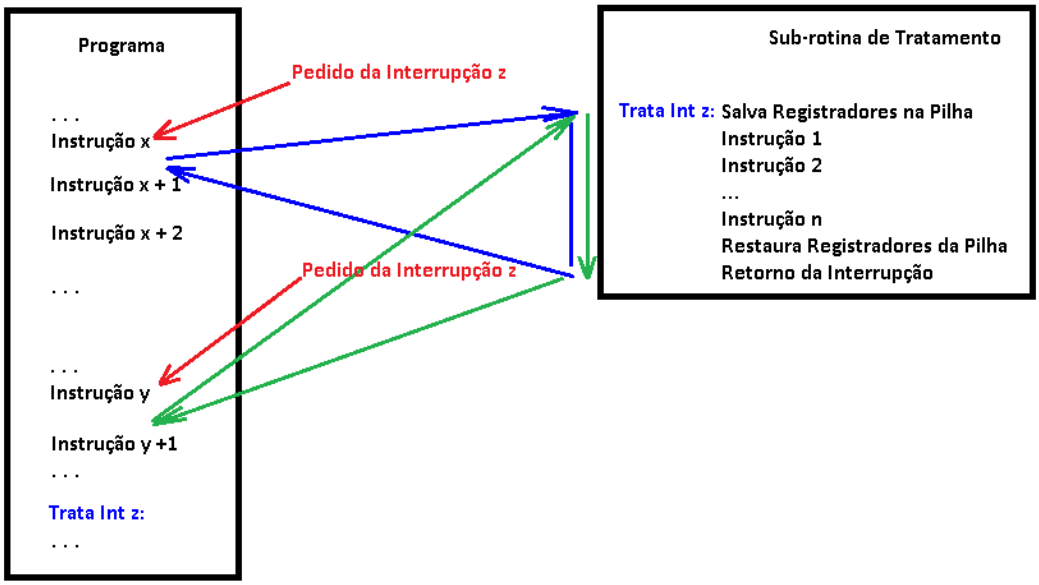
Como foi apresentado, as interrupções possibilitam um gerenciamento mais eficiente das operações de entrada e saída, e permitem melhor atender aos requisitos de aplicações em tempo real. As suas principais características são apresentadas nos próximos itens.

3.1. Características Básicas

a) Pedido e Reconhecimento

A requisição de interrupção pode ocorrer a qualquer momento (assincronamente), sendo ela indicada pela ativação de um ***flag*** pelo dispositivo periférico. A Figura 2 apresenta a representação do atendimento de interrupções habilitadas por um microprocessador.

Figura 2 – Mecanismo de Interrupções



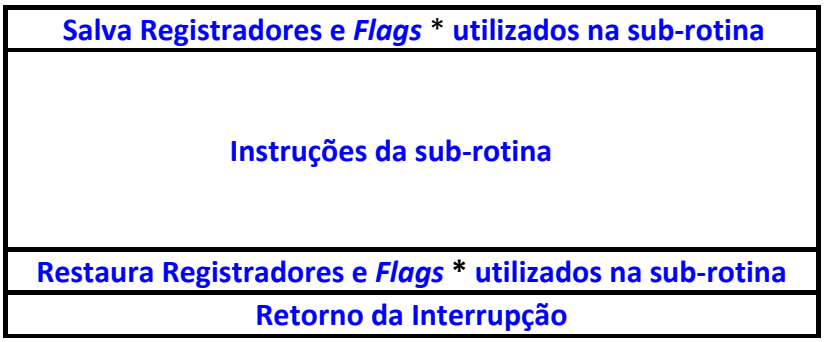
Fonte: Autoria própria

De forma geral, quando o microprocessador recebe um **pedido da interrupção**, sinalizado por meio de um sinal elétrico, e essa interrupção está **habilitada**, as seguintes ações são tomadas:

- reconhecimento do **pedido da interrupção**, pelo envio dos sinais de controle associados, e conclusão da execução da instrução corrente (instruções em execução tem que ser concluídas);
- salva na pilha o conteúdo dos registradores de interesse (contador de programa, status, etc.), e o endereço da próxima instrução do programa;
- obtém o endereço da sub-rotina de tratamento de interrupção;
- transfere o controle do programa para essa sub-rotina, que deverá ser encerrada por uma instrução de retorno ao programa que foi interrompido;
- ao término da execução desta rotina, o microprocessador desativa o ***flag*** de indicação de interrupção, e transfere o controle para a instrução seguinte ao ponto de interrupção do programa.
- restaura os registradores e o endereço da próxima instrução do programa que foram salvos;
- transfere a execução para o programa interrompido.

b) Sub-rotinas de Tratamento de Interrupções

As sub-rotinas de tratamento de interrupções devem ser construídas segundo o seguinte esquema:



* O MC 68000 salva automaticamente os *Flags* na pilha

O salvamento dos registradores e *flags* utilizados na sub-rotina é uma ação indispensável ao bom funcionamento do programa principal, evitando efeitos colaterais. Uma vez que a interrupção pode ocorrer a qualquer instante, ela pode, por exemplo, alterar um registrador que possuía alguma informação importante ao programa principal.

Em geral, as sub-rotinas devem ser curtas, com pouco processamento, evitando que outros pedidos de interrupção possam ocorrer antes do tratamento das anteriores, evitando avalanches e comprometam o processamento. Por exemplo, deve-se sinalizar, em uma variável, a ocorrência da interrupção e essa variável é consultada e limpa no programa principal que realiza o processamento correspondente. Deve-se evitar *loops* demorados, rotinas de espera de tempo grande, envio de mensagens, atualização de displays, etc.

c) Habilitação de Interrupções

Algumas aplicações de tempo real envolvem módulos de programas críticos que não podem ser interrompidos durante a sua execução. Algumas interrupções podem ter seu tratamento postergado, enquanto que outras necessitam de tratamento imediato (por exemplo, sinais de emergência, coleta de dados e alarmes). Em função disso, a maioria dos microprocessadores apresentam **interrupções mascaráveis** e **interrupções não-mascaráveis**. Através de instruções apropriadas o programa pode **habilitar** ou **desabilitar** uma interrupção mascarável, enquanto que a **não-mascarável** deverá ser **sempre atendida**. Muitos processadores apresentam uma de suas interrupções como sendo **não-mascarável**, de forma que sempre que ela ocorrer, o microprocessador deverá necessariamente atendê-la. Ela deve ser utilizada com muito cuidado, devendo ser alocada a eventos de muita importância, por exemplo, sinal de queda da alimentação do sistema, sinal de emergência, alarme, etc.

c) Tipos de Interrupções

As interrupções dos microprocessadores costumam ser de vários tipos, como: ***fixas***, ***vetoradas*** ou ***não vetoradas***, apresentadas a seguir.

- **Interrupções Fixas**

  Requerem um hardware relativamente simples para suportá-las. Um *flag* é ativado pela linha de interrupção, indicando a requisição. Caso apenas um dispositivo periférico esteja ligado a

essa linha, então o controle é transferido para uma posição fixa de memória, que possui a sub-rotina de tratamento da interrupção. Quando vários dispositivos são acoplados a uma mesma linha, cada um com sua própria sub-rotina de tratamento, o microprocessador tem que identificar o dispositivo periférico responsável pela interrupção. Para contornar essa dificuldade, pode-se utilizar múltiplas linhas, uma para cada interrupção, tendo-se uma posição fixa de memória para a colocação da sub-rotina de tratamento de cada uma. Entretanto, existe a necessidade de tantas linhas quantos forem os dispositivos periféricos que solicitam interrupção. Exemplos: interrupções RST5.5, RST6.5, RST7.5 e TRAP do microprocessador 8085 da Intel, e a interrupção NMI/ do Z80 da Zilog.

- **Interrupções Vetoradas**

  Este tipo exige que o dispositivo periférico que solicita interrupção se identifique. Essa identificação (uma espécie de ponteiro) é utilizada para a localização do endereço da sub-rotina de tratamento da interrupção em uma **tabela** na memória, localizada em uma região determinada da memória do microprocessador**,** chamada de **vetor de interrupções**. Exemplo: interrupções da família MC 68000.

- **Interrupções Não Vetoradas**

  Nesse tipo, o dispositivo periférico fornece ao microprocessador diretamente o endereço de sub-rotina de tratamento da interrupção (por exemplo, as interrupções do 8086/8088). Em alguns sistemas, o dispositivo periférico fornece apenas metade de endereço, estando a outra metade armazenada em um registrador do microprocessador, carregado na fase de inicialização do sistema (por exemplo, as interrupções do Z80 operando no modo 2). Em outros sistemas, o dispositivo periférico fornece uma instrução ao microprocessador, normalmente a instrução de chamada da sub-rotina de tratamento da interrupção (por exemplo, a interrupção INTR do 8085, e INT/, do Z80 operando no modo 0).

- **Prioridade no Atendimento de Interrupções**

  Em sistemas com mais de uma interrupção, existe a possibilidade de ocorrerem **pedidos de interrupção simultâneos**, devendo existir algum critério para a escolha de qual será atendida em primeiro lugar. Esse critério, normalmente por **prioridade**, pode ser implementado de duas formas: ***daisy chain*** ou **circuitos de prioridade**.

  - ***Daisy Chain***

    Nesse esquema, os dispositivos periféricos que podem solicitar interrupção são interligados, através de sinais de controle, em uma cadeia conhecida como ***daisy chain***, sendo o primeiro elemento ligado ao microprocessador, o segundo ligado ao primeiro e assim por diante. Em caso de pedido de interrupção, o microprocessador envia um sinal ao primeiro elemento; caso seja ele o autor do pedido, ele responde com o endereço da sub-rotina de tratamento; caso contrário, ele repassa o pedido ao próximo dispositivo

periférico, que repetirá o procedimento descrito. Dessa maneira, o primeiro elemento da cadeia deverá corresponder ao de maior prioridade, e assim sucessivamente. Esse esquema pode ser implementado, por exemplo, com o Z80 operando no modo 2.

