MOTOROLA
intelligence everywhere

digital dna

Baseado na Família
MC68HC908QT/QY

Programa Embedded Software

Treinamento de Microcontroladores

Elaborado por
CNZ Engenharia
www.cnz.com.br

WWW.MOTOROLA.COM/SEMICONDUCTORS

ÍNDICE

Lista de Figuras

Lista de tabelas

1. INTRODUÇÃO

A evolução rápida da eletrônica digital, dos microprocessadores e, em particular, dos microcontroladores provocou uma revolução no cotidiano das pessoas. Nos afazeres domésticos diários, na condução de um veículo, no cenário visual da cidade e, também nos mais variados equipamentos que estão a nossa disposição no trabalho ou na escola encontram-se soluções integradas ("*embbeded*") que utilizam microcontroladores. A inteligência incorporada às máquinas está presente em todos os lugares, e a qualquer momento. Estima-se que, em 2010, em média uma pessoa interagirá com 350 dispositivos com microcontroladores diariamente.

A família Motorola M68HC08, comumente denominada HC08, contém microcontroladores de propósito geral com largas possibilidades de aplicação. Este documento fornecerá aos seus leitores uma introdução à arquitetura dos microcontroladores HC08, bem como o conjunto de instruções para programação utilizando o código fonte (linguagem *Assembly*).

Dentre os inúmeros integrantes da família HC08, os mais econômicos e compactos, que podem ser dedicados a aplicações em que custo e espaço são fundamentais são denominados de MC68HC908QT/QY, e serão apresentados em detalhes neste documento. É importante ressaltar que todos os conceitos abordados são válidos para toda a família de microcontroladores HC08.

Os leitores deverão ler este documento cuidadosamente e completamente, pois as informações e instruções nele contidas são essenciais para a parte prática (laboratórios). Por se tratar de material didático para treinamento básico de microcontroladores da família HC08 serão abordados tópicos referentes ao modelo de programação, modos de endereçamento e o conjunto de instruções completo. Serão abordados, também, os principais periféricos dos microcontroladores da família MC68HC908QT/QY: temporizador interno, conversor A/D, interrupções, sistema monitor e BREAK.

Utilize os documentos técnicos da Motorola como referência ao trabalhar com um componente específico da família HC08, e em especial ao necessitar de detalhes de implementação dos seus periféricos e respectivos registradores. Eles estão disponíveis no website: http://e-www.motorola.com, e podem ser acessados através de uma busca pelo "*part number*" do componente.

2. REVISÃO

Um microcontrolador é um sistema computacional completo, no qual estão incluídos uma CPU (Central Processor Unit), memória, um sistema de clock, sinais 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 (*chip*). 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 (software) para que o sistema faça alguma coisa útil

2.1. Unidade Central de Processamento (CPU)

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 seqüência programada, para uma aplicação específica. Um programa computacional (software) instrui a CPU a ler informações de entradas, ler e escrever informações na memória de trabalho, e escrever informações nas saídas. O diagrama de blocos simplificado da CPU presente nos microcontroladores da família HC08, também denominada CPU08 é apresentada na Figura 1.

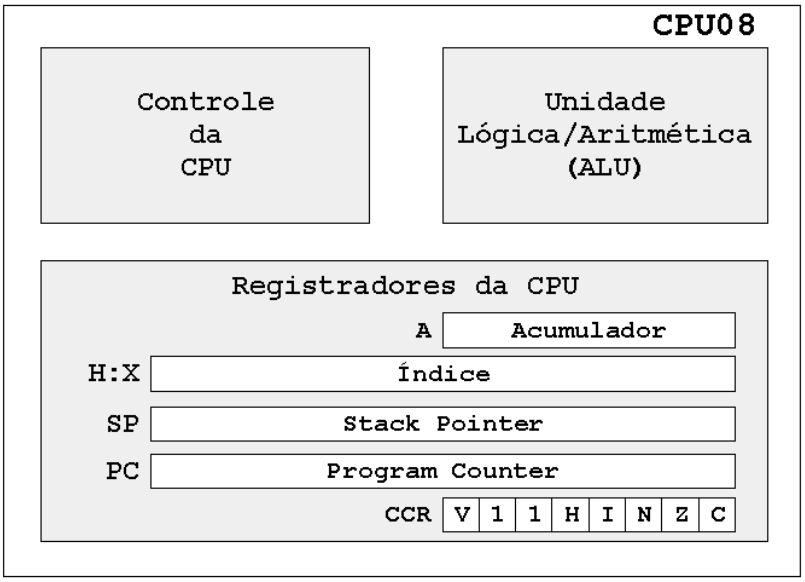

Figura 1 – Diagrama de blocos da CPU08

Unidade Lógica/Aritmética (ALU): A ALU é usada para realizar operações lógicas e aritméticas definidas no conjunto de instruções da CPU08. Vários circuitos implementam as operações aritméticas binárias decodificas pelas instruções e fornecem dados para a execução da operação na ALU. 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 ALU sob controle lógico da CPU.

Controle da CPU: O circuito de controle da CPU implementa o sequeciamento de elementos lógicos necessários à ALU 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* 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 1. Os registradores da CPU são memórias dentro do microprocessador (que não fazem parte do mapa de memória). O conjunto de registradores da CPU é freqüentemente chamado de **modelo de programação**.

O **registrador A**, é também chamado de acumulador porque 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 alto do registrador de índice é denominado H, e o byte baixo denominado X. Sua principal função é servir de apontador para uma área na memória onde a CPU irá carregar (ler) ou armazenar (escrever) informação. Serão apresentados mais detalhes do registrador H:X quando forem discutidos os modos de endereçamento indexado. 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 resetada, 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 status (*flags*) que refletem o resultado de algumas operações da CPU. A instruções de desvio usam estes bits de status para tomar suas decisões.

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).

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 seqüência pré-determinada.

Uma fonte de clock de alta freqüência (normalmente derivada de um cristal ressonador conectado a CPU) é usado para controlar o sequeciamento das instruções da CPU. Normalmente as CPUs dividem a freqüência básica do cristal por 2 ou mais para chegar ao clock do barramento interno. Cada ciclo de leitura ou escrita a memória levam 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 de endereços, ele pode endereçar 2^n posições de memória (p.ex.: um sistema com 11 linhas pode acessar $2^{11} = 2048$ 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 ROM*) – 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 ROM*) – 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 10.000 vezes.

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

I/O (*Input/Output*) – Registradores de controle, status 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 V_{SS}) indica o nível lógico 0 e o sinal de fonte positiva, que tipicamente é $+5V_{DC}$ (V_{DD}) indica o nível lógico 1.

Naturalmente que no mundo real existem sinais puramente analógicos (com uma infinidade de valores) ou sinais que utilizam outro nível de tensão. Alguns dispositivos de entrada traduzem as tensões do sinal para níveis compatíveis com V_{DD} e V_{SS}. 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 V_{DD} e V_{SS} 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 seqüência 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 $4C é entendido pela CPU, mas não é significativo para nós humanos. Para resolver esse problema, um sistema de instruções mnemônicas equivalentes são usadas. O *opcode* $4C corresponde ao mnemônico INCA, lê-se “incrementa o acumulador”, que é muito mais inteligível. Para fazer a translação de mnemônicos em **códigos de máquina** (*opcodes* e outras informações) utilizados pela CPU é necessário um programa computacional chamado **assembler**. 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.

3. FAMÍLIA HC08

3.1. Características principais

A família Motorola M68HC08 contém CPUs de 8 bits (CPU08) que tem uma organização específica denominada arquitetura Von Neumann. Nesta arquitetura, a CPU e a memória são conectadas por um **barramento de endereço** (*address bus*) e um **barramento de dados** (*data bus*). O barramento de endereços é usado para identificar qual posição de memória está sendo acessada, e o barramento de dados é usado para enviar uma informação da CPU para um endereço de memória, ou de um endereço de memória para a CPU.

Na implementação desta arquitetura pela Motorola, existem algumas localizações de memória (denominadas registradores da CPU) internas a CPU, que atuam como pequenas áreas de rascunho e como painel de controle da CPU. Estes registradores formam o **modelo de programação** da CPU e são similares a endereços de memória, pois as informações podem ser escritas ou lidas neles.

