

# Microcontroladores

## 1. Introdução

Atualmente um grande número de microcontroladores, integrados em diversos equipamentos, exercem um papel importante no dia a dia das pessoas. Despertar ao som de um CD *Player* programável, tomar café da manhã preparado por um microondas digital, e ir ao trabalho de carro, cuja injeção de combustível é microcontrolada, são apenas alguns exemplos.

O mercado de microcontroladores apresenta-se em franca expansão, ampliando seu alcance principalmente em aplicações residenciais, industriais, automotivas e de telecomunicações. Segundo dados da *National Semiconductor* (1997), uma residência típica americana possui 35 produtos baseados em microcontrolador. Estima-se que, em 2010, em média uma pessoa interagirá com 250 dispositivos com microcontroladores diariamente.

Em um passado recente, o alto custo dos dispositivos eletrônicos limitou o uso dos microcontroladores apenas aos produtos domésticos considerados de alta tecnologia (televisão, vídeo e som). Porém, com a constante queda nos preços dos circuitos integrados, os microcontroladores passaram a ser utilizados em produtos menos sofisticados do ponto de vista da tecnologia, como máquinas de lavar, microondas, fogões e refrigeradores. Assim, a introdução do microcontrolador nestes produtos cria uma diferenciação e permite a inclusão de melhorias de segurança e de funcionalidade. Alguns mercados chegaram ao ponto de tornar obrigatório o uso de microcontroladores em determinados tipos de equipamentos, impondo um pré-requisito tecnológico.

Muitos produtos que temos disponíveis hoje em dia, simplesmente não existiriam, ou não teriam as mesmas funcionalidades sem um microcontrolador. É o caso, por exemplo, de vários instrumentos biomédicos, instrumentos de navegação por satélites, detetores de radar, equipamentos de áudio e vídeo, eletrodomésticos, entre outros.

Entretanto, o alcance dos microcontroladores vai além de oferecer algumas facilidades. Uma aplicação crucial, onde os microcontroladores são utilizados, é na redução de consumo de recursos naturais. Existem sistemas de aquecimento modernos que captam a luz solar e, de acordo com a demanda dos usuários, controlam a temperatura de forma a minimizar perdas. Um outro exemplo, de maior impacto, é o uso

de microcontroladores na redução do consumo de energia em motores elétricos, que são responsáveis pelo consumo de, aproximadamente, 50% de toda eletricidade produzida no planeta. Portanto, o alcance dessa tecnologia tem influência muito mais importante em nossas vidas, do que se possa imaginar.

O universo de aplicações dos microcontroladores, como já mencionado, está em grande expansão, sendo que a maior parcela dessas aplicações é em sistemas embarcados. A expressão “sistema embarcado” (do inglês *embedded system*) se refere ao fato do microcontrolador ser inserido nas aplicações (produtos) e usado de forma exclusiva por elas. Como a complexidade desses sistemas cresce vertiginosamente, o *software* tem sido fundamental para oferecer as respostas às necessidades desse mercado. Tanto é, que o *software* para microcontroladores representa uma fatia considerável do mercado de *software* mundial. Segundo Edward Yourdon (consultor na área de computação, pioneiro nas metodologias de engenharia do software e programação estruturada) a proliferação dos sistemas embarcados, juntamente com o advento da Microsoft, são os responsáveis pela retomada do crescimento da indústria de *software* nos Estados Unidos da América.

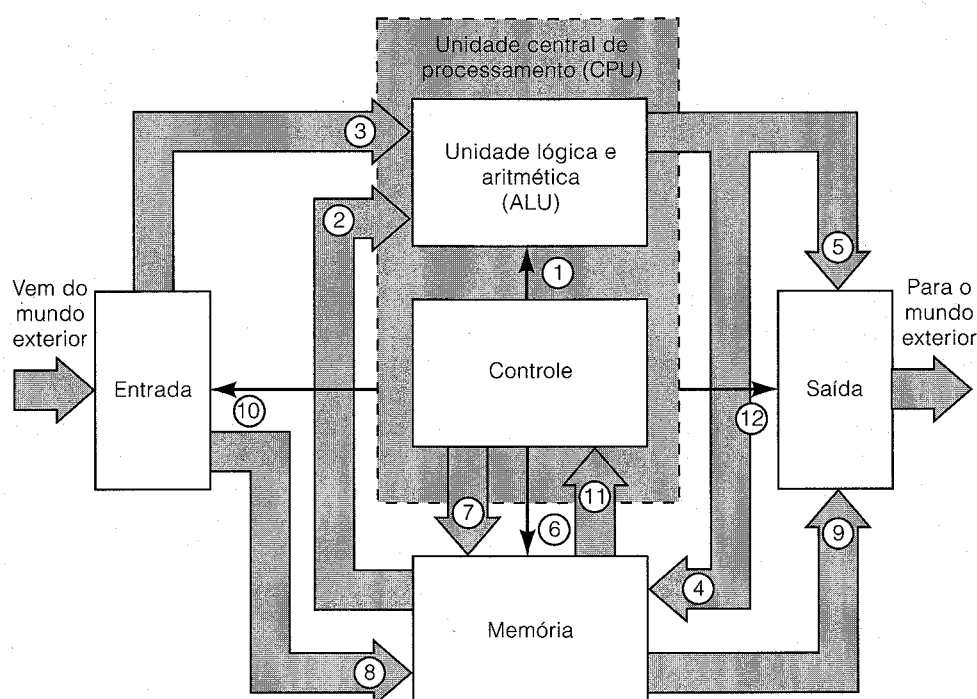
## 2. Definição de Microcontrolador

Um microcontrolador é um sistema computacional completo, no qual estão incluídos uma CPU (*Central Processor Unit*), memória de dados e programa, um sistema de clock, portas de I/O (*Input/Output*), além de outros possíveis periféricos, tais como, módulos de temporização e conversores A/D entre outros, integrados em um mesmo componente. As partes integrantes de qualquer computador, e que também estão presentes, em menor escala, nos microcontroladores são:

- Unidade Central de Processamento (CPU)
- Sistema de *clock* para dar seqüência às atividades da CPU
- Memória para armazenamento de instruções e para manipulação de dados
- Entradas para interiorizar na CPU informações do mundo externo
- Saídas para exteriorizar informações processadas pela CPU para o mundo externo
- Programa (*firmware*) para definir um objetivo ao sistema

## 2.1 Unidade Central de Processamento (CPU)

A unidade central de processamento é composta por uma unidade lógica aritmética (ULA), por uma unidade de controle e por unidades de memória especiais conhecidas por registradores. Para que a CPU possa realizar tarefas é necessário que se agregue outros componentes, como unidades de memória, unidades de entrada e unidades de saída. A figura a seguir apresenta um diagrama de blocos com uma possível interface entre a CPU e outros dispositivos.



A unidade de memória permite armazenar grupos de dígitos binários que podem representar instruções que o processador irá executar ou dados que serão manipulados pelo processador. A unidade de entrada consiste em todos os dispositivos utilizados para obter informações e dados externos ao processador. A unidade de saída consiste em dispositivos capazes de transferir dados e informações do processador para o exterior.

A ULA é a área de uma CPU na qual as operações lógicas e aritméticas são realizadas sobre os dados. O tipo de operação realizada é determinada pelos sinais da unidade de controle. Os dados a serem operados pela ULA podem ser oriundos de uma memória ou de uma unidade de entrada. Os resultados das operações realizadas na ULA podem ser transferidos tanto para uma memória de dados como para uma unidade de saída.

A função da unidade de controle é comandar as operações da ULA e de todas as outras unidades conectadas a CPU, fornecendo sinais de controle e temporização. De certa maneira, a unidade de controle é como um maestro que é responsável por manter cada um dos membros da orquestra em sincronismo. Essa unidade contém circuitos lógicos e de temporização que geram os sinais apropriados necessários para executar cada instrução de um programa.

A unidade de controle busca uma instrução na memória enviando um endereço e um comando de leitura para a unidade de memória. A palavra da instrução armazenada na posição de memória é transferida para um registrador conhecido por registrador de instruções (RI) da unidade de controle. Essa palavra de instrução, que está de alguma forma de código binário, é então decodificada pelos circuitos lógicos na unidade de controle para determinar a instrução que está sendo invocada. A unidade de controle usa essa informação para enviar os sinais apropriados para as unidades restantes a fim de executar a operação específica.

Essa sequência de busca de um código de instrução e de execução da operação indicada é repetida indefinidamente pela unidade de controle. Essa sequência repetitiva de busca/execução continua até que a CPU seja desligada ou até que o RESET seja ativado. O RESET sempre faz a CPU buscar sua primeira instrução no programa.

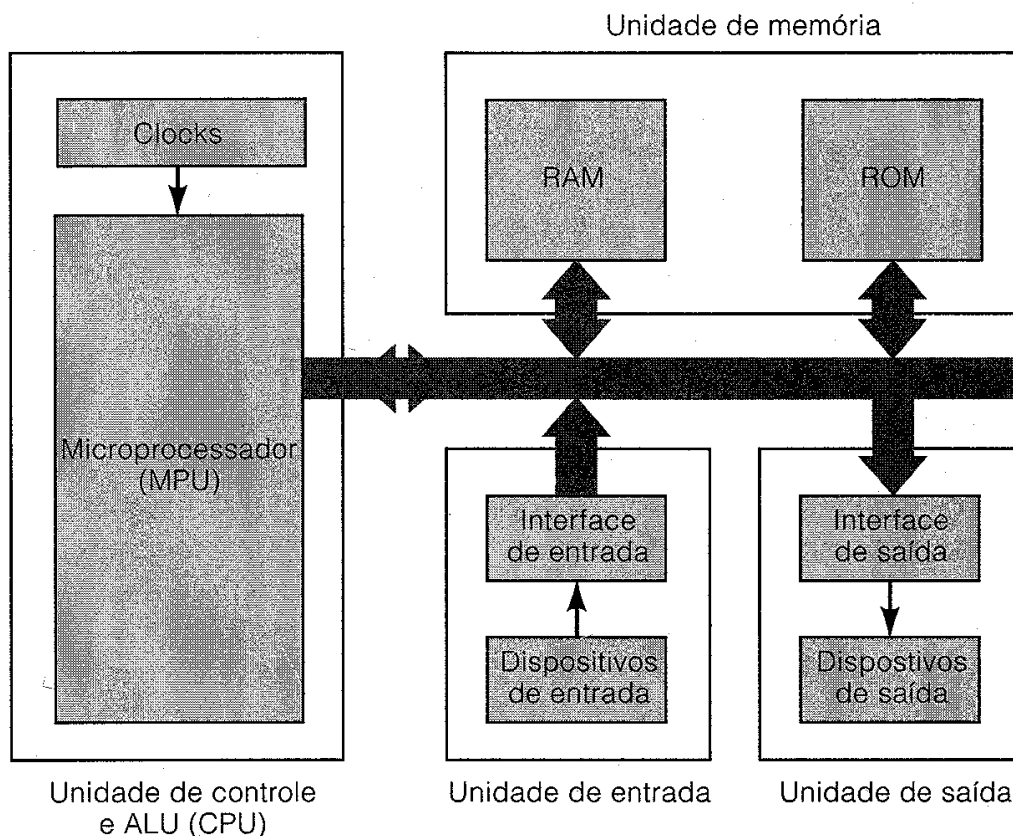
Uma CPU, também conhecida por processador, repete indefinidamente as mesmas operações básicas de busca e execução. Naturalmente, os diversos ciclos de execução serão diferentes para cada tipo de instrução à medida que a unidade de controle envia sinais diferentes para as outras unidades de execução de uma instrução em particular.

Um registrador é um tipo de memória de pequena capacidade porém muito rápida, contida na CPU, utilizado no armazenamento temporário de dados durante o processamento. Os registradores estão no topo da hierarquia de memória, sendo desta forma o meio mais rápido e de maior custo para armazenar um dado.