- **Circuitos de Prioridade**

  Nesse esquema, um codificador de prioridades é utilizado (por exemplo, os circuitos **74x148**, de 8 entradas para 3 saídas, x = LS, HC, HCT), fornecendo em sua saída o código do pedido da interrupção de maior prioridade presente na entrada. Esse código pode ser inspecionado pelo microprocessador para descobrir qual interrupção atender. Muitas vezes são disponíveis esquemas para o mascaramento individual das interrupções, permitindo que uma interrupção alocada em uma linha de maior prioridade não seja atendida quando outra de menor prioridade ocorrer. Esse esquema pode ser implementado, por exemplo, com o 8085, o Z80 operando no modo 0, e a família MC 68000.

3.2. Interrupções na Família MC 68000

Cada família de microprocessadores apresenta formas específicas de implementação para as suas interrupções, podendo apresentar particularidades não encontradas em outras famílias. Recomenda-se, nesse ponto, que seja efetuada uma leitura atenta de [12] e [20], que apresenta as características das interrupções da família MC 68000. A seguir, serão resumidos os principais detalhes envolvidos.

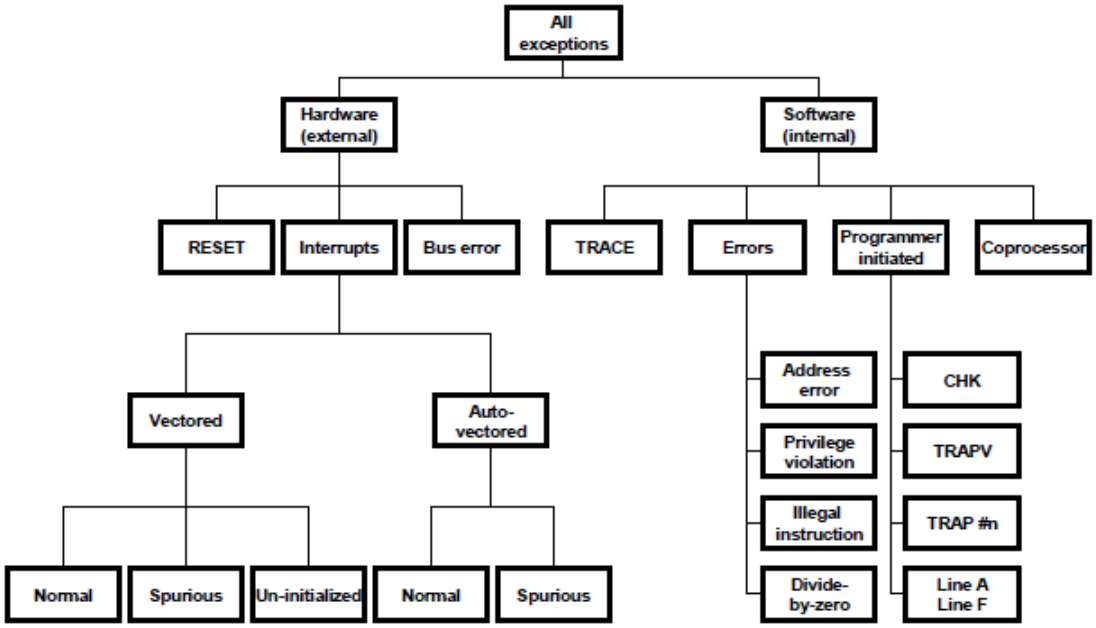
a) Exceções

Conforme introduzido na Experiência 5 [05], o MC 68000 utiliza o conceito de **Exceções**, que podem ser originadas tanto por software como por hardware [20]. A Figura 3 apresenta as diversas exceções consideradas no MC 68000. O processamento de uma exceção ocorrerá quando ocorrer alguma condição excepcional, que alterará o processamento normal de instruções. São exemplos de exceções, as originadas por evento:

- **Externos** – Reset, Erro de Barramento (*Bus Error* - causado por acesso a endereço de memória inexistente) e ocorrência de **interrupções**.
- **Internos** – Erro de Endereçamento (causado por acesso a uma *word* e *long word* em um endereço ímpar), Execução Passo-a-Passo (Trace), execução de instruções **TRAP, TRAPV (**gera exceção na ocorrência de *overflow***)** ou **CHK** (gera exceção caso o conteúdo de um registrador esteja fora da faixa permitida), tentativa de execução de instruções ilegais, tentativa de violação de privilégio, e Divisão por Zero (gerado pelas instruções **DIVS** e **DIVU**).

Nesse contexto, as interrupções se situam na categoria de exceções de hardware, originadas por eventos externos ao processador, manifestados por meio de sinais elétricos.

Figura 3 – Taxinomia da Exceção



Fonte: extraído de [20]

b) Grupos

As exceções do MC 68000 são classificadas em três grupos:

- **Grupo 0: Reset**, **Erro de Barramento** e **Erro de Endereçamento**. O processamento de exceção se inicia tão logo o evento seja reconhecido.
- **Grupo 1:** Trace, Interrupção, Instrução Ilegal, Violação de Privilégio. O processamento de exceção se inicia **antes** da próxima instrução.
- **Grupo 2: TRAP**, **TRAPV**, **CHK**, Divisão por Zero Processamento de Exceção. O processamento de exceção é disparado pela **execução da instrução**.

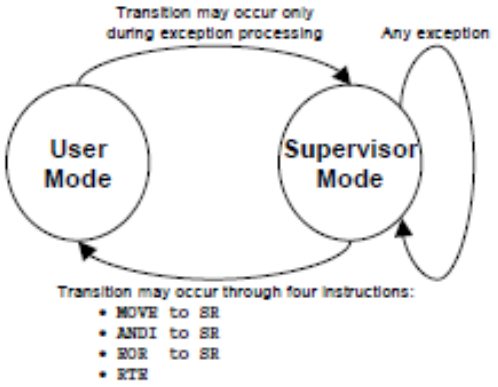
c) Modo Supervisor e Modo Usuário

O MC 68000 atua em dois modos: o **Supervisor** e o **Usuário** (Figura 4). As exceções são tratadas no modo **Supervisor** do MC 68000, que utiliza uma pilha própria (utiliza como ponteiro o **SSP** – *Supervisor Stack Pointer*, registrador A7'), diferente da pilha utilizada pelos programas executados no modo usuário (utiliza como ponteiro o **USP** – *User Stack Pointer*, registrador A7), além de contar com instruções privilegiadas, como:

Instruções que alteram o SR	`AND #data,SR`
	`OR #data,SR`
	`EOR #data,SR`
	`MOVE <ea>,SR`
	`MOVE An,SSP`
	`MOVE SSP,An`

Retorno de Exceção	`RTE`
Provoca um Reset nos dispositivos periféricos	`RESET`
Carrega um valor no SR e para o processador	`STOP`

Figura 4 – Modos de Operação do MC 68000



Fonte: extraído de [20]

d) Vetor de Exceções

O **Vetor de Exceções** do MC 68000 é apresentada na Tabela 1. Ela possui 256 posições de 4 bytes cada, totalizando 1024 bytes, ocupando os endereços de **$00000000** a **$000003FF**. Isso possibilita o armazenamento dos endereços (*long word*) das respectiva sub-rotinas de tratamento da exceção, ou *exception handler* (exceto as 8 primeiras posições destinadas a armazenar os valores iniciais do **PC** (que deverá ter o endereço da primeira instrução do programa) e **SSP** (endereço do ponteiro da pilha do supervisor). Cabe destacar que após o Reset, o processador opera no **modo Supervisor** (bit **S** do **SR** = 1). Os 4 primeiros endereços se encontram na área de programa (memória ROM), enquanto as demais na área de dados (memória RAM), ambas do Supervisor. Na Placa Experimental, as posições de 2 a 255 podem ser carregadas pelos programas do usuário.

O número do vetor é tal que quando multiplicado por 4 resulta no endereço da posição que possui o endereço (***long word***) da sub-rotina de seu tratamento.

e) Interrupções Vetoradas e Autovetoradas

Os microprocessadores da família MC 68000 podem tratar as interrupções de duas maneiras, como interrupções ***vetoradas*** ou como interrupções ***autovetoradas***, descritas a seguir:

- **Vetoradas**

  O microprocessador busca o número da posição no vetor de interrupções no dispositivo periférico que a gerou, que responde fornecendo o sinal **DTACK/**, e um **número de 8 bits**, que é usado para o cálculo da posição no vetor de interrupções, na região de **64** a **255** (**$40** a **$FF)**, correspondendo aos endereços de memória de **$100** a **$3FC**. O endereço é calculado pela fórmula:

  endereço do vetor = número do vetor x 4

Tabela 1 – Tabela do Vetor de Exceções do MC 68000

No. do Vetor	**Endereço (hexadecimal)**	**Atribuição**	**Observações**
0	$000	Reset: SSP inicial	Carrega o SSP e o PC obtidos das 8 primeiras posições de memória (única exceção que não tem retorno)
-	$004	Reset: PC inicial	
2	$008	Erro no Barramento (Bus Error)	Gerada na tentativa de acesso a endereço de memória inexistente
3	$00C	Erro de Endereçamento (Address Error)	Gerada na tentativa de acesso a uma ***word*** e ***long word*** em um endereço ímpar
4	$010	Instrução Ilegal	Gerada na tentativa e execução de um código de instrução inexistente
5	$014	Divisão por Zero	Gerada na tentativa de divisão por **0** pelas instruções **DIVS** e **DIVU**
6	$018	Instrução **CHK**	Dispara o processamento de exceção quando o conteúdo de um registrador se encontra fora da faixa permitida
7	$01C	Instrução **TRAPV**	Dispara o processamento de exceção na ocorrência de **Overflow**
8	$020	Violação de Privilégio	**bit S** de **SR = 0 (modo Usuário)** e houve uma tentativa de execução de instrução privilegiada
9	$024	Trace (Execução Passo-a-Passo) **bit T** de **SR = 1**	**bit T** de **SR = 1 (modo Trace)** gera uma exceção a cada instrução ser executada
10	$028	Emulação 1010	Códigos de instruções cujos 4 bits mais significativos sejam 1010: não implementadas no MC 68000, visando emular instruções não disponíveis (p.ex., ponto flutuante)
11	$02C	Emulação 1111	Similar ao anterior, podendo emular, p.ex. um coprocessador
12	$030	Reservado	
13	$034	Reservado	
14	$38	Reservado	
15	$03C	Interrupção Não-Inicializada	Trata um interrupção de um dispositivo não configurado para tal
16-23	$040-$05F	Reservado	
24	$060	Interrupção Espúria	Pedido recebido mas nenhum dispositivo respondeu
25	$064	Interrupção 1: Autovetor	
26	$068	Interrupção 2: Autovetor	
27	$06C	Interrupção 3: Autovetor	
28	$070	Interrupção 4: Autovetor	
29	**$074**	**Interrupção 5: Autovetor**	**Timer do MC 68230 na Placa Experimental SDC68k**
30	$078	Interrupção 6: Autovetor	
31	$07C	Interrupção 7: Autovetor	Habilitação: sempre habilitada (não mascarável)
32-47	**$080-$0BF**	**Instruções TRAP**	**Existem 16 instruções: TRAP 0 a TRAP 15**
48-63	$0C0-$0FF	Reservado	
64-255	$100-$3FF	Vetores de Interrupção do Usuário	Utilizado em função de cada projeto

Fonte: baseado em [20]

- **Autovetoradas**

  O microprocessador busca o número da **posição no vetor de exceções** no dispositivo periférico que gerou a interrupção, que responde fornecendo o sinal **VPA/**. Dessa forma, o microprocessador irá gerar o **número da posição no vetor de exceções**, de acordo com o **nível da interrupção** que foi solicitada (de **1** a **7**) codificado em 3 bits colocados nas linha de endereço **A1**, **A2** e **A3**. Muitos dispositivos periféricos da família do MC 68000, como o **MC 68230 (PI/T)** [19] permitem a programação das posições do vetor de interrupções, enviando automaticamente o valor programado quando do reconhecimento de alguma interrupção.

Caso nenhum dispositivo periférico responda ao reconhecimento de uma interrupção, deverá ser gerado o sinal **BERR/,** fazendo com que o microprocessador trate tal fato como uma **interrupção espúria**, buscando o endereço da rotina de tratamento na posição **24** (**$18**) do vetor de interrupções.

Durante o tratamento de uma interrupção, o seu nível fica automaticamente **inibido** até o seu término. As sub-rotinas de tratamento de interrupções devem prever a **preservação dos registradores por ela utilizados**, por se tratar de um módulo de programa ativado assincronamente, não se podendo prever, assim, a sua ocorrência. Dessa forma, no início de cada sub-rotina os registradores utilizados devem ser colocados na pilha, e retirados antes de seu retorno, que se dá através da instrução **RTE**.

f) Características dos Mecanismos de Interrupção

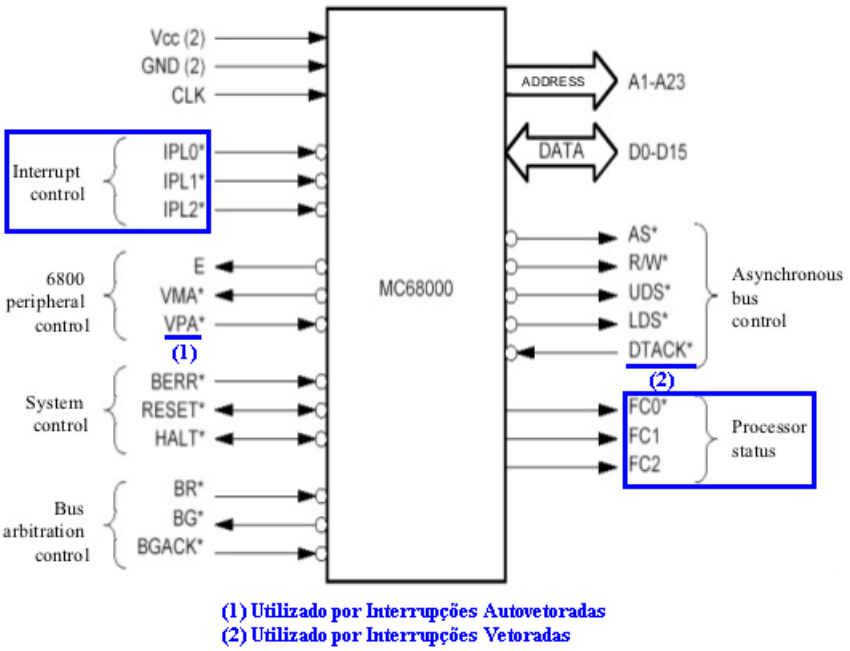
No MC 68000, três sinais de entrada são utilizados para o pedido de interrupção por dispositivos externos: **IPL0/**, **IPL1/** e **IPL2/** (Figura 5). Esses três sinais representam o **código** da interrupção pedida: **0, 0, 0 (nível 0)** indica **nenhum pedido**, enquanto **1 1, 1 (nível 7)** indica o pedido da interrupção **mais prioritária**. Os demais códigos apresentam prioridades decrescentes. Assim, tem-se 7 níveis de interrupções, com prioridades de 1 (menos prioritária) a 7 (mais prioritária).

Quando acionadas, essas linhas deverão ficar estáveis até o microprocessador sinalizar com o reconhecimento da interrupção (***interrupt acknowledge***), através dos sinais **FC0, FC1** e **FC2** simultaneamente em **nível 1**, o que garante que o microprocessador reconheceu a interrupção. O Nível 0 indica ausência de pedido. O microprocessador efetua o *de bounce* do sinal elétrico de pedido da interrupção por dois ciclos de *clock*. O nível 7 é sensível à borda do sinal de pedido.

Os bits **I0, I1** e **I2** do registrador de estado **SR** do MC 68000 (Figura 6) tem a função de **máscara de interrupções**, bloqueando ou não determinados níveis de interrupções. O **valor 0** nesses bits permite o reconhecimento de interrupções. O **nível 7 não permite o seu mascaramento** (interrupção **não-mascarável)**. Ocorrendo um pedido de interrupção que possua **prioridade superior** à da máscara de interrupção programada, a sub-rotina associada a esta interrupção irá se iniciar após a instrução em execução ser concluída. Caso contrário, o atendimento desse pedido será

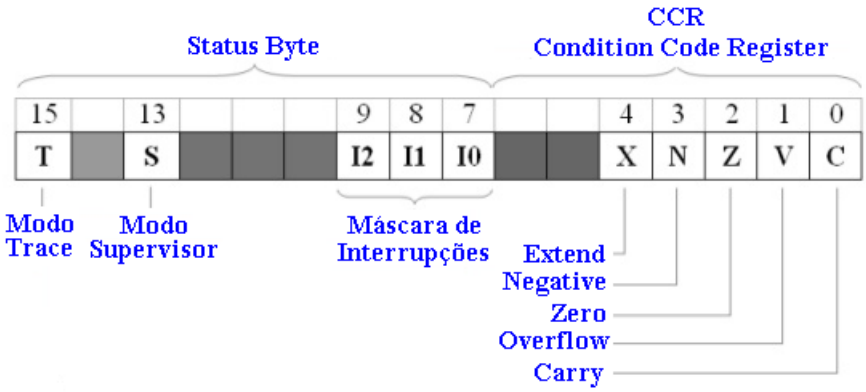
adiado até que a máscara de interrupções receba um valor que permita o atendimento dessa interrupção. Na Tabela 2 têm-se os valores que do Status Byte do SR e respectivos níveis de interrupções aceitos.

Figura 5 – Principais Sinais do MC 68000



Fonte: baseado em [20]

Figura 6 – Status Register (SR) do MC 68000



Fonte: baseado em [20]

Exemplo: se **I2, I1, I0 = 0, 1, 1 (3)**, os níveis de interrupção aceitas são:

- **1, 0, 0 (4), 1, 0, 1 (5), 1, 1, 0 (6) e 1, 1, 1 (7)**.

O microprocessador ignorará temporariamente os pedidos de interrupção de nível igual ou inferior ao mascarado, até que alterações na máscara venham permiti-las.

Tabela 2 – Valores do Status Byte do SR e respectivos níveis de Interrupções aceitos.