Toda informação processada pela CPU que não esteja diretamente ligada ao **modelo de programação** pode ser vista como uma lista de endereços. Esta organização é chamada freqüentemente de sistema de **I/O mapeado em memória** (*memory-mapped I/O*) porque a CPU trata como endereços de memória instruções do programa, variáveis do sistema, ou controles de entrada/saída (I/O).

Entre as características principais da CPU, da família HC08 incluem-se:

- ✓ Código objeto completamente compatível com a família HC05
- ✓ CPU com freqüência de barramento interno de até 8MHz
- ✓ 64 Kbytes endereçáveis para memória de programa, dados e periféricos
- ✓ Barramento interno flexível para acessar periféricos
- ✓ Modos de baixo consumo STOP e WAIT
- ✓ Registrador de 16 bits para *Stack Pointer* com instruções de manipulação da pilha
- ✓ Registrador de 16 bits de Índice (H:X) com manipulação do byte alto (H) e baixo (X)
- ✓ 16 modos de endereçamento
- ✓ Movimentação de dados da memória para memória sem utilizar acumulador
- ✓ Instruções rápidas de multiplicação de 8 bits por 8 bits e divisão de 16 bits por 8 bits
- ✓ Instruções avançadas para manipulação de dados em BCD

3.2. Modelo de Programação

Diferentes CPUs tem diferentes conjuntos de registradores. As diferenças são primariamente o número e o tamanho dos registradores. A Figura 2 mostra os registradores da CPU que se encontram nos microcontroladores da família HC08. Apesar de serem poucos os registradores que fazem parte do modelo de programação, eles são representativos de todos os tipos de registradores da CPU08 e podem ser usados para explicar todos os conceitos fundamentais de programação.

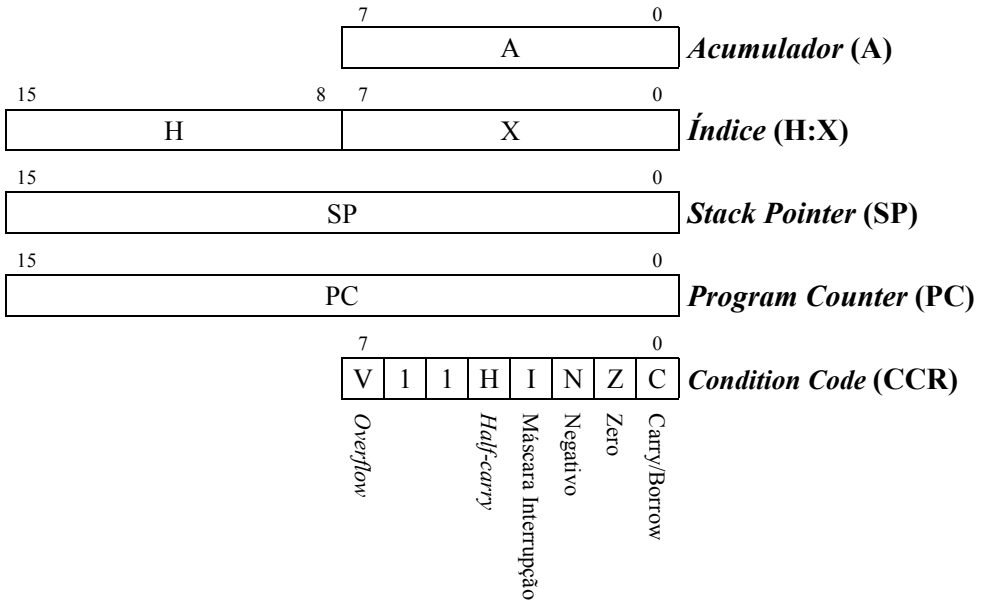

Figura 2 – Modelo de Programação da CPU08

3.2.1. Acumulador (A)

	Bit 7	6	5	4	3	2	1	Bit 0
A								
RESET	X	X	X	X	X	X	X	X

X - Indeterminado

Figura 3 – Acumulador (A)

O acumulador é um registrador de 8 bits de uso geral. A CPU utiliza o acumulador para armazenar os operandos e resultados de operações aritméticas e não aritméticas.

3.2.2. Registrador de Índice (H:X)

	Bit15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	H								X							
H:X																
RESET	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X

X – Indeterminado

Figura 4 – Registrador de Índice (H:X)

O registrador de 16 bits de índice permite ao usuário endereçar indiretamente o espaço de memória de 64Kbytes. A concatenação do registrador de 16 bits é denominada H:X. O byte alto do registrador de índice é denominado H, e o byte baixo denominado X.

Nos modos de endereçamento indexado, a CPU utiliza o conteúdo de H:X para determinar o endereço efetivo do operando. H:X também podem servir como registradores temporários para armazenamento de dados.

3.2.3. Stack Pointer (SP)

	Bit15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
SP																
RESET	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figura 5 – Registrador *Stack Pointer* (SP)

O registrador de 16 bits *Stack Pointer* contém o endereço da próxima posição livre na pilha. Durante um reset, o *Stack Pointer* contém o endereço $00FF. A instrução RSP (Reseta *Stack Pointer*) seta o byte menos significativo com $FF 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 resetada, o SP aponta para um endereço específico na memória RAM ($00FF).

A CPU08 tem modos de endereçamento indexado com *offset*s de 8 ou 16 bits do SP para ser utilizado no acesso de variáveis temporárias inseridas na pilha. A CPU utiliza o conteúdo do registrador SP para determinar o endereço efetivo do operando.

> ***NOTA**: Embora o endereço inicial do SP seja $00FF, a localização da pilha é arbitrária e pode ser realocada pelo usuário em qualquer lugar na RAM. Movimentar o SP para fora da página 0 ($0000 a $00FF) irá abrir espaço de memória que pode ser acessado usando modos de endereçamento mais eficientes.*

3.2.4. Program Counter (PC)

	Bit15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
PC																
RESET	Carregado com o dado do vetor $FFFE								Carregado com o dado do vetor $FFFF							

Figura 6 – Registrador *Program Counter* (PC)

O *Program Counter* é um registrador de 16 bits que contém o endereço da próxima instrução ou operando a ser capturado ("*fetched*") na memória de programa.

Normalmente, o conteúdo do PC é automaticamente incrementado para o próximo endereço toda vez que uma instrução ou operando é capturado. Operações de desvios, chamadas à subrotinas ou interrupções carregam o PC com um endereço diferente do endereço seqüencial.

Durante o reset, o PC é carregado com o conteúdo do vetor de reset localizado nos endereços $FFFE e $FFFF. Eles contém o endereço da primeira instrução a ser executada depois que a CPU08 sai do estado de reset.

3.2.5. Condition Code Register (CCR)

	Bit 7	6	5	4	3	2	1	Bit 0
CCR	**V**	**1**	**1**	**H**	**I**	**N**	**Z**	**C**
RESET	X	1	1	X	1	X	X	X

X – Indeterminado

Figura 7 – Registrador *Condition Code* (CCR)

O registrador CCR contém um bit para mascarar interrupções e 5 bits que indicam o resultado da instrução executada naquele instante. Os bits 5 e 6 são permanentemente setados. A descrição mais detalhada de cada bit é apresentada a seguir:

3.2.5.1. V – Bit de Overflow

A CPU set o bit de *overflow* quando houver transbordo no resultado de uma operação em complemento de 2. O bit V é utilizado pelas instruções de desvios condicionais BGT, BGE, BLE, e BLT.

3.2.5.2. H – Bit de Half-carry

A CPU seta o bit de *half-carry* quando ocorrer transbordo 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.

3.2.5.3. I – Mascara Interrupção

Quando o bit I está setado, todas as interrupções são mascaradas. As interrupções são habitadas quando o bit I é resetado. Quando ocorre uma interrupção, o bit que mascara as interrupções é automaticamente setado depois que os registradores da CPU são armazenados na pilha, mas antes que o vetor de interrupção seja capturado.

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 zero.

A instrução retorno da interrupção (RTI) retorna os registradores da CPU da pilha, e restaura o bit I no seu estado zerado. Após qualquer reset, o bit I é setado e só pode ser resetado por uma instrução de software.