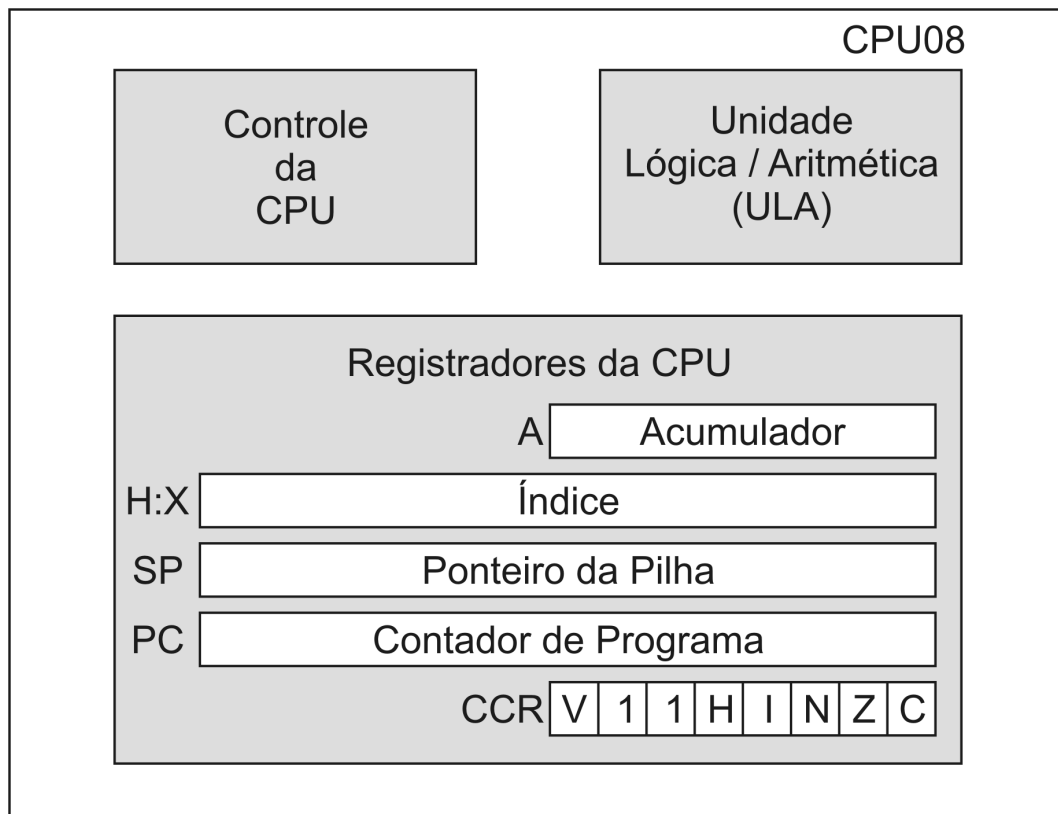
Cada registrador de um processador possui uma função especial. Um dos mais importantes é o contador de programa (*program counter* – PC), que armazena os endereços dos códigos das instruções à medida que são buscadas da memória. Outros registradores são utilizados para realizar funções como: armazenamento de códigos de instrução (RI), manutenção dos dados operados pela ULA (acumulador), armazenamento de endereços de dados a serem lidos na memória (ponteiro de dados), além de outras funções de armazenamento e contagem. Todos os processadores possuem um registrador em especial muito utilizado chamado de acumulador ou registrador A. Ele

armazena um operando para quaisquer instruções, lógica ou matemática. O resultado da operação é armazenado no acumulador após a instrução ser executada.

Para que exista comunicação entre as unidades que formam um processador devemos definir uma forma de conexão entre estas unidades. Em um processador tradicional com arquitetura Von Neuman este meio é o barramento de dados. A largura do barramento de dados em bits é o que determina o número de bits para um dado processador. A figura a seguir apresenta a interface dos principais dispositivos que compõem um sistema microprocessado através de um barramento de dados.



A CPU é o centro de todo sistema computacional, e não é diferente quando se trata de microcontroladores. O trabalho da CPU é executar rigorosamente as instruções de um programa, na sequência programada, para uma aplicação específica. Um programa computacional (software) instrui a CPU a ler informações de entradas, ler e escrever informações na memória de dados, e escrever informações nas saídas. O diagrama de blocos simplificado da CPU presente nos microcontroladores da família HC08, também denominado de CPU08, é apresentado na figura a seguir. Esta arquitetura de processador será utilizada como modelo neste documento.



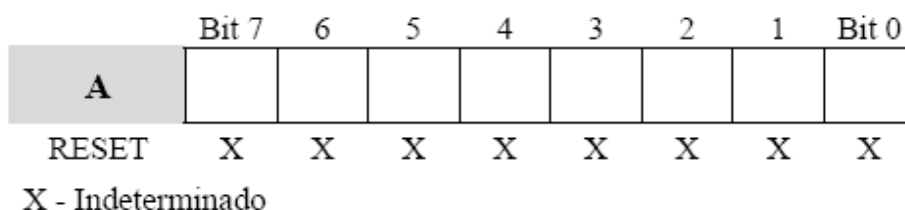
As principais funções de cada um dos componentes da CPU08 serão apresentadas a seguir.

**Unidade Lógica/Aritmética (ULA):** A ULA é utilizada para realizar operações lógicas e aritméticas definidas no conjunto de instruções da CPU. Vários circuitos implementam as operações aritméticas binárias decodificadas pelas instruções e fornecem dados para a execução da operação na ULA. A maioria das operações aritméticas binárias são baseadas em algoritmos de adição e subtração (adição com o valor negativo). A multiplicação é realizada através de uma série de adições e deslocamentos com a ULA sob controle lógico da CPU.

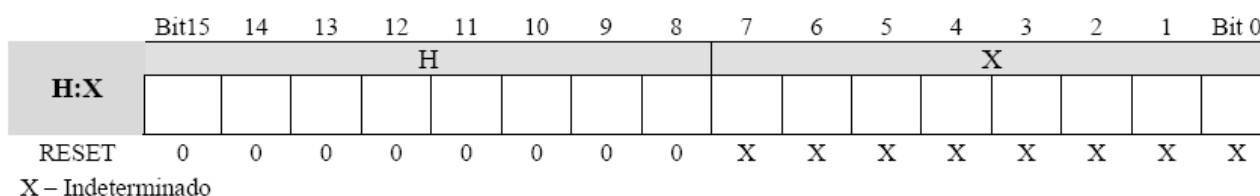
**Controle da CPU:** O circuito de controle da CPU implementa o sequeciamento de elementos lógicos necessários para a ULA realizar as operações requisitadas durante a execução do programa. O elemento central da seção de controle da CPU é o **decodificador de instruções**. Cada *opcode* (código de instrução) é decodificado para determinar quantos operandos são necessários e qual seqüência de passos será necessária para completar a instrução em curso. Quando uma instrução é executada completamente, o próximo *opcode* é lido e decodificado.

**Registradores da CPU:** A CPU08 contém 5 registradores como apresentado na figura anterior. Os registradores da CPU são memórias especiais que não fazem parte do mapa de memória. O conjunto de registradores da CPU é freqüentemente chamado de **modelo de programação**.

O **acumulador**, também chamado de registrador A, é freqüentemente utilizado para armazenar um dos operandos ou o resultado de operações.



O **registrador H:X** é um registrador de 16 bits de índice que possibilita ao usuário endereçar indiretamente o espaço de memória de 64Kbytes. O byte mais significativo do registrador de índice é denominado H, e o byte menos significativo denominado X. Sua principal função é servir de apontador para uma área na memória onde a CPU irá carregar (ler) ou armazenar (escrever) informação. Quando não estiver sendo utilizado para apontar um endereço na memória, ele pode ser utilizado como registrador genérico.



O **registrador Program Counter (PC)** é usado pela CPU para controlar e conduzir ordenadamente a busca do endereço da próxima instrução a ser executada. Quando a CPU é energizada ou passa por um processo de reset, o PC é carregado com o conteúdo de um par de endereços específicos denominados **vetor de reset** (*reset vector*). O vetor de reset contém o endereço da primeira instrução a ser executada pela CPU. Assim que as instruções são executadas, uma lógica interna a CPU incrementa o PC, de tal forma que ele sempre aponte para o próximo pedaço de informação que a CPU vai precisar. O número de bits do PC coincide exatamente com o número de linhas do barramento de endereços, que por sua vez determina o espaço total disponível de memória que pode ser acessada pela CPU.

O **registrador *Condition Code*** (CCR) é um registrador de 8 bits que armazena os bits de estado (*flags*) que refletem o resultado de algumas operações da CPU. As instruções de desvio usam estes bits de estado para tomar suas decisões.

	Bit 7	6	5	4	3	2	1	Bit 0
<b>CCR</b>	<b>V</b>	<b>1</b>	<b>1</b>	<b>H</b>	<b>I</b>	<b>N</b>	<b>Z</b>	<b>C</b>
RESET	X	1	1	X	1	X	X	X

X – Indeterminado

A descrição dos bits do registrador de condição é apresentada abaixo:

**V** (*Bit de Overflow*) - A CPU leva o bit de *overflow* para nível lógico alto quando houver estouro no resultado de uma operação em complemento de 2. O bit V é utilizado pelas instruções de desvios condicionais BGT, BGE, BLE, e BLT.

**H** (*Bit de Half-carry*) - A CPU leva o bit de *half-carry* para nível lógico alto quando ocorrer estouro entre os bits 3 e 4 do acumulador durante as operações ADD e ADC. O bit H é importante nas operações aritméticas codificadas em binário (BCD). A instrução DAA utiliza o estado dos bits H e C para determinar o fator de correção apropriado.

**I** (*Máscara de Interrupções*) - Quando o bit I está em nível lógico alto, todas as interrupções são mascaradas (desabilitadas). As interrupções são habilitadas quando o bit I é levado a nível lógico baixo. Quando ocorre uma interrupção, o bit que mascara as interrupções é automaticamente levado a nível lógico alto. Depois que os registradores da CPU são armazenados na pilha este bit volta ao nível lógico baixo. Se uma interrupção ocorrer enquanto o bit I estiver setado, seu estado será guardado. As interrupções são atendidas, em ordem de prioridade, assim que o bit I for a nível lógico 0. A instrução retorno da interrupção (RTI) retorna os registradores da CPU da pilha, e restaura o bit I no seu estado de nível lógico 0. Após qualquer reset, o bit I é colocado em nível lógico alto e só pode ser limpo por uma instrução de software (CLI).

**N** (*Bit Negativo*) - A CPU coloca o bit N em nível lógico alto quando uma operação aritmética, lógica ou de manipulação de dados produzir um resultado negativo. Corresponde ao 8º bit do registrador que contém o resultado.

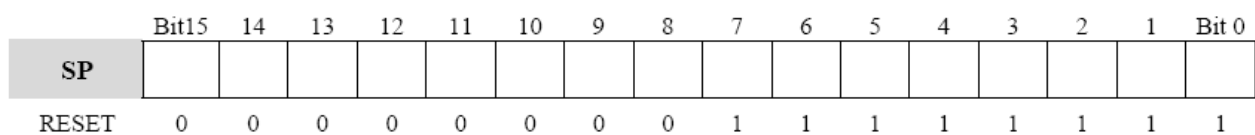
**Z** (*Bit Zero*) - A CPU leva o bit Z para nível lógico alto quando uma operação aritmética, lógica ou de manipulação de dados produzir um resultado igual a 0.

**C** (*Bit Carry/Borrow*) - A CPU coloca o bit C em nível lógico alto quando uma operação de adição produzir um valor superior a 8 bits ou quando uma subtração



necessitar um empréstimo. Algumas operações lógicas e as instruções de manipulação de dados também podem modificar o estado do bit C.