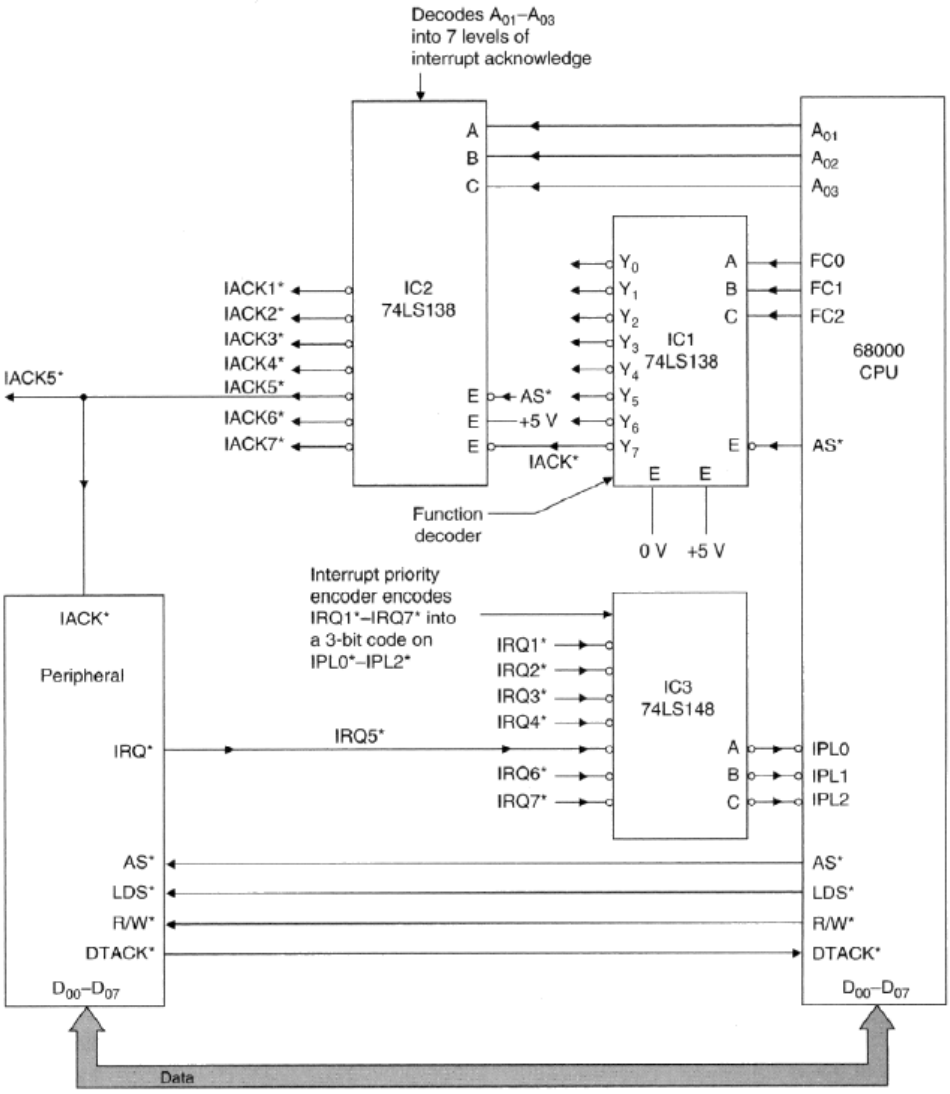
Valor do SR	T x S x x x I2 I2 I0	Níveis de Interrupções Aceitas
$20xx	0 0 1 0 0 0 0 0 1	1, 2, 3, 4, 5, 6, 7
$21xx	0 0 1 0 0 0 0 0 1	2, 3, 4, 5, 6, 7
$22xx	0 0 1 0 0 0 0 1 0	3, 4, 5, 6, 7
$23xx	0 0 1 0 0 0 0 1 1	4, 5, 6, 7
$24xx	0 0 1 0 0 0 1 0 0	5, 6, 7
$25xx	0 0 1 0 0 0 1 0 1	6, 7
$26xx	0 0 1 0 0 0 1 1 0	7

xx: Normalmente pode ser 00

Fonte: autores

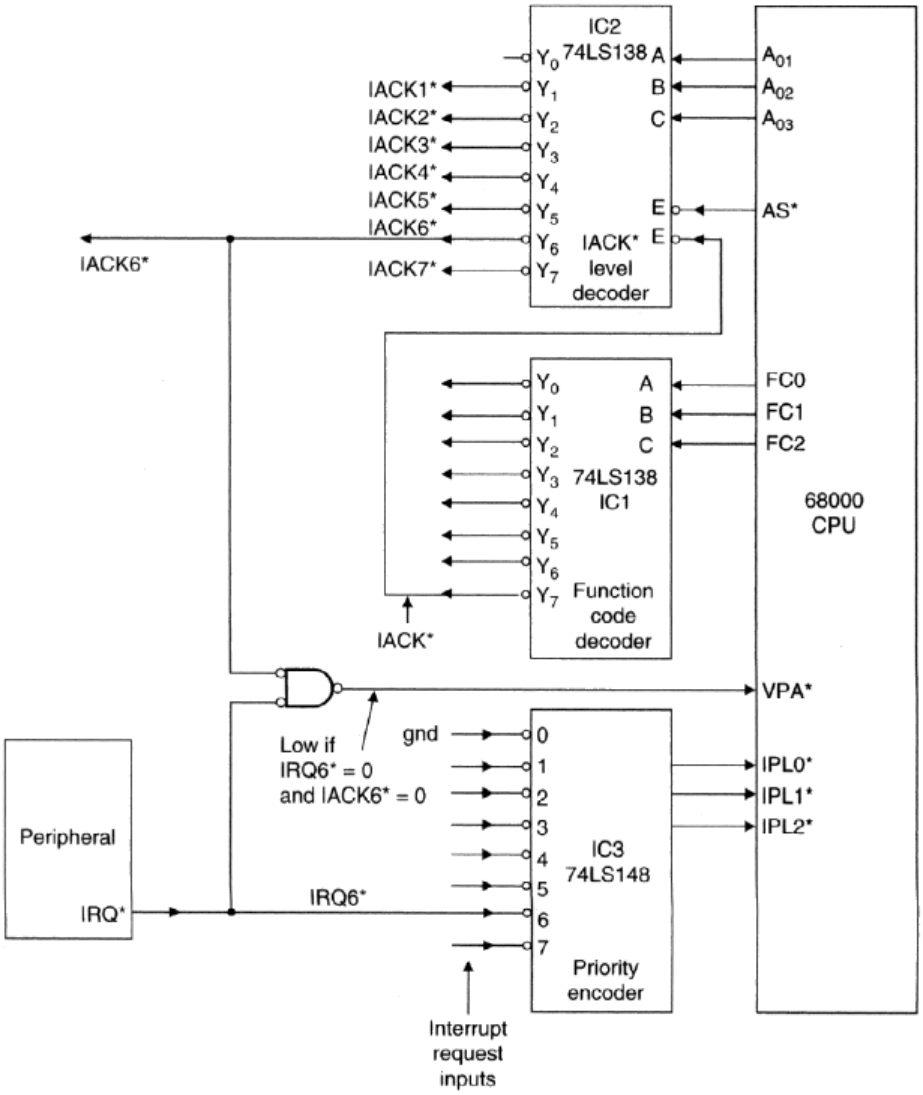
As Figuras 7 e 8 apresentam circuitos típicos que implementam o mecanismo para interrupções **Vetoradas** e **Autovetoradas**, respectivamente.

Figura 7 – Circuito típico para interrupções Vetoradas



Fonte: extraído de [20]

Figura 8 – Circuito típico para interrupções Autovetoradas



Fonte: extraído de [20]

g) Processamento de Exceções

Se o microprocessador receber um **pedido de interrupção** cujo código de prioridade seja **maior** do que o programado (ou seja, prioridade superior), considera-se que ocorreu uma interrupção habilitada, e um **processamento de exceção** é iniciado, disparando os seguintes passos:

- cópia do registrador de estado (**SR**) em um registrador interno temporário;
- **S = 1** e **T = 0** (ou seja, o microprocessador vai para o estado **Supervisor** e suspende a operação de **Trace**, e associa a máscara ao nível da interrupção;
- coloca o código do **nível** da interrupção atendida nas linhas **A1, A2** e **A3** de endereço, e obtém o número da posição no vetor de exceções do dispositivo periférico que gerou a interrupção. Para o caso de interrupções:

- **Vetoradas**, esse dispositivo coloca na via de dados os 8 bits do Número do Vetor de Exceções e ativa o sinal **DTACK/.**
- **Autovetoradas,** esse dispositivo ativa o sinal **VPA/**, e MC 68000 utiliza um número predefinido para cada nível recebido (conforme mostra a Tabela 1).

- coloca na pilha o conteúdo do **Contador de Programa** (**PC**) e do **Registrador de Estado** (**SR**) guardado internamente;
- retira o endereço da sub-rotina de tratamento da interrupção do vetor, e coloca-o no **Contador de Programa** (**PC**), efetuando, assim, o desvio para a sub-rotina de tratamento de interrupção.

h) Sub-rotinas de Tratamento de Exceções

As sub-rotinas de tratamento de exceções, ou *exception handler*, são similares a sub-rotinas de uso geral, com as seguintes diferenças:

- Como elas não são chamadas por instruções com **BSR** ou **CALL**, o respectivo endereço inicial é determinado a partir do número da exceção no Vetor de Exceções.
- Antes da sua chamada, tanto o **PC** (que contém o endereço de retorno ao programa principal) como o SR são colocados automaticamente na pilha do Supervisor.
- São sempre ativadas no modo Supervisor (bit **S** de **SR** é ativado sempre na ocorrência de exceções).
- A instrução de retorno é a **RTE** e não **RTS**, que são utilizadas apenas em sub-rotinas comuns.
- Na ocorrência de múltiplas exceções, a prioridade de cada uma determina a ordem de atendimento.

4. UTILIZAÇÃO DO TEMPORIZADOR DO MC 68230 COM INTERRUPÇÃO

Com o objetivo de avaliar em laboratório um tipo de interrupção do MC 68000, será utilizado o temporizador (Timer) do dispositivo **MC 68230 (PI/T)** que possui portas paralelas (A, B e C, já utilizadas em experiências anteriores), e um **temporizador** [19][20].