3.2.5.4. N – Bit Negativo

A CPU seta o bit N quando uma operação aritmética, lógica ou de manipulação de dados produzir um resultado negativo. Corresponde ao 8º bit do registrador que contém o resultado.

3.2.5.5. Z – Bit Zero

A CPU seta o bit Z quando uma operação aritmética, lógica ou de manipulação de dados produzir um resultado igual a $00.

3.2.5.6. C – Bit Carry/Borrow

A CPU seta o bit C quando uma operação de adição produzir um transbordo do bit 7 do acumulador ou quando uma subtração necessitar um empréstimo. Algumas operações lógicas e as instruções de manipulação de dados também podem resetar ou setar o bit C.

3.3. Modos de endereçamento

Os modos de endereçamento indicam o caminho pelo qual a CPU obtém as informações necessárias para completar uma instrução. A CPU08 tem um total de 16 modos de endereçamento, alguns deles implementados para gerar um código eficiente quando o desenvolvimento de software for realizado com linguagens de alto nível.

A seguir são apresentados todos os modos de endereçamento da família HC08.

3.3.1. Inerente

As instruções são formadas por um *opcode* que contém o operando implícito.

Exemplo: **DECA** – Decrementa conteúdo do acumulador.

3.3.2. Imediato

As instruções contém o *opcode* seguido por um operando byte (8 bits) ou word (16 bits) cujo conteúdo é um valor imediato.

Exemplo: **LDA #$20**[1] – Carrega o acumulador com o valor **$20**.

3.3.3. Direto

As instruções de endereçamento direto são formadas por um *opcode* seguido por um operando, cujo conteúdo é um endereço de 8 bits. Este modo de endereçamento acessa apenas os 256 primeiros bytes da memória (página zero).

Exemplo: **LDA $40** – Carrega o acumulador com o conteúdo do endereço **$0040**.

3.3.4. Estendido

As instruções de endereçamento estendido são formadas por 3 bytes: 1 byte para o *opcode* e 2 bytes para um endereço qualquer dos 64 Kbytes de memória. A maioria dos montadores *Assembly* pode utilizar automaticamente o modo de endereçamento direto quando o operando estiver na primeira página (endereços $00XX).

Exemplo: **LDA $45FA** – Carrega o acumulador com o conteúdo do endereço **$45FA**.

---

[1] *O caracter "#" utilizado como prefixo de um operando é indicativo do modo imediato. O caracter "**$**" utilizado como prefixo de um valor indica que o mesmo está em notação hexadecimal.*

3.3.5. Indexado

Os modos de endereçamento indexado são a chave para a geração de código eficiente para pesquisa à tabelas e outras estruturas de dados. O modo de endereçamento indexado sem *offset* é conhecido popularmente por outras arquiteturas de microcontroladores como "endereçamento indireto". O valor presente no operando das instruções com o registrador de índice é um endereço (ponteiro).

- ✓ **Sem *offset***

  Exemplo: **LDA ,X** – Carrega o acumulador com o conteúdo do endereço armazenado no registrador **H:X**.
- ✓ **Com *offset* de 8 bits**

  Exemplo: **LDA $5E,X** – Carrega o acumulador com o conteúdo do endereço armazenado em **(H:X + $5E)**.
- ✓ **Com *offset* de 16 bits**

  Exemplo: **LDA $485E,X** – Carrega o acumulador com o conteúdo do endereço armazenado em **(H:X + $485E)**.
- ✓ **Sem *offset* e com pós-incremento**

  Exemplo: **CBEQ X+,TAG** – Compara o conteúdo de A com o conteúdo do endereço armazenado em H:X, salta para TAG quando igual, e posteriormente, incrementa **X**.
- ✓ **Com *offset* de 8 bits e pós-incremento**

  Exemplo: **CBEQ $50,X+,TG1** - Compara o conteúdo de A com o conteúdo do endereço armazenado em (H:X + $50), salta para TAG quando igual, e posteriormente, incrementa **X**.

3.3.6. Stack Pointer

Existem dois tipos de endereçamento do *Stack Pointer*: com 8 bits ou 16 bits de *offset*. Eles são similares aos modos de endereçamento indexados, e utilizam o registrador SP ao invés do par de registradores H:X.

- ✓ **Com *offset* de 8 bits**

  Exemplo: **LDA $48,SP** – Carrega o acumulador com o conteúdo do endereço armazenado em **(SP + $48)**.
- ✓ **Com *offset* de 16 bits**

  Exemplo: **LDA $485E,SP** – Carrega o acumulador com o conteúdo do endereço armazenado em **(SP + $485E)**.

3.3.7. Relativo

Todas as instruções de desvio condicional utilizam endereçamento relativo. Se a condição for verdadeira, o conteúdo do registrador PC é adicionado a um valor com sinal (8 bits) que está presente como operando da instrução. Isto fornece uma faixa para o desvio que varia de –128 bytes a + 127 bytes em relação ao endereço da instrução seguinte a instrução de desvio.

Exemplo: **BCC** ***Volta*** – Desvia para o endereço ***Volta*** se o *flag* de carry (C) estiver resetado.

3.3.8. Movimento de Dados de Memória para Memória

Existem quatros modos de endereçamento de memória para memória com a instrução de movimentação, MOV, para transferir dados de um endereço para outro sem utilizar o acumulador.

- ✓ **Imediato para direto**

  Exemplo: **MOV #$40,$25** – Movimenta o valor imediato **$40** para o endereço **$25.**

- ✓ **Direto para direto**

  Exemplo: **MOV $40,$25** – Movimenta o conteúdo do endereço **$40** para o endereço **$25.**

- ✓ **Indexado para direto com pós-incremento**

  Exemplo: **MOV X+,$23** - Movimenta o conteúdo do endereço armazenado no registrador **H**:**X** para o endereço **$0023** e, posteriormente, incrementa **H**:**X**.

- ✓ **Direto para indexado com pós-incremento**

  Exemplo: **MOV $23,X+** - Movimenta o conteúdo do endereço **$0023** para o registrador **H:X** e, posteriormente, incrementa **H:X**.

O modelo de programação associado a instruções do modo indexado, *Stack Pointer* e de desvios condicionais foram projetados para gerar código objeto eficiente quando utilizados com compiladores de linguagens de alto nível (HLL – High Level Language), como por exemplo a Linguagem C.

3.4. Modos de Baixo Consumo

As instruções WAIT e STOP colocam o microcontrolador em um modo de baixo consumo de energia.

3.4.1. Modo WAIT

A instrução WAIT executa as seguintes tarefas:

- ✓ Reseta o bit de máscara da interrupção (Bit I do registrador CCR), habilitando interrupções. Depois de sair do modo WAIT por interrupção, o bit I permanece resetado. Depois de sair por um reset, o bit I é setado.
- ✓ Desabilita o clock da CPU, enquanto que os clocks dos periféricos continuam em execução.

Um módulo que estiver ativo durante o modo WAIT pode despertar a CPU com uma interrupção, se a interrupção estiver habilitada. O modo WAIT pode ser também finalizado por um reset ou um BREAK.

3.4.2. Modo STOP

A instrução STOP executa as seguintes tarefas:

- ✓ Reseta o bit de máscara da interrupção (Bit I do registrador CCR), habilitando interrupções externas. Depois de sair do modo STOP por interrupção externa, o bit I permanece resetado. Depois de sair por um reset, o bit I é setado.
- ✓ Desabilita o clock da CPU e dos periféricos
- ✓ Depois de sair do modo STOP, o clock da CPU começa a rodar depois do tempo de estabilização do oscilador.

Uma requisição de interrupção de um módulo pode causar a saída do modo STOP. O modo STOP pode também ser finalizado por um reset ou um BREAK.

Os sinais do oscilador (BUSCLKX2 e BUSCLKX4) param no modo STOP, parando a CPU e os periféricos. O tempo de recuperação do STOP é selecionável pelo bit SSREC, do registrador de configuração 1 (CONFIG1). Se SSREC = 1 o tempo de recuperação do STOP é reduzido de (4096 x BUSCLKX4) ciclos para 32. Este valor é ideal para utilização com o oscilador interno, oscilador RC, e oscilador externo que não precisam de tempos longos de estabilização.