O **Stack Pointer** (SP) é um registrador cuja função é apontar para a próxima localização disponível (endereço livre) de uma pilha (lista de endereços contíguos). A pilha pode ser vista como um monte de cartas empilhadas, onde cada carta armazena um byte de informação. A qualquer hora, a CPU pode colocar uma carta nova no topo da pilha ou retirar uma carta do topo da pilha. As cartas que estão no meio da pilha não podem ser retiradas até que todas que estejam acima dela sejam removidas primeiro. A CPU acompanha o efeito da pilha através do valor armazenado no SP. O SP sempre aponta para a localização de memória disponível para se colocar a próxima carta (byte).



Normalmente, a CPU usa a pilha para guardar os endereços de retorno e o contexto, isto é, os registradores da CPU, na ocorrência de uma exceção (interrupção ou reset).

Durante um reset, o *Stack Pointer* contém o endereço 0x00FF. A instrução RSP (*Reset Stack Pointer*) carrega o byte menos significativo com 0xFF e o byte mais significativo não é afetado.

Quando a CPU insere um novo dado na pilha, automaticamente o SP é decrementado para o próximo endereço livre. Quando a CPU retira um dado da pilha, o SP é incrementado para apontar para o dado mais recente, e o valor do dado é lido nesta posição. Quando a CPU é energizada ou passa por um processo de reset, o SP aponta para um endereço específico na memória RAM (no caso dos microcontroladores HC08 e HCS08 = 0x00FF).

A CPU08 possui modos de endereçamento indexado com *offsets* de 8 ou 16 bits do SP para acesso de variáveis temporárias inseridas na pilha. A CPU utiliza o conteúdo do registrador SP para determinar o endereço efetivo do operando.

**OBS:** Embora o endereço inicial do SP seja 0x00FF, a localização da pilha é arbitrária e pode ser realocada pelo usuário em qualquer lugar na RAM. Movimentar o SP para fora da página de acesso direto (0x0000 a 0x00FF) permitirá que este espaço de memória seja utilizado para modos de endereçamento mais eficientes.

## 2.2. Sistema de Clock

Todo sistema computacional utiliza um clock para fornecer a CPU uma maneira de se mover de instrução em instrução, em uma sequência pré-determinada.

Uma fonte de clock de alta frequência (normalmente derivada de um cristal ressonador conectado a CPU) é utilizada para controlar o sequenciamento das instruções da CPU. Normalmente as CPUs dividem a frequência básica do cristal por 2 ou mais para chegar ao clock do barramento interno. Cada ciclo de leitura ou escrita a memória é executado em um ciclo de clock do barramento interno, também denominado ciclo de barramento (*bus cycle*).

## 2.3. Memória

Podemos pensar na memória como sendo uma lista de endereços postais, onde o conteúdo de cada endereço é um valor fixo de 8 bits (para CPU de 8 bits). Se um sistema computacional tem  $n$  linhas (bits) de endereços, ele pode endereçar  $2^n$  posições de memória (p.ex.: um sistema com 14 linhas pode acessar  $2^{14} = 16384$  endereços). Entre os diversos tipos de memória encontram-se:

**RAM** (*Random Access Memory*) – Memória de acesso aleatório. Pode ser lida ou escrita pela execução de instruções da CPU e, normalmente é utilizada para manipulação de dados pela CPU. O conteúdo é perdido na ausência de energia (memória volátil).

**ROM** (*Read Only Memory*) – Memória apenas de leitura. Pode ser lida, mas não é alterável. O conteúdo deve ser determinado antes que o circuito integrado seja fabricado. O conteúdo é mantido na ausência de energia (memória não volátil).

**EPROM** (*Erasable and Programmable Read-Only Memory*) – Memória ROM programável e apagável. O conteúdo dessa memória pode ser apagado com luz ultravioleta, e posteriormente, reprogramado com novos valores. As operações de apagamento e programação podem ser realizadas um número limitado de vezes depois que o circuito integrado for fabricado. Da mesma forma que a ROM, o conteúdo é mantido na ausência de energia (memória não volátil).

**OTP** (*One Time Programmable*) – Memória programável uma única vez. Semelhante à EPROM quanto a programação, mas que não pode ser apagada.

**EEPROM** (*Electrically Erasable and Programmable Read-Only Memory*) – Memória ROM programável e apagável eletricamente. Pode ter seu conteúdo alterado através da

utilização de sinais elétricos convenientes. Tipicamente, um endereço de uma EEPROM pode ser apagada e reprogramada até 100.000 vezes.

**FLASH** – Memória funcionalmente semelhante a EEPROM, porém com ciclos de escrita bem mais rápidos.

*I/O (Input/Output)* – Registradores de controle, estado e sinais de I/O são um tipo especial de memória porque a informação pode ser sentida (lida) e/ou alterada (escrita) por dispositivo diferentes da CPU.

## **2.4. Sinais de Entrada**

Dispositivos de entrada fornecem informação para a CPU processar, vindas do mundo externo. A maioria das entradas que os microcontroladores processam são denominadas sinais de entrada digitais, e utilizam níveis de tensão compatíveis com a fonte de alimentação do sistema. O sinal de 0V (GND ou VSS ) indica o nível lógico 0 e o sinal de fonte positiva, que tipicamente é +5VDC (VDD ) indica o nível lógico 1 (atualmente os microcontroladores começaram a reduzir a tensão de VDD para valores na faixa dos 3V).

Naturalmente que no mundo real existem sinais puramente analógicos (com uma infinidade de valores) ou sinais que utilizam outros níveis de tensão. Alguns dispositivos de entrada traduzem as tensões do sinal para níveis compatíveis com VDD e VSS. Outros dispositivos de entrada convertem os sinais analógicos em sinais digitais (valores binários formados por 0s e 1s) que a CPU pode entender e manipular. Alguns microcontroladores incluem circuitos conversores analógicos/digitais (ADC) encapsulados no mesmo componente.

## **2.5. Sinais de Saída**

Dispositivos de saída são usados para informar ou agir no mundo exterior através do processamento de informações realizados pela CPU. Circuitos eletrônicos (algumas vezes construídos no próprio microcontrolador) podem converter sinais digitais em níveis de tensão analógicos. Se necessário, outros circuitos podem alterar os níveis de tensão VDD e VSS nativos da CPU em outros níveis.

## 2.6. Códigos de operação (*opcodes*)

Os programas usam códigos para fornecer instruções para a CPU. Estes códigos são chamados de códigos de operação ou *opcodes*. Cada *opcode* instrui a CPU a executar uma sequência específica para realizar sua operação. Microcontroladores de diferentes fabricantes usam diferentes conjuntos de *opcodes* porque são implementados internamente por hardware na lógica da CPU. O conjunto de instruções de uma CPU especifica todas as operações que podem ser realizadas. *Opcodes* são uma representação das instruções que são entendidas pela máquina, isto é, uma codificação em representação binária a ser utilizada pela CPU. Mnemônicos são outra representação para as instruções, só que agora, para serem entendidas pelo programador.

## 2.7. Mnemônicos das instruções e assembler

Um *opcode* como 0x4C é entendido pela CPU, mas não é significativo para nós humanos. Para resolver esse problema, um sistema de instruções mnemônicas equivalentes foram criadas (Linguagem Assembly). O *opcode* 0x4C corresponde ao mnemônico INCA, lê-se “incrementa o acumulador”, que é muito mais inteligível. Para realizar a tradução de mnemônicos em códigos de máquina (*opcodes* e outras informações) utilizados pela CPU é necessário um programa computacional chamado assembler (compilador para linguagem Assembly). Um programador utiliza um conjunto de instruções na forma de mnemônicos para desenvolver uma determinada aplicação, e posteriormente, usa um assembler para traduzir estas instruções para *opcodes* que a CPU pode entender.

Após a descrição da unidade central de processamento de um microcontrolador podemos partir para o aprendizado da linguagem de programação Assembly. Recomenda-se a leitura da folha de dados (principalmente a seção que trata do conjunto de instruções) do microcontrolador, bem como da apostila do microcontrolador HC08, família QT/QY. O próximo capítulo deste documento irá descrever diversos periféricos que podem compor um microcontrolador, como portas de entrada/saída, temporizadores, entre outros.

### 3. Periféricos

Os microcontroladores normalmente são classificados em famílias, dependendo da aplicação a que se destinam. A partir da aplicação que a família de microcontroladores se destina, um conjunto de periféricos específicos é escolhido e integrado a um determinado microprocessador. Estes microprocessadores normalmente operam com barramentos de 8, 16 ou 32 bits, e apresentam arquiteturas RISC (*Reduced Instruction Set Computer*) ou CISC (*Complex Instruction Set Computer*). Alguns exemplos de microcontroladores que utilizam microprocessadores com arquitetura RISC são o PIC (*Microchip*) e o MSP430 (*Texas Instruments*). Já o MC68HC08 e HCS08 (*Freescale*) e o 8051 (Intel) são exemplos de microcontroladores que utilizam arquitetura CISC.

Apesar da classificação dos microcontroladores em famílias, existem periféricos necessários a praticamente todas as aplicações, que são a memória de dados e a memória de programa. A memória de dados mais utilizada é a RAM (*Random Access Memory*), que é uma memória volátil, ou seja, não preserva o seu conteúdo sem uma fonte de alimentação.

Recentemente as memórias de programa sofreram uma grande mudança. A alguns anos atrás as memórias de programa mais utilizadas eram a ROM (*Read-Only Memory*) e a EPROM (*Erasable Programmable Read-Only Memory*). O grande problema da utilização de tais memórias era a falta de praticidade durante o desenvolvimento de um sistema embarcado. Com a popularização das memórias FLASH e, ainda, devido a facilidade de utilização, cada vez mais os microcontroladores tendem a ser produzidos com esta memória, em substituição a ROM e a EPROM. Importante lembrar que a praticidade da memória FLASH se deve a esta memória ser uma variação das EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) que permitem que múltiplos endereços sejam apagados ou escritos com sinais elétricos.

A seguir serão apresentadas as características e aplicações dos principais periféricos encontrados em microcontroladores, tais como: portas de entrada e saída, temporizadores, portas de comunicação serial e conversores Analógico-digitais (A/D).

### 3.1 Temporizadores

Um microprocessador deve possuir um relógio. O relógio pode ser implementado por um cristal oscilador que sincroniza todo o funcionamento do microprocessador, controlando o tempo de cada um dos eventos relacionados aos dispositivos integrados a ele.

Os temporizadores utilizam a base de tempo do relógio para poder implementar contagens de tempo bem específicas e configuráveis. Estes utilizam contadores, incrementados na mesma base de tempo do relógio. Desta forma é possível descrever tempo em número de ciclos de um relógio. Por exemplo, imagine um microprocessador utilizando um relógio de 20 MHz. O período relativo a esta frequência é 50 ns. Podemos representar então um tempo de 1ms através de períodos de 50ns, obtendo o valor de 20000. Ou seja, se incrementarmos um contador a cada ciclo de relógio, no caso 50ns, quando este contador atingir o valor de 20000, teremos atingido a contagem de 1ms.

Para que este método seja aplicado, existe a necessidade da utilização de pelo menos 2 registradores. O registrador que será incrementado e o registrador que conterá o valor a ser atingido. No entanto, existe um problema relacionado a este método. Imagine que precisemos de um tempo na ordem de segundos, por exemplo, 1 segundos. Se estivermos utilizando um relógio de 4 MHz, seria necessário registradores de 24 bits para representar o valor de  $4 \times 10^6$ . Com o intuito de reduzir o tamanho destes contadores, os temporizadores apresentam a possibilidade de divisão do relógio, normalmente por valores múltiplos de 2. No caso do exemplo acima, poderíamos dividir o relógio por 128, obtendo uma frequência de 31,250 KHz. Então, para representarmos 1 segundo, seria necessário uma contagem de 31250 períodos de 32 $\mu$ s, ou seja, um valor que pode ser representado em 16 bits.