4.1. Características Básicas do Temporizador

a) Principais Características

O MC 68230 possui dois tipos de interface: três portas paralelas (A, B e C), e um temporizador programável, que permite, mediante configuração, a:

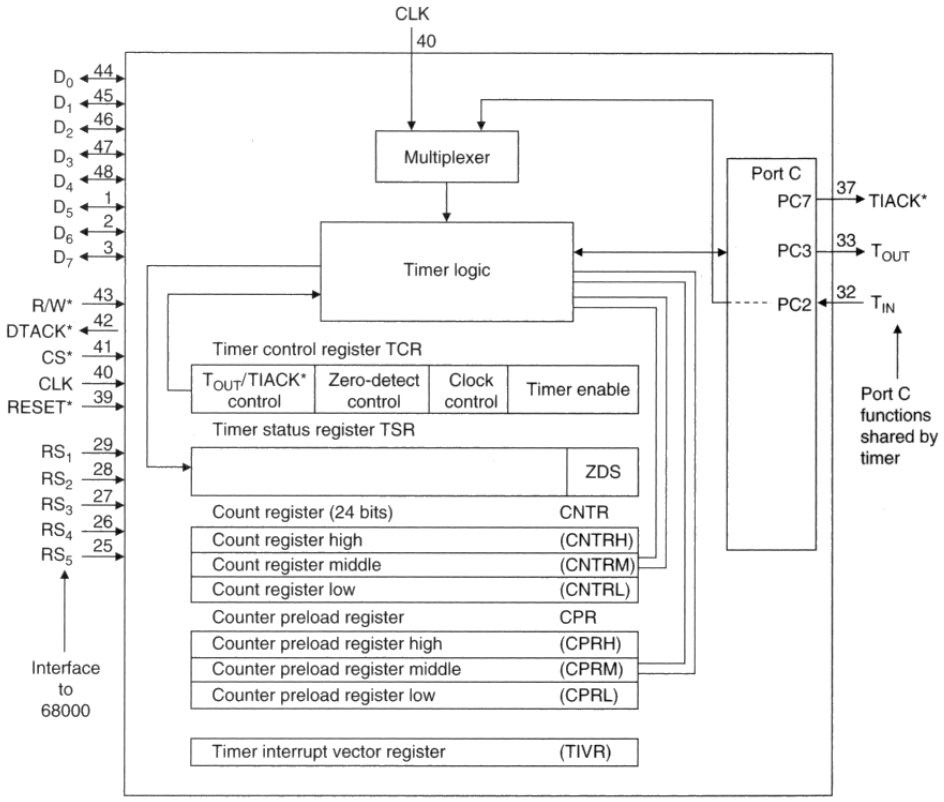
- geração de uma interrupção periódica;
- geração de uma forma de onda quadrada;
- geração de uma interrupção após o tempo limite;
- a medição do tempo transcorrido;
- implementação de circuito de vigilância (conhecidos como *watchdogs*).

Na Placa Experimental SBC68k, o **endereço base** dos 23 registradores (14 para as portas paralelas e 9 para o temporizador) do MC 68230 é **$FE8000**, que permite a programação por meio de escritas e leitura. Como o MC 68230 possui apenas 8 bits em sua via de dados, todos os registradores estão localizados em apenas uma das metades da via de dados, no caso parte da memória de **endereços ímpar**. Assim, a instrução **MOVEP** é utilizada para a transferência de bytes alternados na memória. O formato da instrução é:

MOVEP.size Di, d (Aj) ou **MOVEP.size d (Aj), Di**

A Figura 9 apresenta o diagrama em blocos do temporizador. Pode-se constatar que 3 pinos da **porta C** podem ser utilizados pelo temporizador.

Figura 9 – Diagrama em blocos do temporizador



Fonte: extraído de [19]

b) Registradores do Temporizador

Os registradores do temporizador são apresentados na Tabela 3. Os *offsets* correspondem ao deslocamento a ser somado ao endereço-base **$FE8000.** Em relação aos contadores (**CPR** e **CNTR**), como eles possuem 24 bits (3 bytes), é conveniente na programação utilizar o endereço anterior de cada um (que não tem função), possibilitando a escrita de 32 bits de uma vez (*long word*), sendo os 8 mais significativos iguais a 0.

Tabela 3 – Registradores do Temporizador do MC 68230

offset		sigla	nome	Função
$21		TCR	Timer Control Register	Determina os modos de operação do temporizador
$23		TIVR	Timer Interrupt Vector Register	Armazena o número do vetor de interrupção
$25	CPR	(**)	Counter Preload Register (endereço anterior)	Base dos registradores do contador de pré-carga (3 bytes)
$27		CPRH	Counter Preload Register High	byte mais significativo
$29		CPRM	Counter Preload Register Middle	byte intermediário
$2B		CPRL	Counter Preload Register Low	byte menos significativo
$2D	CNTR	(**)	Counter Register (endereço anterior)	Base dos registradores do contador (3 bytes). Decrementado a cada clock
$2F		CNTRH	Counter Register High (*)	byte mais significativo
$31		CNTRM	Counter Register Middle (*)	byte intermediário
$33		CNTRL	Counter Register Low (*)	byte menos significativo
$35		TSR	Timer Status Register	Determina o estado do temporizador

(*) só leitura; os demais são de escrita e leitura

(**) endereço base para a carga

Fonte: autores

A função de cada registrador é apresentada a seguir.

- **Timer Control Register – TCR**

Os bits do TCR são apresentados na Tabela 4 e descritos a seguir:

- **Ativação do** Temporizador (TCR0): liga (1) e desliga (0) o temporizador.
- **Controle do clock (TCR1-2): o** contador pode ser decrementado a partir de três sinais diferentes:
  - **TIN (PC2)**, a entrada de *clock* externo;
  - a divisão do CLK do MC 68000 por 32 (pre-scaler de 5 bits) e habilitado por **TIN (PC2)**,

- a divisão do CLK do MC 68000 por 32 (pre-scaler de 5 bits). Exemplo: o *clock* SBC68K é de 8 MHz (125×10^{-9}s), de modo que para gerar 1 segundo exigirá 250.000 transições do CLK.

- **Carga de Contador (TCR4):** quando o contador atinge 0, caso esse bit tenha valor **0**, é feita a carga automática do valor do contador de pré-carga; quando em **1**, a carga é feita com o valor máximo (**$00FFFFFF**).
- **Detecção de fim de Contagem (TCR5-7)**: as ações que podem ser realizadas ao fim de uma contagem são apresentadas na Tabela 4.

Tabela 4 – Timer Control Register – TCR do MC 68230

Mode	TCR7	TCR6	TCR5	TCR4	TCR3	TCR2	TCR1	TCR0
1	1	X	1	0	X	00 or 1X		1
2	0	1	X	0	X	00 or 1X		1
3	1	X	1	1	X	00 or 1X		1
4	0	0	X	1	X	0	0	1
5	0	0	X	1	X	0	X	1
6	1	X	1	1	X	0	1	1
	T_{OUT}/TIACK* control			ZD control	Not used	Clock control		Timer enable

Mode 1: Real-time clock
Mode 2: Square wave generator
Mode 3: Interrupt after timeout
Mode 4: Elapsed time measurement
Mode 5: Pulse counter
Mode 6: Period measurement

Fonte: extraído de [19]

- **Timer Status Register – TSR**

Apenas o seu bit 0 apresenta uma função, **Zero Detect Status** ou **ZDS**, que informa que a contagem 0 foi atingida. Para limpar o ZDS após uma detecção de zero, deve-se escrever o valor **1** nele (**não o valor 0!**).

c) Geração de Interrupções em Intervalos Periódicos

Resumidamente, o temporizador programável opera sob o controle do Registrador de Controle do Temporizador (TCR), e tem um Registrador de Estado (**TSR**). O *clock* por ele utilizado pode ter diversas origens, selecionadas pelos bits 1 e 2 do **TCR**. Normalmente, ambos os bits serão definidos para **0**, e o *clock* utilizado será o do MC 68000, que é 8 MHz, dividido por 32. Os bits 7-5 do registrador de controle controlam a forma como o temporizador opera na detecção do fim de contagem (bit do TSR, ZDS = 1).

A interrupção pode ser configurada como **autovetorada**. Para tanto, o pino de saída **TOUT (PC3)** se comporta como o pedido de interrupção ao MC 68000, enquanto que o pino de entrada **TIACK / (PC7)** é o sinal de reconhecimento dessa interrupção.

O temporizador é ativado colocando-se **1** no **bit 0** do **TCR**. Caso ele esteja ativado, o seu contador será decrementado a cada ciclo do *clock*; caso contrário manterá a última contagem.

Nos modos que utilizam a ativação de um *clock* externo, o temporizador diminui cada ciclo de *clock* apenas se o bit de ativação do temporizador estiver definido e o pino **PC2** estiver e nível lógico **1**.

O contador possui 24 bits, divididos em 3 registradores de 8 bits: **CNTRH**, **CNTRM** e **CNTRL**, respectivamente as partes mais significativa, intermediária e menos significativa. Quando a contagem atinge 0, o bit de detecção de zero **ZDS** (**bit 0 do TSR**) recebe valor **1**. Para apagar essa notificação, ele deve ser escrito com o valor **1** (não é 0!).

Após a contagem atingir 0, ela prosseguirá ou do valor máximo (**$00FFFFFF**), ou será carregado um valor a partir dos registradores de pré-carga **CPRH**, **CPRM** e **CPRL**. A escolha dentre esses dois modos é feita pelo **bit 4** do **TCR**: se ele for **1**, então o contador continua contando; caso contrário o contador é carregado com os valores de **CPRH, CPRH e CPRH**. O uso dos registradores de pré-carga permite que o contador conte, por exemplo, um determinado período.

O temporizador também tem a capacidade de executar outras ações quando a contagem atinge 0. Essas ações são controladas pelos **bits 5, 6** e **7** do **TCR**, e utiliza o **bit 3** da **porta C (PC3)** como saída, renomeado como **TOUT**, utilizado como um pedido de interrupção; e o **bit 7** da **porta C (PC7)** como entrada, renomeado como **TIACK/ (PC7)**.

Para a geração de interrupções em intervalos periódicos, o pino **TOUT** é **1** quando o contador atinge **0** e o **bit ZDS** é **1**. O pino **TOUT** pode ser usado para causar uma interrupção autovetorada. A sub-rotina de tratamento da interrupção **deve limpar o bit ZDS** para limpar o pedido da interrupção. Este modo é útil para aguardar um período fixo entre iterações de um *loop* ou **interromper o processador em intervalos regulares**.

d) Configurações da Placa Experimental para Interrupções em Intervalos Periódicos

A Placa Experimental SDC68k permite que o temporizador gere interrupções em intervalos periódicos. Para tanto deve-se fazer a seguinte configuração.