3.5. Processamento de exceções

A CPU08, como em outros microcontroladores, executa as instruções seqüencialmente. Porém, em uma grande variedade de aplicações é necessário executar um conjunto de instruções em resposta a eventos excepcionais durante a execução de programas. Estes eventos freqüentemente são assíncronos em relação à execução do programa principal.

Um reset para a CPU e reinicializa a execução do programa (aplicação) a partir de sua condição inicial.

Uma interrupção não para a CPU ou a operação da instrução que está sendo executada, mas começa seu processamento quando a instrução corrente for completada. A CPU muda temporariamente a seqüência do programa para responder a um evento particular, em um endereço específico.

3.5.1. Reset

O **reset** é necessário para inicializar a CPU em um estado conhecido, incluindo o armazenamento no registrador *Program Counter* do endereço da primeira instrução a ser executada. Periféricos e diversos bits de controle e de status são também forçados para um estado conhecido como resultado do reset. Durante um reset ocorrem diversas ações internas a CPU, como seguem:

- ✓ Todos os registradores de direção são zerados (I/O configurados como entrada);
- ✓ *Stack Pointer* é inicializado com o endereço $00FF;
- ✓ Bit I do registrador CCR é setado para inibir as interrupções mascaráveis;
- ✓ *Flag* de interrupção externa é resetado;
- ✓ *Flag* do modo STOP é resetado;
- ✓ *Flag* do modo WAIT é resetado.

A interrupção do processamento durante a execução normal de um programa devido a um Reset gera as tarefas citadas acima e na saída do reset é feita a busca do vetor Reset e a reinicialização do programa.

3.5.2. Interrupções

As interrupções fornecem um meio de suspender temporariamente a execução normal do programa, de tal forma que a CPU tenha tempo livre para processar a requisição pedida.

Existem 2 categorias de interrupções: interrupções de hardware e interrupções de software.

Uma interrupção de software ocorre como resultado da execução da instrução SWI, e sempre é executada como parte do fluxo de instruções.

Uma interrupção de hardware pode ser gerada por eventos de internos (p.ex.: estouro do temporizador) ou externos (p.ex.: circuito lógico alterando estado do pino /IRQ). As interrupções de hardware são mascaráveis e, portanto podem ser reconhecidas apenas quando o bit I estiver resetado. As fontes mais comuns de interrupção de hardware são:

- ✓ Interrupção externa (/IRQ) - utilizada para monitorar sistemas ou eventos externos;
- ✓ Interrupções do temporizador - utilizadas para processar eventos baseados em contagem (hardware) ou referências de tempo;
- ✓ Interrupção de teclado - utilizadas para monitorar mudanças nos pinos de I/O.

O processamento de exceções é realizado através de tarefas discretas, também denominadas "Mudança de Contexto". As tarefas executadas durante o processamento são:

- ✓ Reconhecimento (*Recognition*): evento causador da interrupção, pode ser hardware ou software;
- ✓ Arbitramento (*Arbitration*): determinação das fontes de interrupção e priorização, no caso de haver mais de uma interrupção simultânea;
- ✓ Empilhamento (*Stacking*): armazenamento na pilha do contexto do programa atual;
- ✓ Busca do vetor (*Vector Fetching*): armazenamento no PC do vetor da interrupção;
- ✓ Rotina de Serviço da Interrupção (*Interrupt Service Routine*): execução da rotina de serviço da interrupção, a partir do endereço armazenado no PC;

Ao final da execução da rotina de serviço da interrupção (instrução RTI) ocorre a retirada dos dados da pilha (desempilhamento), retornando ao contexto do programa inicial. Se interrupções adicionais estiverem pendentes, o processo iniciará novamente.

Periféricos internos ao chip geram interrupções mascaráveis que são reconhecidas apenas se o bit de máscara global de interrupções (I) do registrador CCR estiver resetado (I=0). A interrupções mascaráveis são priorizadas de acordo com um arranjo pré-determinado.

4. FAMÍLIA MC68HC908QT/QY

4.1. Descrição Funcional

Os microcontroladores que fazem parte da família MC68HC908QT/QY tem como características básicas: baixo custo, alto desempenho e baixa pinagem (8 ou 16 pinos). Todos os membros dessa família utilizam a unidade central de processamento CPU08, desenvolvida para a arquitetura HC08, e estão disponíveis com uma variedade de módulos, tamanhos e tipos de memória, e tipos de encapsulamento. Os principais componentes estão apresentados na tabela a seguir:

Tabela 1 – Microcontroladores da família MC68HC908QT/QY

Dispositivo	Memória FLASH	Conversor A/D	Nº pinos
MC68HC908QT1	1536 bytes	-	8 pinos
MC68HC908QT2	1536 bytes	4 canais de 8 bits	8 pinos
MC68HC908QT4	4096 bytes	4 canais de 8 bits	8 pinos
MC68HC908QY1	1536 bytes	-	16 pinos
MC68HC908QY2	1536 bytes	4 canais de 8 bits	16 pinos
MC68HC908QY4	4096 bytes	4 canais de 8 bits	16 pinos

4.1.1. Características principais

- ✓ Núcleo da CPU de alto desempenho HC08
- ✓ Tensão de operação (V_{DD}) de 5V e 3 V
- ✓ 8MHz @5V (4MHz@3V) para operações do barramento interno
- ✓ Oscilador interno ajustável
  - ✓ 3.2MHz para operações do barramento interno
  - ✓ Capacidade de ajuste com registrador de 8 bits
  - ✓ Precisão de ±25% sem utilizar o ajuste
  - ✓ Precisão de ±5% com o ajuste
- ✓ Capacidade de acordar automaticamente da condição de STOP
- ✓ Programação "*in-system*" da memória FLASH
- ✓ Segurança da memória FLASH
- ✓ Memória FLASH programável na própria aplicação (com geração interna de tensões de apagamento/programação) com capacidades de 1,5K ou 4Kbytes
- ✓ 128 bytes de memória RAM

- ✓ Temporizador com 2 canais de 16 bits (Timer Interface Module – TIM)
- ✓ 4 canais de conversor A/D de 8 bits (ADC)
- ✓ 5 ou 13 linhas de entrada/saída (I/O) bidirecionais e mais 1 entrada:
  - ✓ 6 compartilhadas com função de interrupção do teclado e ADC
  - ✓ 2 compartilhadas com os canais do temporizador
  - ✓ 1 compartilhada com interrupção externa (IRQ)
  - ✓ 8 linhas extras no empacotamento de 16 pinos
  - ✓ Capacidade de alta corrente dreno/fonte em todos os pinos
  - ✓ Pull-ups selecionáveis individualmente em todos os pinos
  - ✓ Habilidade de "*tri-state*" em todos os pinos
- ✓ 6 bits de interrupção de teclado com característica de "despertar"
- ✓ Inibição por baixa tensão (*Low-Voltage Inhibit* – LVI)
  - ✓ Ponto de disparo selecionável por software no registrador CONFIG
- ✓ Sistema de proteção
  - ✓ Computador operando adequadamente (*Computer Operating Properly* – COP)
  - ✓ Detecção de baixa tensão com reset
  - ✓ Detecção de *opcode* ilegal com reset
  - ✓ Detecção de endereço ilegal com reset
- ✓ Interrupção externa assíncrona com pull-up interno (/IRQ) compartilhada com pino de I/O de propósito geral
- ✓ Reset assíncrono (/RST) compartilhado com pino de I/O de propósito geral
- ✓ Reset na energização (POR)
- ✓ Pull-ups internos no /IRQ e /RST para redução de componentes externos
- ✓ Registradores de I/O mapeados em memória
- ✓ Modos de redução de consumo – STOP e WAIT

4.1.2. Pinagem

Os componentes da família MC68HC908QT/QY estão disponíveis em 8 e 16 pinos nos encapsulamentos PDIP, SOIC ou TSSOP, conforme ilustrado nas figuras:

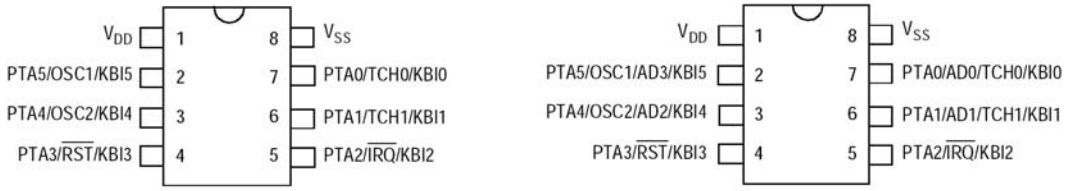

Figura 8 – Pinagem dos microcontroladores MC68HC908QT1/QT2/QT4 – PDIP/SOIC

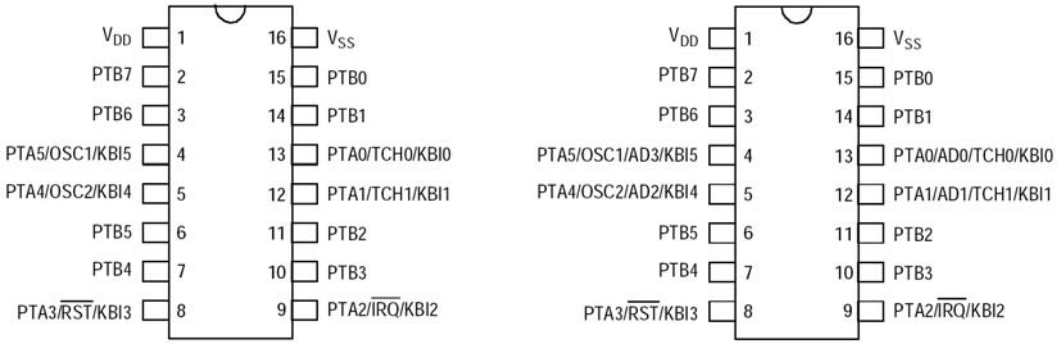

Figura 9 – Pinagem dos microcontroladores MC68HC908QY1/QY2/QY4 – PDIP/SOIC

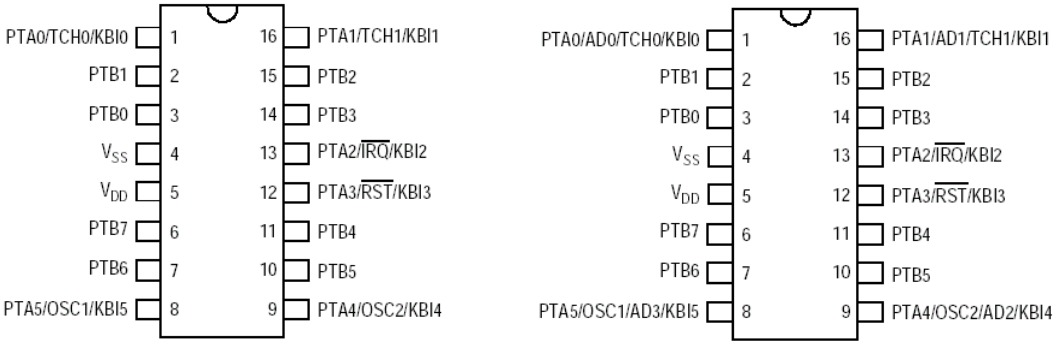

Figura 10 - Pinagem dos microcontroladores MC68HC908QY1/QY2/QY4 – TSSOP

Os pinos multifuncionais dos microcontroladores da família MC68HC908QT/QY tem as seguintes descrições:

Tabela 2 – Descrição dos pinos dos microcontroladores MC68HC908QT/QY

Pino	Descrição	Tipo
V_{DD}	Alimentação – 5V ou 3V	Potência
Vss	Alimentação – GND	Potência
PTA0	PTA0 – Porta de entrada/saída de uso geral	Entrada/Saída
	AD0 – Entrada analógica, canal 0	Entrada
	TCH0 – Entrada/Saída do timer – canal 0	Entrada/Saída
	KBI0 – Entrada da interrupção de teclado 0	Entrada
PTA1	PTA1 – Porta de entrada/saída de uso geral	Entrada/Saída
	AD1 – Entrada analógica, canal 1	Entrada
	TCH1 – Entrada/Saída do timer – canal 1	Entrada/Saída
	KBI1 – Entrada da interrupção de teclado 1	Entrada
PTA2	PTA2 – Porta de entrada de uso geral	Entrada
	/IRQ – Interrupção externa	Entrada
	KBI2 – Entrada da interrupção de teclado 2	Entrada
PTA3	PTA3 – Porta de entrada/saída de uso geral	Entrada/Saída
	/RST – Entrada de reset (ativo em 0)	Entrada
	KBI3 – Entrada da interrupção de teclado 3	Entrada
PTA4	PTA4 – Porta de entrada/saída de uso geral	Entrada/Saída
	OSC2 – Saída do oscilador a cristal Saída do oscilador interno ou RC	Saída Saída
	AD2 – Entrada analógica, canal 2	Entrada
	KBI4 – Entrada da interrupção de teclado 4	Entrada
PTA5	PTA5 – Porta de entrada/saída de uso geral	Entrada/Saída
	OSC1 – Entrada do oscilador a cristal	Entrada
	AD3 – Entrada analógica, canal 3	Entrada
	KBI5 – Entrada da interrupção de teclado 5	Entrada
PTB[0:7][2]	PTB[0:7] – 8 entradas/saídas de uso geral	Entrada/Saída

[2] *Os pinos do port PTB não estão disponíveis nos componentes de 8 pinos.*

Os pinos que tem múltiplas funções num mesmo pino possuem uma seqüência de prioridade que é dada pela tabela a seguir:

Tabela 3 – Seqüência de prioridade dos pinos multifuncionais

Pino	Seqüência de prioridade (maior para menor)
PTA[0]	AD0 → TCH0 → KBI[0] → PTA[0]
PTA[1]	AD1 → TCH1 → KBI[1] → PTA[1]
PTA[2]	/IRQ → KBI[2] → PTA[2]
PTA[3]	/RST → KBI[3] → PTA[3]
PTA[4]	OSC2 → AD2 → KBI[4] → PTA[4]
PTA[5]	OSC1 → AD3 → KBI[5] → PTA[5]

***NOTA**: Na presença da condição de reset todos os pinos se tornam entradas, independente da tabela de prioridade.*

4.2. Memória

Os componentes da família MC68HC908QT/QY possuem os seguintes tipos de memória implementadas:

- ✓ 4096 bytes de memória FLASH (MC68HC908QT4 e MC68HC908QY4)
- ✓ 1536 bytes de memória FLASH (MC68HC908QT1/QT2 e MC68HC908QY1/QY2)
- ✓ 128 bytes de memória RAM para dados
- ✓ 48 bytes para vetores de interrupção programáveis e armazenados na FLASH
- ✓ 416 bytes de memória ROM para o monitor
- ✓ 1536 bytes de rotinas para apagamento e programação da memória FLASH, localizados na ROM

O acesso a endereços de memória reservados, pelo componente, podem ter efeitos imprevisíveis na operação da CPU.

O mapeamento de memória da família MC68HC908QT/QY sempre é implementado dentro do espaço de 64 Kbytes, embora o tamanho da memória real (FLASH + ROM + RAM + Registradores I/O) não ocupe todos os endereços.

A **Figura 10** mostra o mapa de memória da família MC68HC908QT/QY.