Devemos lembrar que a divisão terá influência somente no tempo de incrementação do contador, continuando o barramento interno do microprocessador a operar com o relógio original.

Para exemplificar a configuração dos registradores relativos a um temporizador em um microcontrolador será utilizado o microcontrolador MC68HC908QY4. O processo de configuração é apresentado a seguir.

Os microcontroladores da linha HC08 normalmente possuem 1 ou 2 temporizadores. Os registradores relativos a estes temporizadores apresentam nomes semelhantes, tendo apenas o número no temporizador para diferenciá-los. Os três

registradores de configuração do temporizador neste microcontrolador são: TSC, TCNT e TMOD. Abaixo é apresentado o diagrama de blocos do temporizador destes microcontroladores. É importante ressaltar que alguns dos registradores apresentados na figura são relativos ao módulo PWM e captura de entrada.

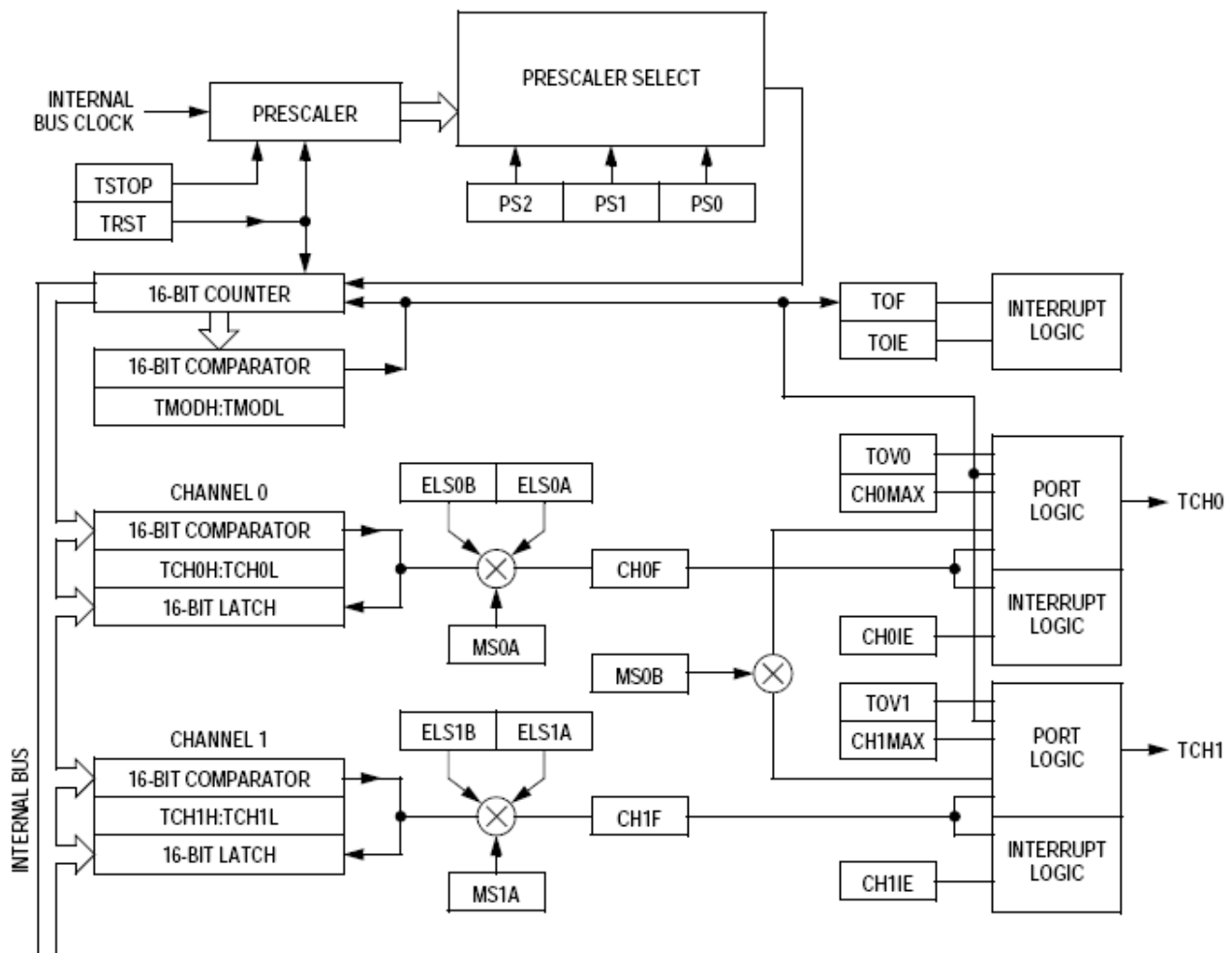


Figura – Diagrama de blocos do módulo temporizador de microcontrolador MC68HC908QY4

#### TSC (*Timer Status and Control Register*):

Possibilita habilitar a interrupção do temporizador, verificar o estado da *flag* de interrupção, para-lo, reiniciar a contagem e dividir o relógio para obter a base de tempo.

Address: \$0020

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	TOF	TOIE	TSTOP	0	0	PS2	PS1	PS0
Write:	0			TRST				
Reset:	0	0	1	0	0	0	0	0


 = Unimplemented

Figura – Registrador TSC

Abaixo serão descritos as funções de cada um dos bits deste registrador:

**TOF (*Timer Overflow Flag*):** Este bit de escrita/leitura torna-se é setado quando o registrador contador (TCNT) atinge o valor do registrador de módulo de contagem (TMOD), condição essa que indica o estouro da contagem de tempo. O procedimento correto para limpar esta indicação é ler o registrador TSC e escrever um “0” lógico para o bit TOF;

1 = O módulo temporizador atingiu o valor desejado

0 = O módulo temporizador não atingiu o valor desejado

**TOIE (*Timer Overflow Interrupt Enable Bit*):** Este bit de escrita/leitura habilita a interrupção do temporizador quando o bit TOF for setado.

1 = Interrupção do temporizador ativa

0 = Interrupção do temporizador desabilitada

**TSTOP (*Timer Stop Bit*):** Este bit de escrita/leitura para o incremento do contador de tempo.

1 = Contador de tempo parado

0 = Contador de tempo ativo

**TRST (*Timer Reset Bit*):** Levar este bit de escrita para nível lógico 1 irá iniciar o contador de tempo com zero e colocar o divisor de base de tempo para o estado inicial, ou seja, divisão por 1.

1 = Pré-Escala de base de tempo e contador iniciados com o valor “0”;

0 = Sem efeito



PS[2:0] (*Prescaler Select Bits*) – Bits de pré-escala da base de tempo. Estes bits de escrita/leitura selecionam um dos sete possíveis valores de divisão da base de tempo do relógio para utilização como base de tempo do temporizador.

PS2	PS1	PS0	Base de tempo do temporizador
0	0	0	Clock de barramento interno / 1
0	0	1	Clock de barramento interno / 2
0	1	0	Clock de barramento interno / 4
0	1	1	Clock de barramento interno / 8
1	0	0	Clock de barramento interno / 16
1	0	1	Clock de barramento interno / 32
1	1	0	Clock de barramento interno / 64
1	1	1	Não disponível

TCNT (*Timer Count Registers*):

Estes registradores são somente de leitura e contém o valor mais significativo (TCNTH) e menos significativo (TCNTL) do contador do temporizador. A leitura do registrador mais significativo deve ser realizada primeiro.

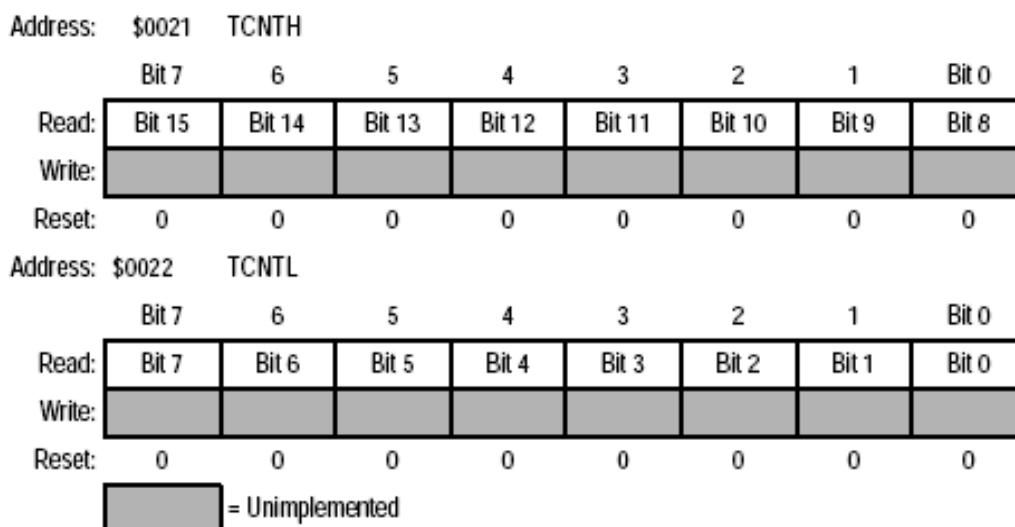


Figura – Registrador TCNTH e TCNTL.

TMOD (*Timer Module Registers*):

Estes registradores de escrita/leitura contém o valor do módulo da contagem do temporizador. Quando os registradores de contagem (TCNT) atingem o valor dos registradores de módulo (TMOD), o bit TOF torna-se nível lógico “1” e os registradores de

contagem resumem a contagem para \$0000 até o próximo passo de clock. Escrever no registrador TMODH inibe o bit TOF até que o registrador TMODL seja escrito.

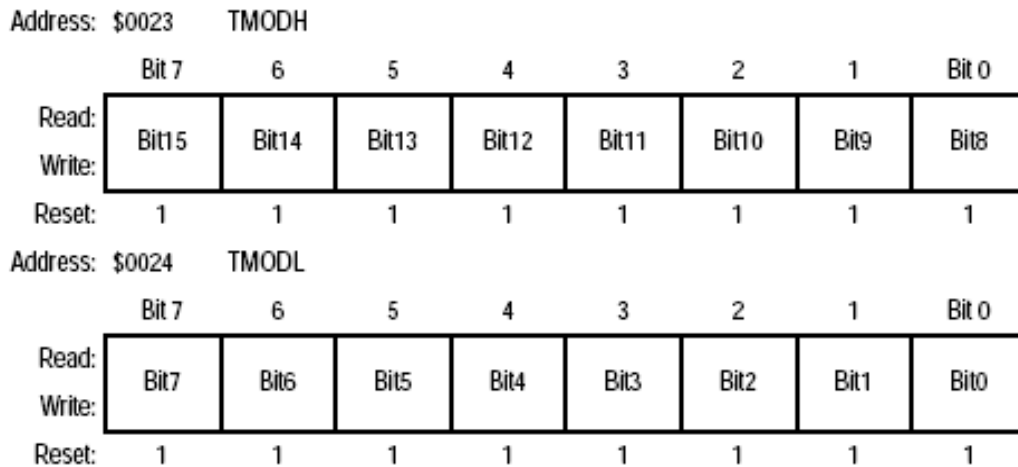


Figura – Registradores TMODH e TMODL

Estando os registradores a serem configurados apresentados pode-se demonstrar um exemplo de utilização. Abaixo será implementada a configuração de um temporizador utilizando um relógio de 3,2 MHz para obter a base de tempo de 10ms.

$$\text{Período relativo a base de tempo: } \text{Período} = \frac{1}{(3,2 \times 10^6)} = 312,5 \times 10^{-9}$$

$$\text{Cálculo do valor de módulo de tempo: } \text{Módulo} = \frac{10 \times 10^{-3}}{(312,5 \times 10^{-9})} = 32000$$

O valor 32000 deve ser escrito nos registradores TMODH e TMODL da seguinte maneira: O valor deve ser convertido para hexadecimal, sendo este igual a 7D00h. O valor obtido é um valor válido dentro dos 16 bits referentes aos registradores TMODH e TMODL, ou seja, a base de tempo do relógio foi dividida por 1. Desta forma, TMODH é igual a 7Dh e TMODL é igual a 00h. Abaixo o exemplo de configuração dos registradores para este caso é apresentado, tanto em assembly, quanto em linguagem “c”.