- Conectar o pinos **1** e **2** de **J6**: conecta o sinal **TIACK/ (PC7)** ao sinal de reconhecimento da interrupção de nível 5, fornecido por circuitos da placa.
- Conectar o pinos **7** e **8** de **J6**: conecta o sinal **TOUT (PC3)** na posição 5 do codificador de prioridade da placa (**interrupção autovetorada de nível 5**). Consultando-se a Tabela 1, tem-se que essa interrupção ocupa a posição **29** do Vetor de Exceções, que corresponde ao endereço **$74**.

5. REFERÊNCIAS

[01] HIRAKAWA, A.R.; CUGNASCA, C.E. **PCS 3432 – Laboratório de Processadores, Experiência 1 - Familiarização com a Placa Experimental do Microprocessador MC 68000**. PCS - EPUSP, 2017. (*)

[02] HIRAKAWA, A.R.; CUGNASCA, C.E. **PCS 3432 – Laboratório de Processadores, Experiência 2 - Programação em Linguagem Assembly para a Placa Experimental do Microprocessador MC 68000.** PCS - EPUSP, 2017. (*)

[03] HIRAKAWA, A.R.; CUGNASCA, C.E. **PCS 3432 – Laboratório de Processadores, Experiência 3 - Ferramentas Para o Desenvolvimento de Software: Linguagem Assembly e Simulador.** PCS - EPUSP, 2017. (*)

[04] HIRAKAWA, A.R.; CUGNASCA, C.E. **PCS 3432 – Laboratório de Processadores, Experiência 4: Periféricos de Entrada e Saída: Acionamento de Motor de Passo Utilizando a Linguagem C.** PCS - EPUSP, 2017. (*)

[05] HIRAKAWA, A.R.; CUGNASCA, C.E. **PCS 3432 – Laboratório de Processadores, Experiência 5: Processamento de Exceções na Placa Experimental para o Uso de Rotinas Utilitárias.** PCS - EPUSP, 2017. (*)

[06] HIRAKAWA, A.R.; CUGNASCA, C.E. **PCS 3432 – Laboratório de Processadores, Experiência 6: Conexão de Módulo Display de Cristal Líquido na Placa Experimental.** PCS - EPUSP, 2017. (*)

[07] KINOSHITA, J.; CUGNASCA, C.E.; HIRAKAWA, A.R. **PCS 2498 – Laboratório de Processadores II, Experiência 5: Interrupções.** PCS - EPUSP, 2001.

[08] ARNEWSH INC. **SBC68K-User´s Manual**, Revision 1.1, Arnewsh Inc. C.O., USA, 1990. (*)

[09] ARNEWSH INC. **SBC68K - 68000 Single Board Computer User Manual**, Arnewsh Inc. C.O., USA, 1990. (*)

[10] MOTOROLA. **M68000 16/32-Bit Microprocessor Programmer's - Reference Manual**. N.J., EUA, Prentice Hall, 4a. edição, 1984. 218p. (*)

[11] MOTOROLA. **M68000 Family Reference**. EUA, 1988. (*)

[12] MOTOROLA. **M68000 8-/16-/32-Bit Microprocessors User's Manual**. N.J., EUA, 9ª edição, 1993. (*)

[13] MOTOROLA. **M68000 Family Programmer's Reference Manual (Part 1)**, N.J., EUA, 1992. (*)

[14] MOTOROLA. **M68000 Family Programmer's Reference Manual (Part 2)**, N.J., EUA, 1992. (*)

[15] FONDSE, P. J. **68000/020 Integrated Development Environment**, v 3.30, 2015. <http://home.kpn.nl/pj.fondse/ide68k/>

[16] FONDSE, P. J. **IDE68K-getting_started**, v 3.30, 2015. <http://home.kpn.nl/pj.fondse/ide68k/>

[17] FONDSE, P. J. **IDE68k Help**, help do aplicativo, v 3.30, 2015.

[18] ALFACOM. **Módulos Multi-Matrix.** Manual de Utilização. (*)

[19] MOTOROLA. **MC68230 - Peripheral Interface and Timer (PI/T)**. Motorola Inc. (*)

[20] CLEMENTS, A. **Microprocessor System Design**, 2nd Edition, 1992. (**)

(*) Disponíveis no formato pdf no site da disciplina.

6. PARTE PRÁTICA

A parte prática prevê duas partes, envolvendo a avaliação de interrupções tanto no simulador do IDE68k, como na Placa Experimental SBC68k.

O planejamento envolve a modificação e elaboração de programas com a respectiva documentação, que deverá incluir:

- **Diagramas em Blocos** para programas elaborados em linguagem *Assembly*.
- **Diagramas de Nassi-Shneiderman** (**NSD**) para programas elaborados em linguagem C. Utilizar os Programas-Exemplo como referência para os demais programas a serem elaborados.
- Incluir **comentários** em todos os programas.

Respostas às perguntas propostas deverão também ser incluídas no planejamento.

As etapas a seguir deverão ser seguidas para a experiência.

6.1. 1ª Etapa: Parte 1 - Programa-Exemplo do Apêndice I para Teste no Simulador

a) Estudar o Programa-Exemplo elaborado em linguagem C, identificando o que ele possui especificamente para suportar o recebimento e tratamento de interrupções.

b) Testar o programa para todos os valores de **SR** apresentados na **Tabela 2**, iniciando pelo valor **$2000**. Colocar no relatório a diferença de comportamento para cada valor.

c) Explicar como funcionam as diversas funções que o programa utiliza:

`timer, show7seg showslider, show7seg, showleds, showswitches, showbar, clear7seg, write7seg.`

d) Explicar como as interrupções são consideradas no programa, o Vetor de Exceções e as rotinas de tratamento.

e) Itens Específicos para cada Turma

Turma 1: Alterar o programa para que quando a chave 0 for ligada o contador congele a contagem, retomando-a quando ela for desligada.

Turma 2: Alterar o programa para que no início seja digitado um valor de contagem pelo terminal que quando atingida, envie uma mensagem ao terminal.

6.2. 2ª Etapa – Parte 2 - Programa-Exemplo do Apêndice II para Teste na Placa Experimental

a) Estudar o Programa-Exemplo elaborado em linguagem Assembly, identificando o que ele possui especificamente para suportar o recebimento e tratamento de interrupções.

b) Identificar como foi feita a habilitação da interrupção.

c) Alterar o intervalo de tempo entre interrupções para: (i) dobrar o intervalo; (ii) diminuir pela metade o intervalo, anotando os resultados.

d) Itens Específicos para cada Turma

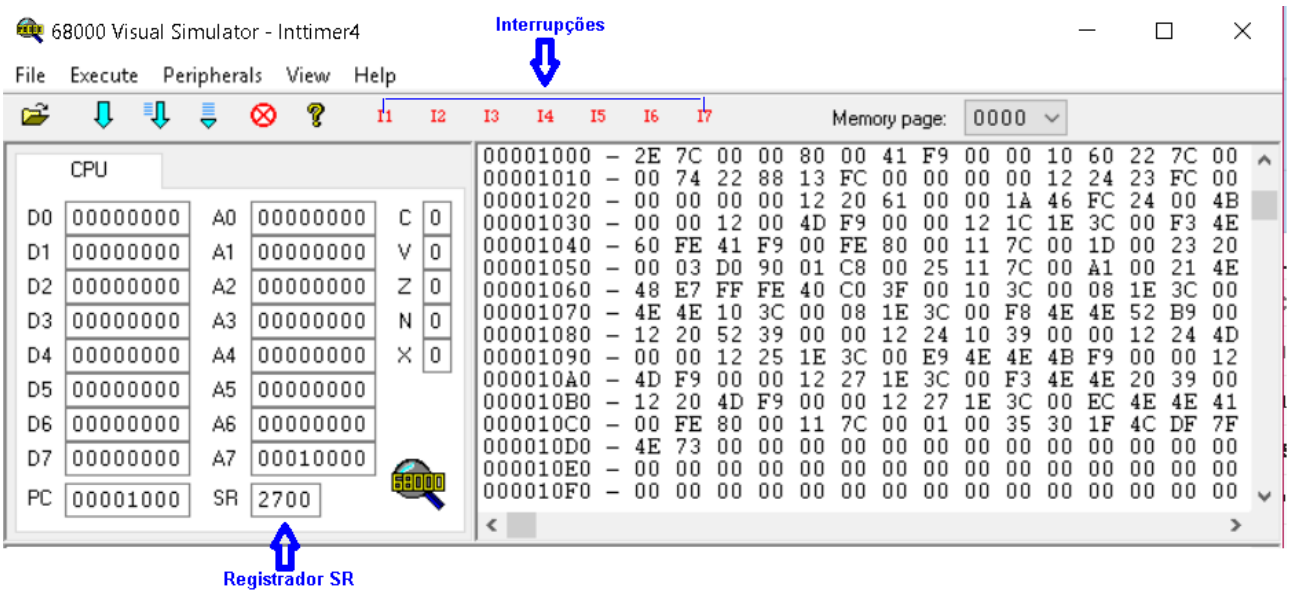
Turma 1: Elaborar um programa que apresente um relógio no módulo LCD, com horas, minutos e segundos. O ajuste do valor deverá ser feito por meio de comando no terminal (por exemplo, exibir um menu de opções: 1 para alterar horas e 2 para alterar minutos, 3 para alterar segundos, 4 para carregar os novos valores). A cada segundo um *bip* deverá ser enviado.

Turma 2: Elaborar um programa que implemente um cronômetro semelhante ao simulado na Parte 1, porém ele deverá ser decrescente. Os controles do cronômetro deverão ser feitos digitando-se comandos no terminal. Quando a contagem atingir 0, o display deverá piscar e o bip deverá ser ligado/desligado, ambos a uma taxa de 1 Hz.