$0000	Registradores de I/O 64 bytes
$0040	Reservado 64 bytes
$0080	RAM Interna 128 bytes
$0100	Não implementado 9984 bytes
$2800	ROM Auxiliar 1536 bytes
$2E00	Não implementado 49152 bytes
$EE00	Memória FLASH (MC68HC908QT4/QY4) 4096 bytes
$FE00	Status de BREAK (BSR)
$FE01	Status do Reset (SRSR)
$FE02	Auxiliar de BREAK (BRKAR)
$FE03	Controle de BREAK (BFCR)
$FE04	Status de Interrupção (INT1)
$FE05	Status de Interrupção (INT2)
$FE06	Status de Interrupção (INT3)
$FE07	Reservado para Controle de Teste da FLASH
$FE08	Controle da FLASH (FLCR)
$FE09	Endereço alto de BREAK (BRKH)
$FE0A	Endereço baixo de BREAK (BRKL)
$FE0B	Controle e status de BREAK (BRKSCR)
$FE0C	Status do LVI (LVISR)
$FE0D	Reservado para Teste da FLASH 3 bytes
$FE10	ROM Monitor 416 bytes
$FFB0	FLASH 14 bytes
$FFBE	Proteção de Blocos da FLASH (FLBPR)
$FFBF	Reservado para FLASH
$FFC0	Ajuste do oscilador interno (OSCTRIM)
$FFC1	Reservado para FLASH
$FFC2	FLASH 14 bytes
$FFD0 $FFFF	Vetores de Interrupção 48 bytes

Não implementado 9984 bytes	$0100
ROM Auxiliar 1536 bytes	$2800
Não implementado 51712 bytes	$2E00
Memória FLASH (MC68HC908QT1/QY1/QT2/QY2) 1536 bytes	$F800 $FDFF

Figura 11 – Mapeamento da memória da família MC68HC908QT/QY

A faixa de endereços de $0000 a $003F contém a maioria dos registradores de dados, status e controle. Registradores de I/O adicionais estão localizados no final do endereçamento da memória, a partir do endereço $FE00. Todos estes registradores podem ser vistos com detalhes no *data sheet* da família MC68HC908QT/QY.

Os endereços dos vetores de interrupção são discriminados na tabela a seguir:

Tabela 4 – Vetores de interrupção da família MC68HC908QT/QY

Prioridade do Vetor	Vetor	Endereço	Descrição
Baixa	IF15	$FFDE	Conversão A/D completa (end. Alto)
↓		$FFDF	Conversão A/D completa (end. Baixo)
↓	IF14	$FFE0	Teclado (end. Alto)
↓		$FFE1	Teclado (end. Baixo)
↓	IF13 ↓ IF6	-	Não utilizado
↓	IF5	$FFF2	Estouro do TIM (end. Alto)
↓		$FFF3	Estouro do TIM (end. Baixo)
↓	IF4	$FFF4	Canal 1 TIM (end. Alto)
↓		$FFF5	Canal 1 TIM (end. Baixo)
↓	IF3	$FFF6	Canal 0 TIM (end. Alto)
↓		$FFF7	Canal 0 TIM (end. Baixo)
↓	IF2	-	Não utilizado
↓	IF1	$FFFA	/IRQ (end. Alto)
↓		$FFFB	/IRQ (end. Baixo)
↓	-	$FFFC	SWI (end. Alto)
↓		$FFFD	SWI (end. Baixo)
↓	-	$FFFE	Reset (end. Alto)
Alta		$FFFF	Reset (end. Baixo)

4.2.1. Memória RAM

A memória RAM interna está localizada na faixa de endereços $0080 a $00FF. O registrador *Stack Pointer* (SP) permite que a pilha esteja localizada em qualquer lugar dentro do 64 Kbytes, mas para que funcione corretamente, deve obrigatoriamente apontar para uma região de memória RAM.

Antes do processamento de uma interrupção, a CPU utiliza 5 bytes da pilha para guardar o conteúdo de registradores especiais.

Durante uma chamada de subrotina a CPU utiliza 2 bytes da pilha para guardar o endereço de retorno.

4.2.2. Memória FLASH

A memória FLASH pode ser lida, programada e apagada com a utilização de apenas uma fonte de alimentação externa. As operações de programação e apagamento são habilitadas através da utilização de um "*charge pump*" interno.

A memória FLASH consiste em 4096 ou 1536 bytes, com 48 bytes adicionais para vetores de interrupção. A memória FLASH pode ser apagada em blocos com tamanho mínimo de 64 bytes; e pode ser programada em blocos de 32 bytes, em um ciclo de programação. As operações de programação e apagamento são facilitadas através de bits de controle do registrador de controle da memória FLASH (FLCR). As faixas de endereço para a memória do usuário e dos vetores são:

- ✓ $EE00 - $FDFF: 4096 bytes de memória do usuário para o MC68HC908QT4/QY4
- ✓ $F800 - $FDFF: 1536 bytes de memória do usuário para o MC68HC908QT1/QY1 e para o MC68HC908QT2/QY2
- ✓ $FFD0 - $FFFF: 48 bytes para os vetores de interrupção.

Os procedimentos para apagar uma página, apagar toda a memória, programar e proteger a memória FLASH estão detalhados no *data sheet* da família de microcontroladores MC68HC908QT/QY.

> ***NOTA**: Bits apagados da memória FLASH são lidos como nível lógico 1 e bits programados são lidos como nível lógico 0.*

4.3. Módulo de Integração do Sistema (SIM)

Junto com a CPU, o módulo SIM controla todas as atividades do microcontrolador. O SIM é um controlador de estados que coordena as atividades da CPU e as exceções de tempo. O módulo SIM é responsável por:

- ✓ Geração do clock do barramento e controle da CPU e periféricos
  - ✓ Entrada e recuperação das condições de STOP/WAIT/RESET/BREAK
  - ✓ Controle do clock interno
- ✓ Controle do reset principal, incluindo o reset na energização (POR) e o timeout do computador operando corretamente (COP)
- ✓ Controle de interrupções:
  - ✓ Temporização do reconhecimento da interrupção
  - ✓ Temporização do controle de arbitramento
  - ✓ Geração do endereço do vetor da interrupção
- ✓ Temporização para habilitação/desabilitação da CPU
- ✓ Arquitetura modular expansível para 128 fontes de interrupção

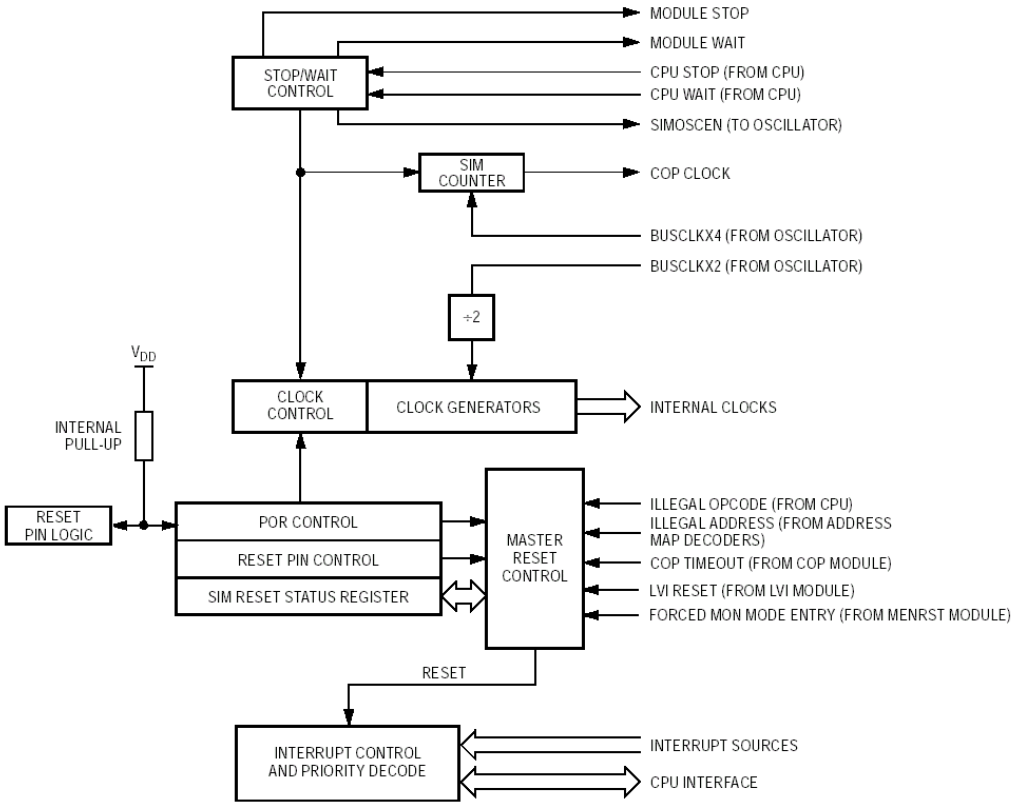

Figura 12 – Diagrama de bloco do módulo SIM

Os pinos /RST e /IRQ estão compartilhados com os sinais PTA3 e PTA2 respectivamente. As funções de /RST e /IRQ podem ser ativadas pela programação adequada do registrador de configuração 2 (CONFIG2).