Assembly:

```
MOV    #$7D,TMODH
CLR     TMODL
; Configuração do registrador TSC:
; Divisão por 1, timer ativo com interrupção do temporizador habilitada.
; TOF TOIE TSTOP TRST Reservado PS2 PS1 PS0
; 0 1 0 0 0 0 0 0
MOV     #$40,TSC
```

“c”:

```
TMOD = 32000;
/*Configuração do registrador TSC:
Divisão por 1, timer ativo com interrupção do temporizador habilitada.
TOF TOIE TSTOP TRST Reservado PS2 PS1 PS0
0 1 0 0 0 0 0 0 */
TSC = 0x40;
```

Para demonstrar um caso onde se faz necessário a divisão da base de tempo, o temporizador será configurado utilizando um relógio de 3,2 MHz para obter a base de tempo de 100ms.

$$\text{Período relativo a base de tempo: } \textit{Período} = \frac{1}{(3,2 \times 10^6)} = 312,5 \times 10^{-9}$$

$$\text{Cálculo do valor de módulo de tempo: } \textit{Módulo} = \frac{100 \times 10^{-3}}{(312,5 \times 10^{-9})} = 320000$$

Para representar 320000 é necessário mais do que os 16 bits disponíveis. Desta forma se faz necessário a divisão da base de tempo. Dividindo 320000 pelo maior valor possível em 16 bits, encontramos 4,88. Assumimos então o próximo valor válido de divisão, ou seja, oito, como fator de divisão da base de tempo.

$$\text{Fator de divisão da base de tempo: } \textit{Fator} = \frac{320000}{65535} = 4,88$$

E, recalculando os valores de configuração:

$$\text{Período relativo a base de tempo: } \textit{Período} = \frac{1}{\left(\frac{3,2 \times 10^6}{8}\right)} = 2,5 \times 10^{-6}$$

Cálculo do valor de módulo de tempo:  $Módulo = \frac{100 \times 10^{-3}}{(2,5 \times 10^{-6})} = 40000$

Sendo 40000 um valor válido em 16 bits, a configuração do temporizador em linguagem “c” é apresentada abaixo:

“c”:

```
TMOD = 40000;
/*Configuração do registrador TSC:
Divisão por 8, timer ativo com interrupção do temporizador habilitada.
TOF TOIE TSTOP TRST Reservado PS2 PS1 PS0
0 1 0 0 0 0 1 1 */
TSC = 0x43;
```

Obs: Devemos lembrar que o código contido nas interrupções deve ser o menor possível, com o intuito de evitar que o tempo de execução deste código seja superior ao tempo da próxima interrupção. Por exemplo, se configurarmos a interrupção do temporizador para ocorrer a cada 100µs, o tempo de execução do código contido nesta interrupção não deve ser superior a esta base de tempo.

Exemplo de utilização da interrupção do temporizador (esta interrupção é conhecida como interrupção de estouro de tempo):

Assembly:

```
; Interrupção do temporizador.
TOVER: BCLR 7,TSC      ; Limpa a flag da interrupção do temporizador
        INC  I          ; Incrementa uma variável qualquer
        RTI             ; Retorna da interrupção
```

“c”:

```
// Interrupção do temporizador
interrupt void tover(void) {
    TSC_TOF = 0;          // Limpa a flag da interrupção do temporizador
    i++;                  // Incrementa uma variável qualquer
}                          // Retorna da interrupção
```

### 3.2 PWM (*Pulse Width Modulation*)

O módulo de geração de Modulação por Largura de Pulso (PWM) é um recurso muito utilizado para o controle de motores e conversores CC-CC em geral. A partir dele é possível gerar um sinal analógico, apesar de sua saída ser um sinal digital que assume apenas os níveis lógicos alto (um) e baixo (zero). A saída gerada é uma onda quadrada, com frequência constante e largura de pulso variável. Estes conceitos estão diretamente relacionados com o período fixo e o ciclo ativo (*duty cycle*) respectivamente.

A frequência de uma onda pode ser definida como a quantidade de vezes que ela se repete no tempo. E o período é cada pedaço dessa onda que irá se repetir.

O *duty cycle* define o tempo de sinal ativo (nível lógico alto) em um período fixo. Assim, quando temos um *duty cycle* de 100%, temos nível lógico alto por todo o período. Um *duty cycle* de 50% define a metade do período em nível lógico alto e a outra metade em nível lógico baixo. Se uma saída TTL for utilizada, a tensão média de saída em um *duty cycle* de 50% será 2,5V. Estes conceitos são demonstrados na figura abaixo. Devemos lembrar que o PWM nem sempre possui estado inicial positivo, podendo iniciar o período com nível lógico baixo.

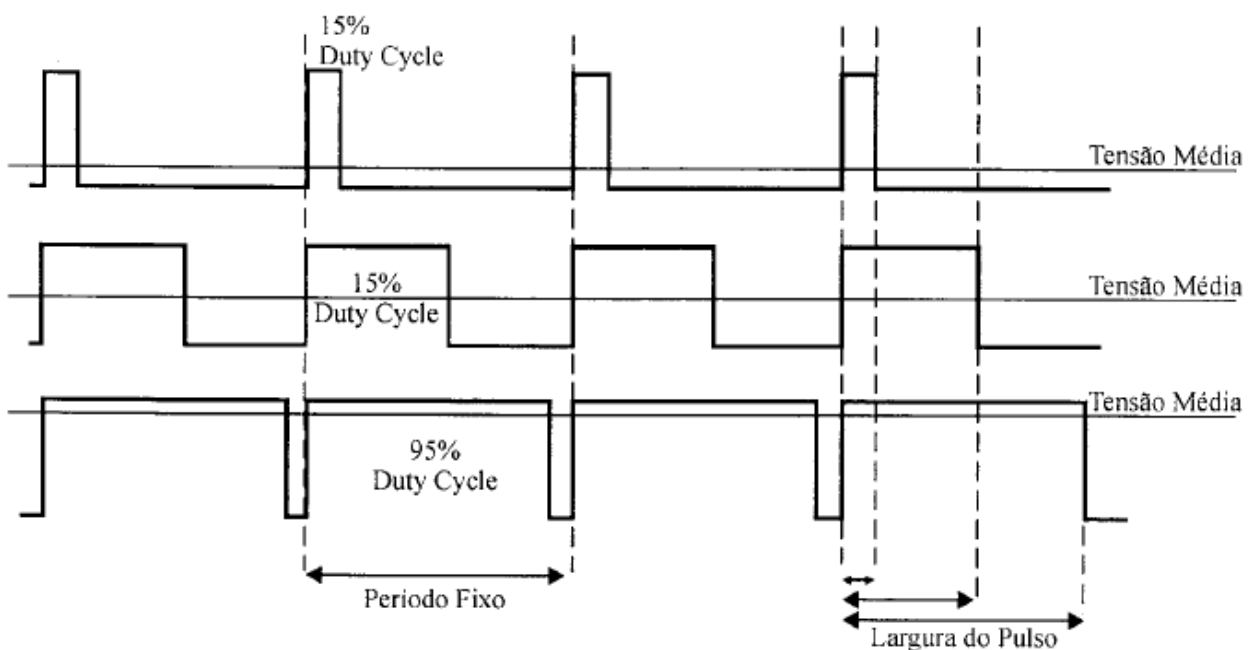


Figura – Sinais modulados por Largura de Pulso

A base de tempo dos módulos PWM normalmente é implementada de duas formas. Uma destas formas é utilizando o próprio módulo temporizador como base de tempo no PWM, ou seja, se o temporizador está configurado para um período de 1ms, a frequência do PWM será de 1 KHz. A outra forma é utilizando um temporizador específico para o PWM, que deve ser configurado para a frequência desejada. Ainda, um temporizador pode ser utilizado como base de tempo de várias saídas PWM, ou seja, vários PWM com a mesma frequência, mas larguras de pulso diferentes.

A figura a seguir irá exemplificar o funcionamento de um PWM em um microcontrolador onde o registrador PTPER possui o valor referente ao período do PWM e os registradores PWM1H e PWM2H representam dois canais de saída PWM.

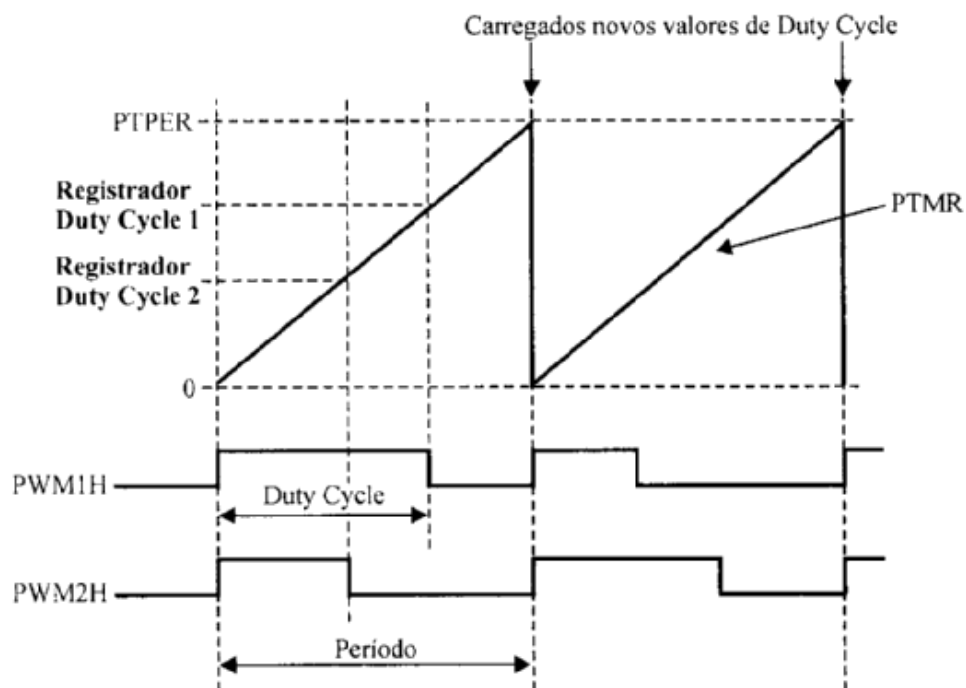


Figura – Sinais PWM com mesmo período e largura de pulso diferentes.

Para exemplificar a configuração de um módulo PWM será utilizado novamente o microcontrolador MC68HC908QY4. Este microcontrolador utiliza o temporizador como base de tempo para o período do PWM. Desta forma, para configurar a frequência do PWM deve-se utilizar a mesma metodologia adotada para o temporizador. Neste exemplo será configurado um PWM de 10 KHz com largura de pulso inicial do ciclo ativo de 40%, onde o relógio utilizado será de 3,2 MHz.