APÊNDICE I – PROGRAMA EXEMPLO EM LINGUAGEM C PARA TESTE DE INTERRUPÇÕES NO SIMULADOR DO IDE68K

O simulador do IDE68k permite que interrupções sejam provocadas pelo usuário. Para tanto, ele deve clicar com o mouse sobre a interrupção desejada (**I1** a **I7**), de acordo com a Figura 10. Como o programa foi elaborado em linguagem C, o registrador **SR** não pode ser alterado nessa linguagem. Para determinar o nível de interrupção que deve ser aceito, de acordo com a Tabela 2, **antes da execução do programa** esse registrador deverá receber um valor conveniente. Por padrão o valor que ele recebe é **$2700**, desabilitando todos os níveis de interrupção exceto o nível 7, que é não mascarável.

Figura 10 – Interrupções e Registrador SR no Simulador IDE68k



Fonte: extraído de [16][17]

Em linhas gerais o programa implementa um **contador progressivo** utilizando diversos recursos, que podem ser inseridos por meio da instrução de exceção **TRAP 15**, utilizada em experiências anteriores. São eles:

- Display;
- Botões e LEDs;
- Slider;
- Barra de sinalização (bar);
- Timer;
- Terminal.

As principais características do programa são:

- **I5**: inicia/para contagem.
- **I6**: zera contagem.
- **I7**: termina o programa.
- **I1**: lê chave 1 e exibe seu valor no LED correspondente.
- **I2**: lê chave 2 e exibe seu valor no LED correspondente.
- **I3**: lê chave 3 e exibe seu valor no LED correspondente.
- **I4**: lê valor do *slider* e envia uma mensagem com o seu valor (0 a 255).

Por utilizar diversos recursos, como interrupções e dispositivos de entrada e saída, alguns arquivos devem fazer parte do projeto. Caso o nome do programa seja **`ex07ex01.c`**, deve-se criar um projeto, por exemplo, de nome **`ex07ex01.prj`**, com os seguintes arquivos (supondo que **`ex07ex01.c`** esteja na pasta **`Examples`** do Ide68k e a instalação em **`C:\Ide68k:`**

```
C:\Ide68k\Lib\cstart.asm
C:\Ide68k\Examples\ ex07ex01.c
C:\Ide68k\Lib\std68k.lib
```

A ordem dos arquivos no projeto não deve ser alterada. Recomenda-se consultar a documentação do IDE68k e os seus exemplos de programas para o melhor entendimento de Programa-Exemplo. Embora ele possa ser testado em qualquer um dos modos, o mais recomendado é o **Run mode**.

```
// Experiencia 07 - Exemplo: Cronometro e Teste de Interrupçoes
// Para executar no simulador IDE68k
// Autor: Carlos E. Cugnasca
// I5: start/stop; I6: zera; I7: sai do programa
// I1, I2, I3: mostra no LED o estado da chave
// I4: le valor do slider e o envia ao terminal
// Cada interrupçao envia uma mensagem ao terminal
//
// Interrupçoes - autovetores
void *int1vec = 0x0064; // int 1
void *int2vec = 0x0068; // int 2
void *int3vec = 0x006C; // int 3
void *int4vec = 0x0070; // int 4
void *int5vec = 0x0074; // int 5
void *int6vec = 0x0078; // int 6
void *int7vec = 0x007C; // int 7

// Ponteiros para os dispositivos de E/S
unsigned short *const display = (unsigned short *) 0xE010; // display[0]: digito + a esquerda
unsigned long *const timer = (unsigned long *) 0xE040;     // timer
// padrao de bits para o display de 7 segmentos: 0 - 9
unsigned short const bitpat[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

unsigned char *leds     = 0xE003; // end. do array de LEDs
unsigned char *bar      = 0xE007; // end. do display de barra
unsigned char *switches = 0xE001; // endereço das chaves
unsigned char *slider   = 0xE005; // endereço das chaves
unsigned char chaveaux, estadochaves, conta, para, zera;
unsigned long counter = 0;
unsigned long aux, aux1, posslider;

char mens     [] = "Ocorreu interrupcao ";
char mens4    [] = "  Valor do Slider: ";
char mens5    [] = "  Inicia/Para";
char mens6    [] = "  Zera";
char mens7    [] = "  Aborta o Programa";
char mensCRLF [] = "\r\n";

// SR so' pode ser alterado no modo supervisor:
// nivel da mascara de interrupcoes deve ser
// colocado antes da execucao do programa
// $2000: habilita todas ... $2600: apenas I7 (nao masc.)

void delay(void){  //atraso
  int i = 0;
  do {++i;}
  while (i < 30000);}

void showslider(void){  // mostra slider
  _trap(15);
  _word(33);}

void show7seg(void){  // mostra display 7-seg
  _trap(15);
  _word(35);}

void showleds(void){  // mostra LEDs
  _trap(15);
  _word(31);}

void showswitches(void){ // mostra chaves
  _trap(15);
  _word(32);}

void showbar(void){  // mostra bar
  _trap(15);
  _word(34);}

void clear7seg(void){  // limpa display 7-seg
  int i;
  for (i = 0; i < 3; i++) display[i] = 0;  // limpa los. 3 dig.
    display[3] = bitpat[0];}               // ultimo dig.= '0'

void write7seg(long n, int i){  // escreve no display 7-seg (recursiva)
  if (n > 9) write7seg(n / 10, i - 1);
      display[i] = bitpat[n % 10];}
```

```
void contar (void){  // realiza contagem
  long cntr = *timer / 10;
  (*bar)++;               // incrementa display BAR
  delay();                // temporiza
  if (counter != cntr){  // timer mudou
      counter = cntr;
      write7seg(counter, 3);}}

void CRLF (void){  // envia CR e LF ao terminal
  _A0 = mensCRLF;  // A0: ponteiro para a string
  _trap(15);
  _word(7);}      // system call PRTSTR

void print (long num){ // envia numero ao terminal
  _D0 = num; // D0: numero com sinal para imprimir
  _trap(15);
  _word(5);} // system call PRTNUM

void envmen (long ender){  // envia mensagem
  _A0 = ender;  // A0: ponteiro para a string
  _trap(15);
  _word(7);}    // system call PRTSTR

interrupt void int1proc(void){ // le/atualiza so estado chave 1
  chaveaux = *switches & 0x02;
  estadochaves = ((estadochaves & 0xFD)|chaveaux);
  envmen (mens);
  print (1);
  CRLF ();}

interrupt void int2proc(void){ // le/atualiza so estado chave 2
  chaveaux = *switches & 0x04;
  estadochaves = ((estadochaves & 0xFB)|chaveaux);
  envmen (mens);
  print (2);
  CRLF ();}  // CR e LF

interrupt void int3proc(void){  // le/atualiza so estado chave 3
  chaveaux = *switches & 0x08;
  estadochaves = ((estadochaves & 0xF7)|chaveaux);
  envmen (mens);
  print (3);
  CRLF ();}  // CR e LF

interrupt void int4proc(void){ // envia valor slider ao terminal
  envmen (mens);
  print (4);
  envmen (mens4);
  posslider = *slider;
  _D0 = posslider; // D0: numero com sinal para imprimir
  _trap(15);
  _word(5); // system call PRTNUM
  CRLF ();} // CR e LF

interrupt void int5proc(void){ // inicia/para/continua contagem
  envmen (mens);
  print (5);
  envmen (mens5);
  CRLF ();    // CR e LF
  if (conta == 0){  // esta parado, pedido de contagem
      conta = 1;}
  if (conta == 2){  // contando, pedido de parada
      conta = 3;}
  if (conta == 4){  // contando, pedido de parada
      conta = 1;}}

interrupt void int6proc(void){ // zera contagem
  envmen (mens);
  print (6);
  envmen (mens6);
  CRLF ();     // CR e LF
  zera = 1;}

interrupt void int7proc(void){ // termina o programa
  envmen (mens);
  print (7);
  envmen (mens7);
```

```
  CRLF ();
  _trap(15);  // aborta o Programa
  _word(0);}  // system call EXIT

// Programa Principal

void main(void){
// carrega enderecos das sub-rotinas de trat. int. no vetor
  *int1vec = int1proc;
  *int2vec = int2proc;
  *int3vec = int3proc;
  *int4vec = int4proc;
  *int5vec = int5proc;
  *int6vec = int6proc;
  *int7vec = int7proc;

  showslider();   // mostra slider
  showleds();     // mostra LEDs
  showswitches(); // mostra chaves
  showbar();      // mostra bar
  show7seg();     // mostra display 7-seg
  clear7seg();    // limpa display 7-seg

  conta = 1;      // estado do contador
  zera = 0;       // zerar contador
  *timer = 0;     // inicializa timer
  counter = 0;    // valor do coontador
  aux, aux1 = 0;  // variaveis auxiliares

  for (;;){
    *leds = estadochaves; // atualiza estado das chaves
    if (zera == 1){  // pedido para zerar:
       zera = 0;     // limpa pedido de zerar
       *timer = 0;   // zera timer
       counter = 0;  // zera contador
       aux = 0;
       aux1 = 0;
       write7seg(counter, 3); // atualiza display com 0s
       clear7seg();}          // limpa display
       else{
         if (conta == 1){ // inicio de parada
             conta = 2;   // estado = parado
             aux = *timer;   // salva valor do timer
             aux1 = counter;} // salva valor do contador
         if (conta == 2){ // parado
            }             // nada faz
         if (conta == 3){ // voltando a contar contar
             conta = 4;   // estado = contado
             *timer = aux;   // restaura valor do timer
             counter = aux1; // restaura valor do contador
             contar();}      // contar
         if (conta == 4){ // contando
             contar();}   // contar
         }
       if (counter == 10000) break; // para apos contar ate 9999
  }
}
```