O gerador de clock do barramento fornece sinais de clock do sistema para a CPU e periféricos do microcontrolador. Os clocks do sistema são gerados a partir do clock BUSCLKX2, com apresentado a seguir.

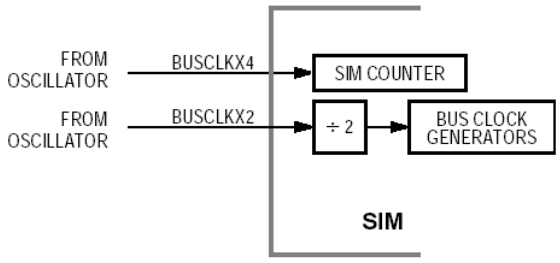

Figura 13 – Sinais de clock do módulo SIM

No modo usuário, a freqüência do barramento interno é igual à freqüência do oscilador dividida por quatro. A freqüência do oscilador é denominada BUSCLKX4.

Quando o módulo de energização (POR) gera um reset, os clocks da CPU e periféricos são desativados e permanecem nesse estado até 4096 BUSCLKX4 ciclos sejam completados. O pino /RST é mantido em nível lógico 0 pelo módulo SIM durante todo esse período.

4.3.1. Inicialização do sistema e reset

O microcontrolador tem várias fontes de reset:

- ✓ Módulo POR (Power-On Reset)
- ✓ Pino de reset externo (/RST)
- ✓ Módulo COP (Computer Operating Properly)
- ✓ Módulo LVI (Low-Voltage Inhibit)
- ✓ *Opcode* ilegal
- ✓ Endereço ilegal

Todas estas fontes de reset carregam o vetor $FFFE-$FFFF no contador de programa (PC) e ativam o sinal de reset interno (IRST). Isto faz com que todos os registradores retornem com seus valores de reset e todos os módulos retornem ao estado de reset.

Todos os resets internos zeram o contador do módulo SIM, mas não os resets externos. Cada reset seta um bit correspondente no registrador de status do reset (SRSR).

4.3.1.1. Reset durante a energização (POR)

Na energização ocorrem os seguintes eventos:

- ✓ Um pulso POR é gerado.
- ✓ O sinal de reset interno é ativado.
- ✓ O módulo SIM habilita o oscilador para fornecer o sinal BUSCLKX4.
- ✓ Clocks internos à CPU e os módulos são mantidos inativos por 4096 BUSCLKX4 ciclos para permitir a estabilização do oscilador.
- ✓ O bit POR do registrador de status do reset (SRSR) é setado e todos os outros bits do registrador são resetados.

4.3.1.2. Reset do módulo COP

Uma das entradas do módulo SIM é reservada para o sinal de reset do COP. O estouro na contagem do contador do COP causa um reset interno e seta o bit COP no registrador de status do reset (SRSR). Maiores detalhes poderão ser vistos no item que descreve o funcionamento do módulo COP.

4.3.1.3. Reset por Instrução Ilegal

O módulo SIM decodifica sinais da CPU para detectar instruções ilegais. Uma instrução ilegal seta o bit ILOP do registrador de status de reset (SRSR) e provoca um reset.

Se o bit de habilitação do modo STOP do registrador de opções de máscara estiver em nível lógico 0, o módulo SIM trata a instrução STOP como um *opcode* ilegal e causa um reset por *opcode* ilegal.

4.3.1.4. Reset por Endereço Ilegal

Uma busca por *opcode* em um endereço não mapeado gera um reset por endereço ilegal. O módulo SIM verifica se a CPU está fazendo a busca a um *opcode* antes de ativar o bit ILAD do registrador de status de reset (SRSR) e, resetar a CPU.

4.3.1.5. Reset do módulo LVI

O sinal de reset do módulo LVI é ativado quando a tensão V_{DD} cai abaixo do valor de desligamento $V_{TRIPF.}$ O bit LVI do registrador de status de reset (SRSR) é setado. 64 BUSCLKX4 ciclos depois, a CPU e as memórias saem do reset para permitir que a seqüência do vetor de reset ocorra.

4.3.2. Controle de exceções

A seqüência normal de execução de programas pode ser alterada em 3 diferentes meios:

- ✓ Interrupções
  - ✓ Interrupções de hardware mascaráveis
  - ✓ Interrupção por software não mascarável (Instrução SWI)
- ✓ Reset
- ✓ Interrupção BREAK

As interrupções são memorizadas e o arbitramento é desenvolvido pelo módulo SIM no início do processo de interrupção. O resultado do arbitramento determina qual vetor de interrupção a CPU vai utilizar. Uma vez que a interrupção é memorizada pelo módulo SIM, nenhuma outra pode tomar precedência, independente da prioridade, antes que a interrupção memorizada seja servida (ou o bit I seja zerado).

No início da interrupção, a CPU salva o conteúdo dos registradores principais na pilha e seta o bit de máscara da interrupção (I) para prevenir interrupções adicionais. No final da interrupção, a instrução RTI recupera o conteúdo dos registradores da CPU na pilha e o processo normal pode continuar.

Uma interrupção de hardware não para a instrução corrente. O processamento da interrupção começa depois de completada a execução da instrução corrente. Quando a instrução é completada, o módulo SIM checa a existência de interrupções de hardware pendentes. Se as interrupções não estiverem mascaradas (bit I = 0 no registrador CCR), e se a interrupção correspondente estiver habilitada, o módulo SIM prossegue com o processamento da interrupção; caso contrário a próxima instrução será buscada e executada.

Se mais de uma interrupção estiverem pendentes ao final da execução de uma instrução, a interrupção de maior prioridade é executada primeiro.

Os bits dos registradores de status de interrupções identificam as fontes de interrupção mascaráveis. A família MC68HC908QT/QY possui as seguintes fontes de interrupção:

Tabela 5 – Fontes de interrupção

Prioridade	Fonte de Interrupção	*Flag*	Masc.	Reg. INT	End. Vetor
Alta	Reset	-	-	-	$FFFE - $FFFF
	Instrução SWI	-	-	-	$FFFC - $FFFD
↑	Pino /IRQ	IRQF1	IMASK1	IF1	$FFFA - $FFFB
	Timer – Canal 0	CH0F	CH0IE	IF3	$FFF6 - $FFF7
	Timer – Canal 1	CH1F	CH1IE	IF4	$FFF4- $FFF5
↓	Estouro do timer	TOF	TOIE	IF5	$FFF2 - $FFF3
	Teclado	KEYF	IMASKK	IF14	$FFDE - $FFDF
Baixa	Conversão ADC completa	COCO	AIEN	IF15	$FFE0 - $FFE1

A instrução SWI é uma interrupção não mascarável que causa uma interrupção independente do estado da máscara da interrupção (bit I) no registrador CCR.

4.4. Módulo Oscilador (OSC)

A família HC08 utiliza um esquema de temporização onde o ciclo de execução das instruções mais simples é formado por 4 fases de um *clock* interno. Se a CPU estiver configurada para receber o sinal de frequência de um cristal oscilador externo, então o ciclo de execução será um quarto da frequência do cristal. Isto é denominado **ciclo de barramento** ou **ciclo da CPU**. Todas as instruções têm sua execução especificada em número de ciclos da CPU. Dessa forma, se o hardware utilizar um cristal externo de 32MHz, a frequência do barramento será de 8 MHz. Portanto, instruções de um ciclo são executadas em 125ns.

Este módulo é utilizado para fornecer uma fonte de clock estável para o sistema. O módulo oscilador gera 2 saídas de clock, BUSCLKX2 e BUSCLKX4. O clock BUSCLKX4 é usado no módulo SIM e no módulo COP. O clock BUSCLKX2 é dividido por 2 no módulo SIM para ser usado como clock do barramento do microcontrolador. Portanto, a freqüência do barramento será um quarto da freqüência de BUSCLKX4.