Período da base de tempo do relógio:  $Período = \frac{1}{(3,2 \times 10^6)} = 312,5 \times 10^{-9}$

Valor de módulo de tempo para 100µs:  $Módulo = \frac{100 \times 10^{-6}}{(312,5 \times 10^{-9})} = 320$

Estando o módulo da base de tempo do PWM configurado, deve-se partir para a configuração da largura do pulso e do nível do ciclo inicial do PWM. Através do módulo PWM deste microcontrolador é possível obter duas saídas PWM para cada temporizador. Como o microcontrolador utilizado possui somente um temporizador, é possível obter duas saídas PWM, conhecidas como canal 0 e canal 1. Os registradores de configuração do módulo PWM são: TSC0, TSC1, TCH0 e TCH1. Pode-se notar que os valores 0 e 1 são relativos ao canal a que se destina a configuração.

Este microcontrolador apresenta duas maneiras de se utilizar estes canais. As saídas PWM podem ser configuradas no modo com *buffer* ou sem *buffer*. No modo com *buffer* ambos os canais são aproveitados para gerar uma saída PWM. O modo com *buffer* opera da seguinte maneira: Ambos os canais são configurados para operar com *buffer*. A configuração da largura de pulso inicial é realizada no canal zero. Quando se desejar alterar a largura de pulso da saída PWM, altera-se o valor da largura de pulso no canal um. Ou seja, toda vez que se desejar alterar o valor da largura de pulso é realizado um intercalamento entre os dois canais. A saída PWM ficará agregada ao pino de saída do canal zero.

Já no modo sem *buffer*, cada canal de tempo pode gerar uma saída PWM. O cuidado que deve ser tomado neste caso é que alterações de largura de pulso devem ser realizadas em locais bem específicos. A alteração da largura de pulso para um valor superior deve ser realizada na interrupção de estouro de tempo (temporizador). A alteração da largura de pulso para um valor inferior deve ser realizada na interrupção de comparação do devido canal. Na figura a seguir é apresentado os locais referentes as interrupções supracitadas.

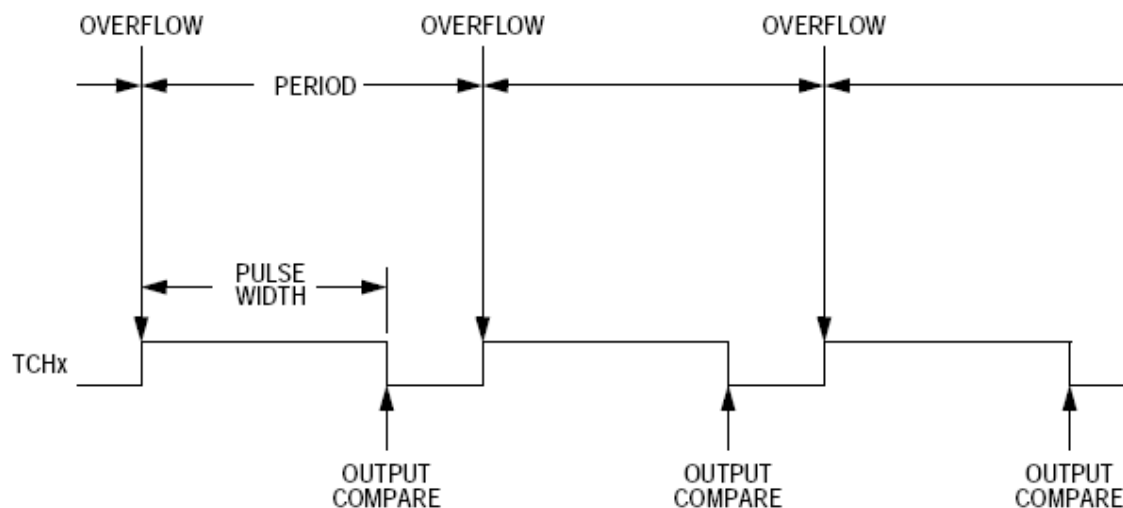


Figura – Largura de pulso e período de um PWM

Abaixo temos a configuração de bits relativa aos registradores TSC0 e TSC1.

Address: \$0025	TSC0							
	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX
Write:	0							
Reset:	0	0	0	0	0	0	0	0

Address: \$0028	TSC1							
	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CH1F	CH1IE	0	MS1A	ELS1B	ELS1A	TOV1	CH1MAX
Write:	0							
Reset:	0	0	0	0	0	0	0	0


 = Unimplemented

Figura – Timer Channel Status And Control Registers

A seguir serão descritos as funções de cada um dos bits deste registrador:

**CHxF (Channel x Flag Bit):** Quando o canal x está configurado para comparação de saída (modo que permite a implementação de um PWM), esta flag torna-se 1 quando o valor do registrador contador de tempo (TCNT) atinge a valor contido no registrador do canal x (TCHx). Para limpar esta flag deve-se ler o registrador TSCx e escrever um zero lógico para este bit.



- 1 = Comparação de saída no canal x
- 0 = Sem comparação de saída no canal x

ChxIE (Channel x Interrupt Enable Bit): Este bit de escrita/leitura permite habilitar a interrupção de comparação de saída para o canal x.

- 1 = Interrupção do canal x habilitada
- 0 = Interrupção do canal x desabilitada

TOVx (*Toggle On Overflow Bit*): Quando o canal x está configurado para comparação de saída, este bit de escrita/leitura controla o comportamento da saída do canal x quando ocorre um estouro de tempo no temporizador.

- 1 = Valor lógico no pino relativo ao canal x se altera no estouro de tempo
- 0 = Valor lógico no pino relativo ao canal x não se altera no estouro de tempo

ChxMAX (*Channel x Maximum Duty Cycle Enable Bit*): Quando o bit TOVx está em nível lógico 1, ao setar o bit ChxMAX irá forçar o *Duty Cycle* do canal para 100%. Como a figura a seguir demonstra, o efeito relativo a este bit só é notado 1 ciclo após que este bit é alterado. O *Duty Cycle* permanece 100% até que este bit volte ao estado lógico zero.

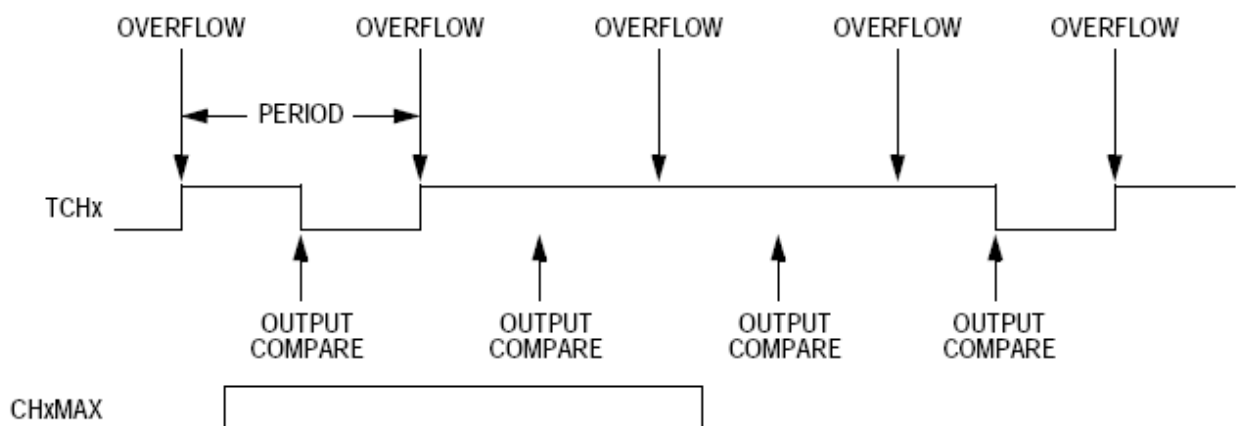


Figura – Latência do bit CHxMAX

Os outros bits relativos a estes registradores são configurados a partir da tabela abaixo:

MSxB	MSxA	ELSxB	ELSxA	Modo	Configuração
X	0	0	0	Porta I/O	Nível de saída inicial alto
X	1	0	0		Nível de saída inicial baixa
0	0	0	1	Captura de Entrada	Captura somente na borda de subida
0	0	1	0		Captura somente na borda de descida
0	0	1	1		Captura em ambas as Bordas
0	1	0	1	Comparação de saída ou PWM	Inverte nível na comparação
0	1	1	0		Nível para baixo na comparação
0	1	1	1		Nível para cima na comparação
1	X	0	1	Comparação de saída ou PWM com Buffer	Inverte nível na comparação
1	X	1	0		Nível para baixo na comparação
1	X	1	1		Nível para cima na comparação

Tabela – Seleção de modo, nível e borda

O registrador onde é realizada a configuração da largura de pulso é o TCHx, sendo TCHxH o registrador de maior significado e TCHxL o de menor significado na palavra de 16 bits que irá representar a largura do pulso. Abaixo estes registradores são apresentados.

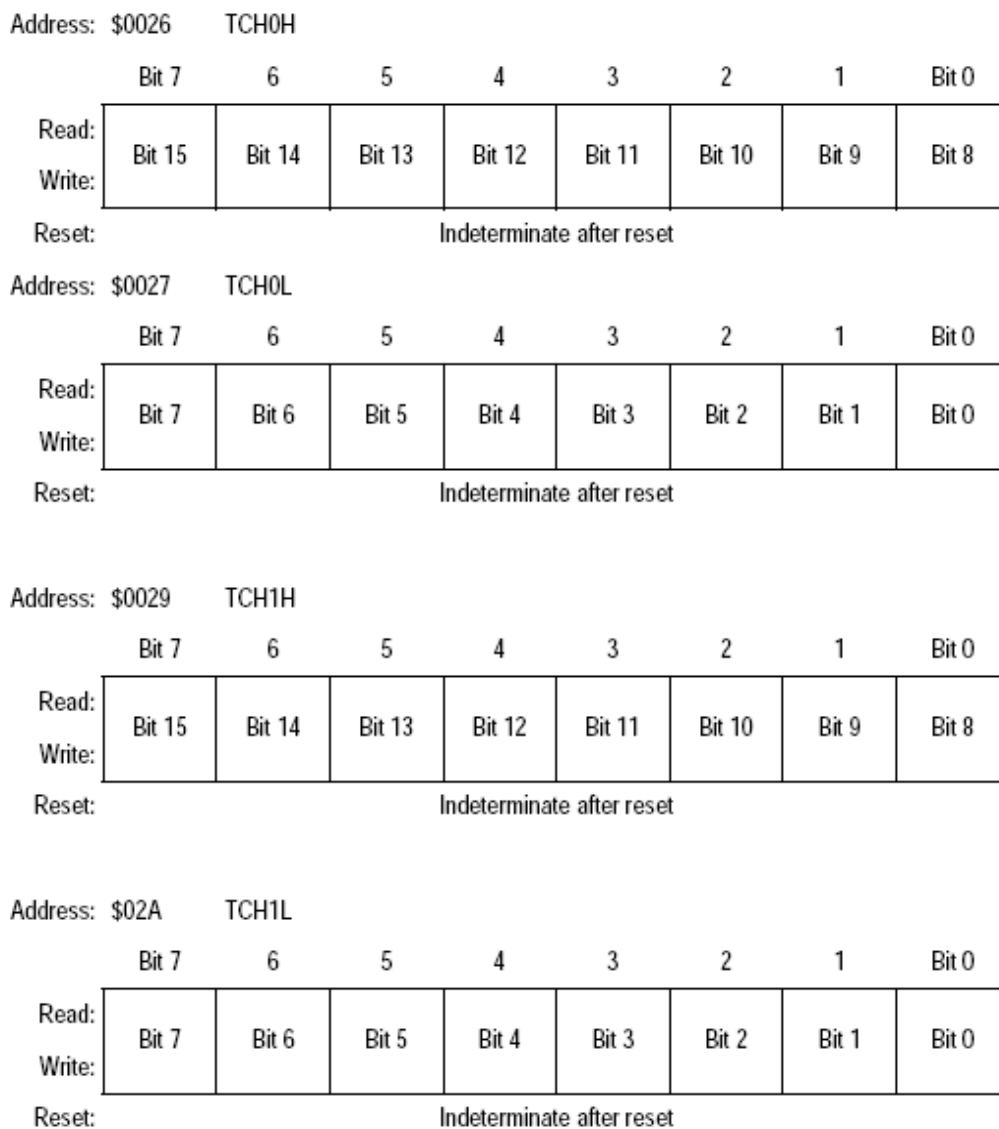


Figura – Registradores TCH0H, TCH0L, TCH1H e TCH1L.