APÊNDICE II – PROGRAMA EXEMPLO EM LINGUAGEM ASSEMBLY PARA TESTE DE INTERRUPÇÕES NA PLACA EXPERIMENTAL SBC68K

Esse programa gera uma interrupção periódica de 1 segundo, gerada a partir do **temporizador** do dispositivo MC 68230. A cada segundo um *bip* é emitido. Esse *bip* foi implementado por meio de um *buzzer*, ligado ao **bit 4** da **Porta B** do MC 68230.

```
; Experiencia 07 - Exemplo: Interrupçao do Timer do MC 68230
; Para executar na Placa Experimental SDC68k
; Author: Carlos E. Cugnasca
; Programa que aguarda interrupçao periodica do Timer,
; enviando uma mensagem ao terminal a cada interrupção,
; exibindo o n. de interrupções ocorrido (8 bits em hexa)

; Funcoes TRAP #14: n. de cada uma deve ser colocado em d7
OUTCH     equ     248               ; envia 1 caractere p/ a tela
OUTPUT    equ     243               ; envia 1 string p/ a tela
PNT2HX    equ     233               ; converte 2 digitos hexa(D0) p/ ASCII(A6)=pont.
BACKSPACE equ     08                ; codigo ASCII do Backspace

; Registradores da Porta B do MC 68230
PGCR      equ     $FE8001           ; registrador de controle geral
PBCR      equ     $FE800F           ; Porta B - registrador de controle
PBDDR     equ     $FE8007           ; Porta B - registrador de direção de dados
PBDR      equ     $FE8013           ; Porta B - registrador de dados

; Registradores do Timer do MC 68230
timevec   equ     $1D               ; Placa SDC68k: interrup. timer = nível 5
                                    ; posição 29D ($1D) na tabela
PIT       equ     $FE8000           ; endereço base da PI/T na SBC68K
TCR       equ     $21               ; offset p/ o reg. contr. timer
TIVR      equ     $23               ; offset p/ o reg. do vetor interr. timer
CPR       equ     $25               ; offset p/ o reg. de precarga do contador
TSR       equ     $35               ; offset p/ o reg. status timer
TIME      equ     $4*timevec        ; posiçao da sub-rotina trat. inter. no vetor interr.

; tempo entre interrupçoes TINT=(CLK/32)/CONTAGEM = 0,25 MHz, CLK = 8MHz
; PARA TINT = 1Hz, CONTAGEM = 0,25M/1 = 250000 = $0003D090
CONTAGEM  equ     250000            ; valor da divisao para 1Hz

INIPILHA  equ     $007FFF+1         ; inicio da pilha: ultima posicao da RAM + 1

BS        macro                     ; envia 1 backspace ao terminal
          move.b  #BACKSPACE,D0
          move.b  #OUTCH,D7
          trap    #14
          endm

; PROGRAMA PRINCIPAL

          org     $1000

PROGR     move.l  #INIPILHA,a7      ; carrega base da pilha
          lea.l   TRATINT,a0        ; carrega end. rotina trat. interrupcao
          move.l  #TIME,a1          ; carrega a posicao do vetor de exceçoes
          move.l  a0,(a1)           ; carrega no vetor o end. da rotina trat. int.
          move.b  #$0,CONINT        ; inicializa contador (8 bits) de int.
          bsr     PTIMER            ; programa timer e dispara contagem
          move.w  #$2400,sr         ; habilita nivel interrupcao > 4 e limpa flags
          lea.l   mensg1,a5         ; a5 aponta para o inicio da mensagem
          lea.l   mensg,a6          ; a6 aponta para o fim da mensagem +1
          move.b  #OUTPUT,d7        ; envia mensagem inicial para tela
          trap    #14
          bsr     INICPIB           ; inicia a Porta B (bip)
VOLTA     bra     VOLTA             ; loop infinito, aguardando interrupçao

; Sub-rotina de programaçao do Timer
PTIMER    lea     PIT,a0            ; carrega endereço base da PI/T
          move.b  #timevec,TIVR(a0) ; carrega a posição no vetor de exceções
          move.l  #CONTAGEM,d0      ; carrega contagem 24 bits p/ div. clock
          movep.l d0,CPR(a0)        ; nos bytes CPR (H, M, e L)
          move.b  #%10100001,TCR(a0) ; Programa TCR: modo 0, CLK/32, liga timer
          rts
```

```
; Sub-rotina de tratamento da interrupçao de nivel 5 do Timer
TRATINT   movem.l d0-d7/a0-a6,-(a7)  ; salva registadores na pilha
          not.b   BIP                ; alterna bits da Porta B,
          move.b  BIP,PBDR           ; gerando o som de bip
          BS                         ; envia 2 backspaces ao terminal
          BS
          addq.b  #1,CONINT          ; incrementa contador (8 bits)
          move.b  CONINT,d0          ; converte contador para 2 dig. ASCII
          lea     CONINTA,a6
          move.b  #PNT2HX,d7
          trap    #14
          lea     CONINTA,a5         ; envia contador (8 bits) ASCII
          lea     FIM,a6             ; ao terminal (em hexa)
          move.b  #OUTPUT,d7
          trap    #14
          lea     PIT,a0             ; limpa pedido de interrupção
          move.b  #1,TSR(a0)         ; para isso, escreve 1 em TSR
          movem.l (a7)+,d0-d7/a0-a6  ; restaura registadores usados da pilha
          rte                        ; retorno de excecao (nao e´ RTS !)

INICPIB   move.b  #$30,PGCR          ; programa modo do MC 68230
          move.b  #$A0,PBCR          ; programa submodo da Porta B
          move.b  #$FF,PBDDR         ; programa direção da Porta B (saída)
          rts

; Area de armazenamento de mensagens
mensg1    dc.b    'Interrupcoes ocorridas: $  '
mensg

; Area de variaveis
          org     $2000

BIP       ds.b    1                  ; variável do bip (toogle)
CONINT    ds.b    1                  ; variavel incrementada a cada int. (hexa)
CONINTA   ds.b    2                  ; variavel incrementada a cada int. (ASCII)
FIM       ds.b    1
```

ANEXO I - MODELO DE RELATÓRIO E ITENS EXPERIMENTAIS

Esse anexo apresenta a parte experimental que deverá ser realizada por cada grupo, constituído por **dois alunos** (eventualmente um grupo possuirá três alunos, no caso de número ímpar de alunos na turma).

Apenas os nomes dos alunos presentes deverão constar do Relatório.

Diversas atividades previstas no **Modelo de Relatório** de cada experiência deverão ser realizadas **antes da aula**, por **todos** os componentes do grupo, que deverão se reunir e elaborar o **Planejamento da Experiência**, que é o Modelo de Relatório com os devidos itens preenchidos ou realizados (algoritmos, documentação, especificação e códigos de programas, etc). No caso de programas, arquivos específicos com **programas-fonte** deverão também ser trazidos.

O **Planejamento da Experiência** não precisa ser impresso, porém os arquivos correspondentes devem ser trazidos **por cada componente do grupo**. Cada um deverá saber explicar e justificar o que foi elaborado pelo grupo, **não sendo aceita a divisão de tarefas**, com cada um apenas conhecendo a parte que realizou. Essa explicação será exigida pelo professor.

Todos os arquivos deverão ser trazidos em ***pen drives*** (mais de uma cópia), além de cópias de segurança em espaços de armazenamento na nuvem (por exemplo, GDrive, Dropbox, OneNote, etc).

Grupos sem planejamento (por qualquer motivo) ou que não conseguirem explicá-lo ao professor inviabilizarão a realização da experiência, com prejuízo na respectiva nota.

Na página seguinte segue a cópia do **Modelo de Relatório**, cujo **arquivo .doc** se encontra no site da disciplina.

Cada grupo realizará a experiência em uma das bancadas do laboratório, numeradas de **0** a **7**. Esse número passará a ser o **número do grupo** até o final do semestre, e deverá ser colocado no cabeçalho do Relatório.

Os **nomes dos arquivos** necessariamente deverão seguir o padrão a seguir, bem como o **título dos emails**:

- nome do arquivo de cada Relatório:

pcs3432-Tx-E07.ext

onde

x é o número da Turma: 1 = segundas-feiras, 2 = terças-feiras e 3 = sextas-feiras

yy é o número da Experiência: 01, 02, ...

ext é a extensão do arquivo: doc, docx, pdf (preferido).

Caso arquivos adicionais sejam enviados (por exemplo, arquivos de programas), os respectivos nomes deverão ser:

pcs3432-Tx-E07-Azz.ext

onde

zz é o número do Anexo (01, 02, ...)

ext é a extensão do arquivo: dependerá do tipo de arquivo.

Os anexos necessariamente deverão ser referenciados no Relatório.

Por comodidade, todos os arquivos de um Relatório (incluindo os anexos) poderão ser compactados no padrão **zip**. Nesse caso, basta enviar um arquivo

pcs3432-Tx-E07.zip

que incluirá o Relatório e respectivos anexos.

PCS 3432 – LABORATÓRIO DE PROCESSADORES
RELATÓRIO - EXPERIÊNCIA 07 - xx/xx/2018

Bancada b: 1º. Componente do Grupo
2º. Componente do Grupo
3º. Componente do Grupo (quando houver)

Professor:
Turma x

1) Seguir o item 6 - PARTE PRÁTICA da apostila:

a) Para todas as turmas, incluir os **itens 6.1 e 6.2** incluindo a parte prevista para a sua turma.

Encaminhar, juntamente com este relatório, os arquivos relativos a todos os programas modificados e elaborados.

2) COMENTÁRIOS E CONCLUSÕES

Incluir comentários relevantes sobre a experiência, bem como problemas enfrentados e respectivas soluções, e itens não realizados da experiência, justificando.

Colocar todos os documentos e arquivos em um único arquivo compactado:
pcs3432-Tx-E07.zip.