Voltando ao exemplo onde deseja-se configurar um PWM de 10KHz com largura de pulso inicial igual a 40% e ciclo inicial ativo, utilizando um relógio de 3,2MHz. O módulo referente ao período do PWM já foi definido, sendo igual a 320. Para configurarmos a largura inicial para 40% deve-se utilizar 40% do valor do módulo do período encontrado, ou seja,  $TCH0 = 320 \times 40\% = 128$ .

A seguir será apresentado a configuração do PWM do exemplo acima, utilizando modo sem *buffer* com saída no canal zero, tanto para Assembly quanto para “c”.

### Assembly:

```
; Configuração do módulo de tempo
LHDX    #!320          ; Valor relativo ao período de 100us
STHX    TMD            ; Move para o registrador de módulo do temporizador
LDHX    #!128          ; Valor de 40% de largura de pulso (40% de 320)
STHX    TCH0           ; Move para o registrador do canal 0
CLRH                    ; Limpa a parte alta do registrador de índice, utilizado anteriormente
; Saída PWM no canal 0, com alteração para nível lógico baixo na comparação
; CH0F CH0IE MS0B MS0A ELS0B ELS0A TOV0 CH0MAX
; 0      1      0      1      1      0      1      0
MOV     #$5A,TSC0
; Período do PWM igual a 100us
; TOF TOIE TSTOP TRST Reservado PS2 PS1 PS0
; 0      1      0      0      0      0      0      0
MOV     #$40,TSC
```

“c”:

```
TMOD = 320;          /* Período do PWM */
TCH0 = 128;          /* 40% de PWM */

/* Saída PWM no canal 0, com alteração para nível lógico baixo na comparação
CH0F CH0IE MS0B MS0A ELS0B ELS0A TOV0 CH0MAX
  0      1      0      1      1      0      1      0      */
TSC0 = 0x5A;

/* Período do PWM igual a 100us
TOF TOIE TSTOP TRST Reservado PS2 PS1 PS0
  0      1      0      0      0      0      0      0      */
TSC = 0x40;          /* Não divide o clock */
```

Apesar de termos definido as inicializações, se o modo sem *buffer* está sendo utilizado, precisamos implementar também no código as interrupções de estouro de tempo e comparação de saída para ser possível alterar o valor da largura de pulso desta saída.

### Assembly:

```
; Interrupção do temporizador.
TOVER: BCLR    7,TSC          ; Limpa a flag da interrupção do temporizador
        BRCLR  condição,SAIR_OVER ; Verifica a necessidade de aumentar a largura
                                   ; Se a flag estiver com zero, não corrige
        BCLR   AUM,FLAG        ; Limpa a flag da necessidade de aumentar
        MOV    Valor_corrigido,TCH0 ; aumenta a largura de pulso
SAIR_OVER
        RTI                    ; Retorna da interrupção
```

```

; Interrupção da comparação
COMP:  LDA    TSC0
      BCLR    7,TSC0          ; Limpa a flag da interrupção da comparação
      BRCLR   condição,SAIR_COMP ; Verifica a necessidade de diminuir a largura
                                   ; Se a flag estiver com zero, não corrige
      BCLR    DIM,FLAG        ; Limpa a flag da necessidade de diminuir
      MOV     Valor_corrigido,TCH0 ; diminui a largura de pulso
SAIR_COMP
      RTI                    ; Retorna da interrupção
“c”:

// Interrupção do temporizador
interrupt void tover(void) {
    TSC_TOF = 0;              // Limpa a flag da interrupção do temporizador
    if (condição){           /* Indicação para aumenta largura do PWM*/
        flag_aum = 0;         // Limpa flag de condição
        TCH0 = Valor_corrigido; // Corrige a largura do pulso
    }
}                               // Retorna da interrupção

// Interrupção da comparação
interrupt void comparacao(void){
    byte i;
    i = TSC0;                 // Lê registrador de estado do PWM
    TSC0_CH0F = 0;           // Limpa a flag
    if (condição) {          // Verifica se é necessário diminuir a largura
        flag_dim = 0;         // Limpa flag de condição
        TCH0 = Valor_corrigido; // Corrige a largura do pulso
    }
}

```

Em algumas aplicações é necessário a inserção de um tempo morto (*dead time*) devido ao tempo de comutação de certos tipos de componentes utilizados ou devido as topologias dos sistemas agregados a estas saídas PWM.

Normalmente estes tempos mortos são inseridos em pares complementares de saídas PWM, ou seja, quando uma das saídas comuta para nível lógico alto, a outra saída comuta para nível lógico baixo. Abaixo é apresentado um exemplo de utilização de tempos mortos.

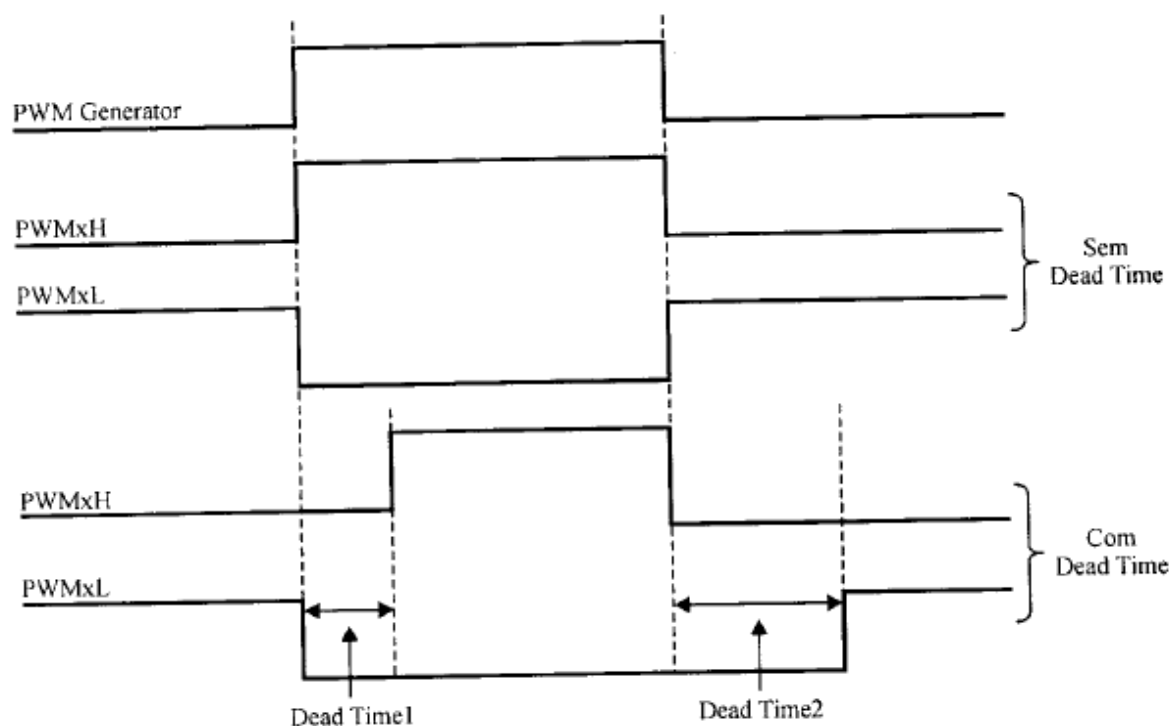


Figura – Exemplo de utilização de tempo morto

### 3.3 Conversores Analógico-Digital e Digital-Analógico

De um modo geral, os sinais encontrados no mundo real são contínuos (ou analógicos, pois variam no tempo de forma contínua), como, por exemplo: a intensidade luminosa de um ambiente que se modifica com a distância, a aceleração de um carro de corrida, etc. Os sinais manipulados por computadores e sistemas embarcados são digitais, como por exemplo, uma faixa de áudio lida de um compact disk.

A conversão analógico-digital (A/D) é o processo que possibilita a representação de sinais analógicos no mundo digital. Desta forma é possível utilizar os dados extraídos do mundo real para cálculos ou operar seus valores.

Em geral, o conversor A/D está presente internamente nos processadores e controladores de sinais digitais e alguns microcontroladores, mas também existem circuitos integrados dedicados a este fim.

Basicamente é um bloco que apresenta portas de entrada e saída. A entrada recebe sinais elétricos de forma contínua e possui uma faixa de tensão de entrada máxima e mínima. Nos microcontroladores que possuem um conversor A/D e operam na faixa de 5V, geralmente a faixa de tensão aceita sinais elétricos entre -5V e +5V.

Na saída o sinal é amostrado em um dado intervalo de tempo fixo (determinado pela frequência de amostragem). Esta amostra disponibiliza um certo valor que representa o sinal original naquele momento (quantização). As características de quantização estão relacionadas à precisão do conversor.

Para ilustrar esta situação, imagine que você queira mostrar a temperatura de um forno em um *display* de cristal líquido (LCD). Para isto seriam necessários alguns componentes eletrônicos. Os mais expressivos são: um transdutor (sensor de temperatura), um *display* de cristal líquido (LCD), um processador digital e um conversor analógico digital.

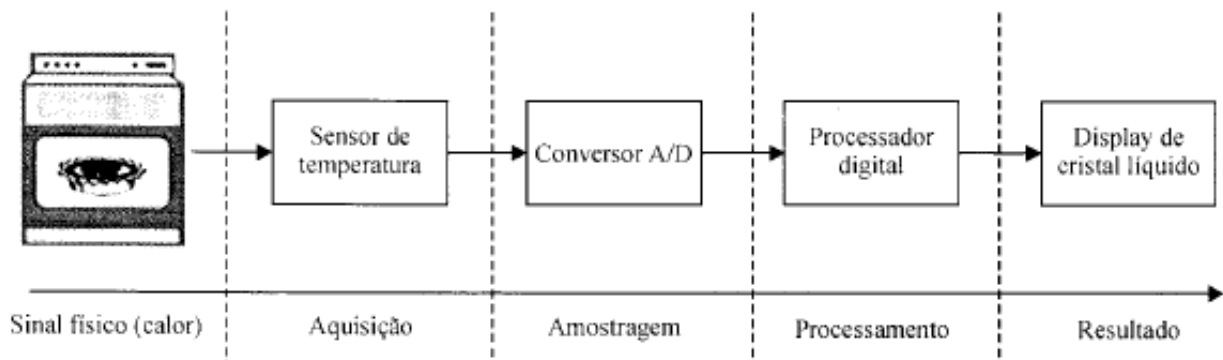


Figura - Diagrama de blocos de uma conversão A/D de um sinal de temperatura

A temperatura é um sinal analógico. O sensor de temperatura converte a temperatura em um sinal de impulsos elétricos analógicos. O conversor A/D recebe esse sinal e o transforma em sinal digital, através de amostragem, entregando ao processador. Este, por sua vez, manipula esses dados e envia-os para o *display*, mostrando em graus a temperatura do forno. A figura abaixo mostra a representação do sinal analógico de temperatura e seu equivalente na forma digital.

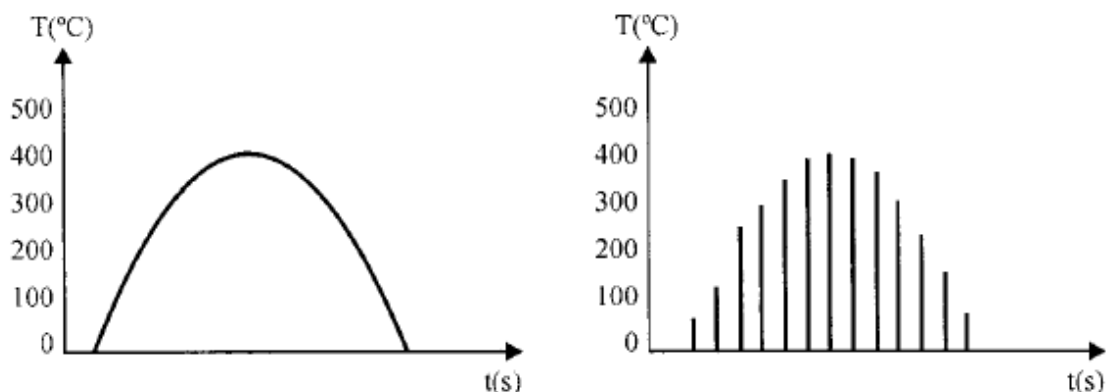


Figura – Representação de um sinal de temperatura analógico e digital.

A informação digital é diferente de sua forma original contínua em dois aspectos fundamentais:

- É amostrada porque é baseada em amostragens, ou seja, são realizadas leituras em um intervalo fixo de tempo no sinal contínuo;
- É quantizada porque é atribuído um valor proporcional a cada amostra.

Explorando um pouco mais o caso do forno, a figura abaixo detalha um pouco mais as três etapas mais importantes do processo: a aquisição, a amostragem e o processamento.

Neste diagrama de blocos, o sinal analógico é capturado pelo transdutor (sensor), em seguida passa por um filtro, denominado de *anti-alias*, a fim de diminuir os ruídos. A chave representa a frequência de amostragem do conversor A/D, entregando ao processador o sinal digitalizado.

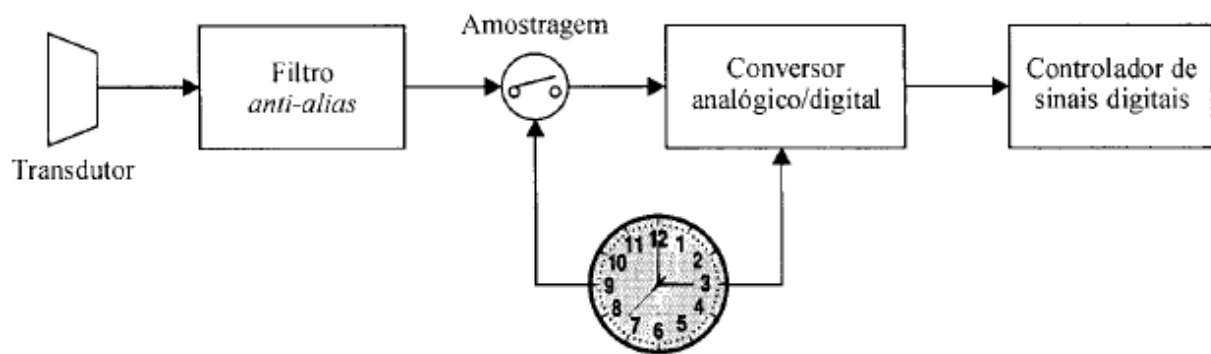


Figura – Diagrama de blocos da conversão A/D.

A frequência de amostragem é o número de amostras capturadas em um segundo. Esta frequência é dada em *Hertz* (Hz) e é considerada adequada quando se pode reconstruir o sinal analógico razoável a partir de amostras obtidas na conversão.

A taxa de conversão ou frequência de amostragem é de suma importância para o processamento de sinais reais. Para obter uma taxa de amostragem adequada pode-se utilizar os teoremas de *Nyquist* ou *Shannon*. Estes teoremas indicam que um sinal contínuo  $x(t)$  pode ser amostrado adequadamente se tiver banda limitada, ou seja, seu espectro de frequência não pode conter frequências acima de um valor máximo ( $F_{máx}$  – frequência máxima). Ainda, outro ponto importante é que a taxa de amostragem ( $F_a$  – Frequência de amostragem) deve ser escolhida para ser no mínimo duas vezes maior que a frequência máxima ( $F_{máx}$ ). Por exemplo, para representar um sinal de áudio com



freqüências até 10 KHz, o conversor A/D deve amostrar esses sinais utilizando uma freqüência de amostragem de no mínimo 20 KHz.

Para melhor entendimento, vamos ver como funciona um conversor A/D de 4 bits (Figura abaixo).

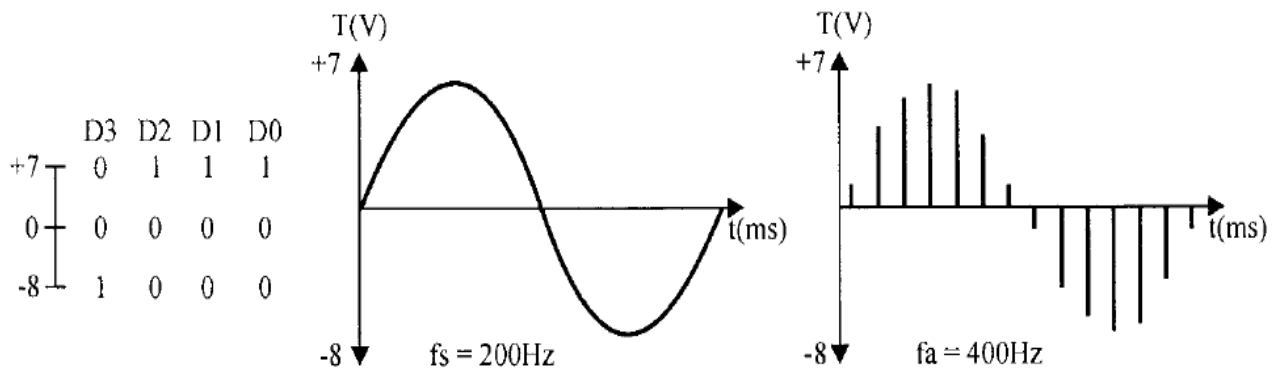


Figura – Conversão A/D de 4 bits.

Com 4 bits o máximo representável é o número 16. Isso quer dizer que temos uma faixa de 0 a 15 (não sinalizado) ou +7 a -8 (sinalizado). Nesse conversor fictício, teremos uma variação a cada 1 volt. A figura anterior mostra um sinal de áudio de 200 Hz variando de +7 a -8 volts, que pode ser capturado por um microfone. Conforme o teorema de *Nyquist*, seria necessário uma freqüência de amostragem de 400 Hz.

Lembrando que, se o sinal de áudio possuir amplitude maior que a faixa representável do conversor A/D (7V a -8V), então não seria possível converter tal sinal.

O conversor D/A possui todas as características do conversor A/D, as quais diferem apenas porque o conversor D/A pega um sinal digital e transforma em analógico. Por exemplo, em uma aplicação de áudio, um microfone captura o áudio e envia a um conversor A/D, que entrega o sinal amostrado e quantizado a um processador digital. Este último efetua diversas operações com o sinal de áudio. Só então o processador envia ao conversor D/A, para remontar o sinal analógico a partir do sinal digital, para ser reproduzido em um alto-falante. Um exemplo de conversor D/A de 16 bits é o DAC1221, da *Texas Instruments*.

Novamente para exemplificar a configuração de um conversor A/D em um microcontrolador será utilizado o microcontrolador MC68HC908QY4. Este microcontrolador possui um conversor A/D por aproximação sucessiva linear com quatro canais, ou seja, existem 4 entradas possíveis para sinais analógicos que são multiplexadas para um único conversor A/D. Isto implica em que só um canal será convertido em um determinado momento. A resolução deste A/D é de 8 bits, no entanto, a

*Freescale* disponibilizou uma nova versão com A/D de 10 bits, o MC68HC908QY4A. Na figura a seguir é apresentado o diagrama de blocos do conversor A/D deste microcontrolador.

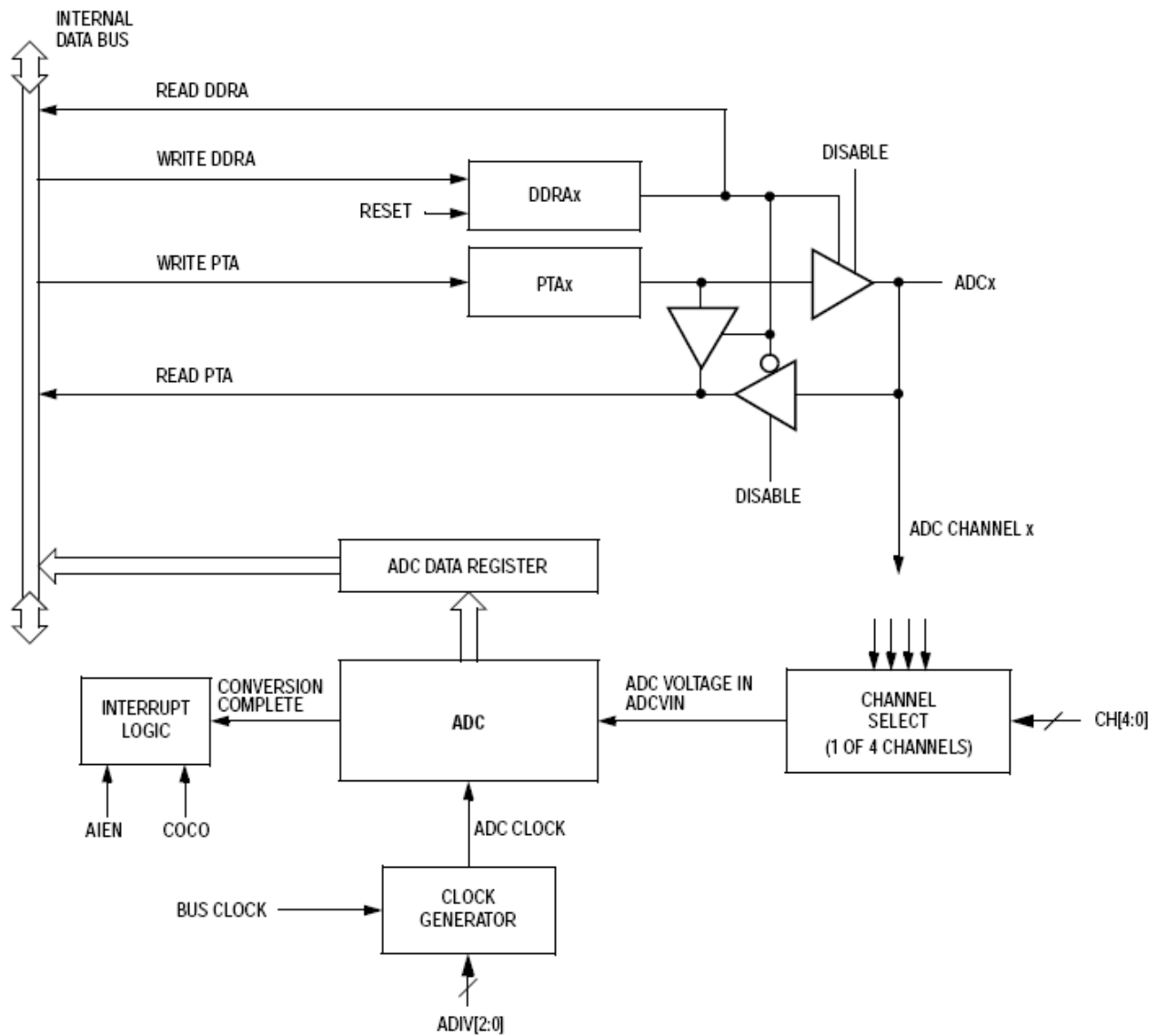


Figura – Diagrama de blocos do conversor A/D do microcontrolador MC68HC908QY4