O oscilador tem 4 opções de fonte de clock disponíveis:

- ✓ Oscilador interno: O microcontrolador gera internamente, uma freqüência fixa de clock ajustável em ±5%. Esta é a opção padrão na saída do reset.
- ✓ Oscilador externo: Um clock externo que pode ser inserido diretamente no OSC1.
- ✓ RC externo: Esta opção utiliza um resistor externo (R) para gerar uma freqüência. O capacitor é interno ao chip.
- ✓ Cristal externo: Módulo oscilador interno ao chip que necessita um cristal externo ou ressonador cerâmico.

4.4.1. Oscilador RC

O circuito oscilador RC foi projetado para uso com um resistor externo (R) para fornecer uma fonte de clock com tolerância menor do que 25%. Nesta configuração típica, necessita-se de componentes externos, um resistor e um capacitor. No microcontrolador MC68HC908QY4, o capacitor é interno ao chip. O valor R deve ter tolerância de 1% ou menor, para obter uma fonte de clock com menos de 25% de tolerância.

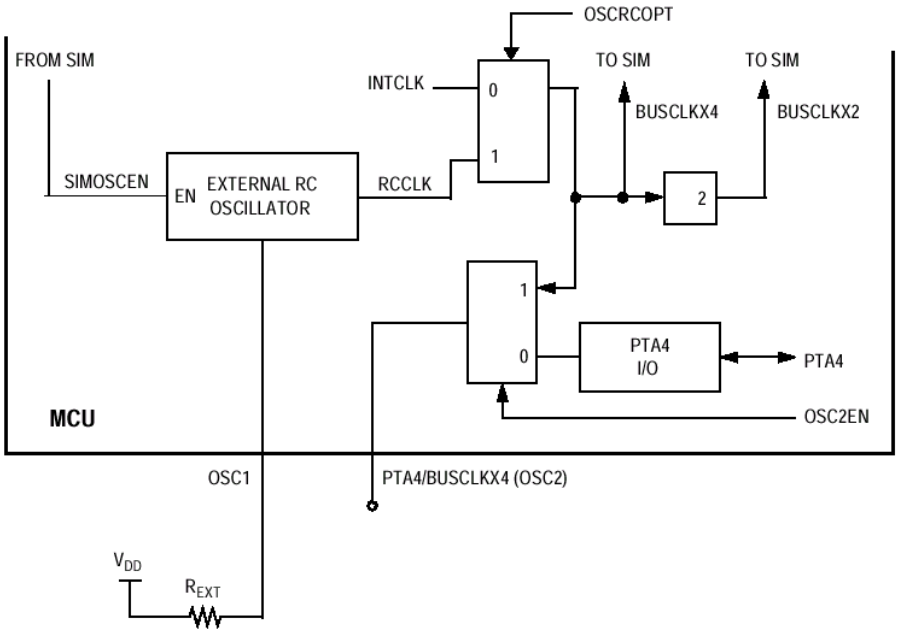

Figura 14 – Conexão com oscilador RC externo

O pino OSC2 pode ser configurado como pino de I/O (PTA4), ou o bit OSC2EN pode ser setado para habilitar o a função OSC2 sem afetar o clock.

4.4.2. Oscilador Interno

O oscilador interno gera uma freqüência típica de 12.8 MHz (INTCLK) resultando em uma velocidade de barramento de 3.2 MHz com uma tolerância de ±25% (sem ajustes).

Nesta opção existe a possibilidade de ajustar a freqüência do clock entre +127 e –128 passos. O registrador utilizado para isso é o OSCTRIM, que tendo seu valor incrementado, aumenta o período do clock. Ajustando o valor em OSCTRIM a freqüência do clock poderá chegar à ±5% em torno de 12.8 MHz.

4.4.3. Oscilador Externo

A opção de clock externo foi projetada para uso quando existe um sinal em freqüência disponível na aplicação para fornecer uma fonte de clock para o microcontrolador. O pino OSC1 é habilitado como uma entrada pelo módulo do oscilador. O sinal de clock é usado diretamente para criar os sinais BUSCLKX4 e também dividido por 2 criar o sinal BUSCLKX2.

Nesta configuração, o pino OSC2 não pode ser configurado como saída do BUSCLKX4. Portanto, o bit OSC2EN no registrador de habilitação do Port A será zerado para habilitar as funções de I/O (PTA4) no pino.

4.4.4. Oscilador a Cristal

O circuito oscilador a cristal (XTAL) foi projetado para uso com um cristal externo ou um ressonador cerâmico para fornecer uma fonte de clock precisa. Nesta configuração, o pino OSC2 é dedicado ao circuito do cristal externo, e conectado na configuração de um oscilador Pierce, como apresentado na figura abaixo. No microcontroladores da família MC68HC908QT/QY, a freqüência de cristal máxima permitida é de 32MHz, que dividido por 4 fornece a freqüência do barramento interno de 8MHz.

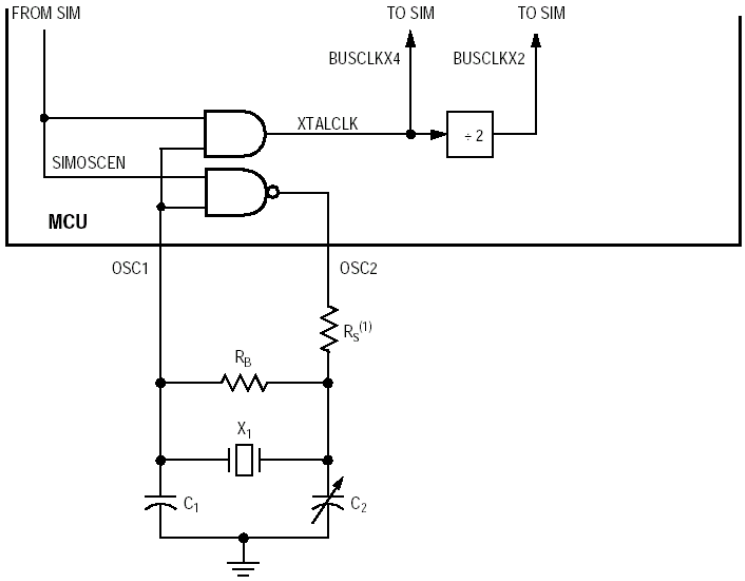

Figura 15 – Conexão com cristal externo

> ***Nota**: O resistor Rs pode ser zero quando utilizado com cristais de alta freqüência. Procure por informações mais detalhadas dos valores dos componentes, com os fabricantes de cristal.*

4.5. Monitor ROM (MON)

O módulo Monitor ROM contém um conjunto de funções implementadas para controlar completamente o microcontrolador através de uma interface serial (uma única linha) conectada a um computador. Dá-se o nome Monitor ROM porque os procedimentos estão armazenados em uma memória do tipo ROM que já vem programado de fábrica, e que não podem ser modificados.

Quando o microcontrolador estiver operando em Modo Monitor, quem tem o controle sobre a execução do programa é o software da memória Monitor ROM. Através de uma interface de comunicação serial e a conexão com um computador, pode-se programar a memória FLASH do microcontrolador. Pode-se também executar um programa já gravado na flash em tempo real, com capacidade e execução passo-a-passo, ou com a inserção de um *breakpoint*, tendo visibilidade de todos registradores internos da CPU e de todo o mapa de memória. Se o programador estiver utilizando a ferramenta de desenvolvimento *CodeWarrior* (ver capítulo Ferramentas de Desenvolvimento – *CodeWarrior*) ele não tem que se preocupar com os comandos do modo monitor que possibilitam essas funcionalidades pois o ambiente de desenvolvimento já o faz, dessa forma a interação com o Modo Monitor, do ponto de vista do software é transparente para o usuário.

Entre as principais características do monitor incluem-se:

- ✓ Funcionalidade normal para o usuário na maioria dos pinos
- ✓ Um pino dedicado para comunicação entre o Monitor ROM e um computador central
- ✓ Comunicação serial padrão com o computador central
- ✓ Execução do código em RAM ou FLASH
- ✓ Características de proteção de código da memória FLASH
- ✓ Interface de programação da memória FLASH
- ✓ Utilização de cristal externo ou oscilador (taxa comunicação = frequência/1024)
- ✓ Modo de operação com oscilador interno (sem freqüência externa ou tensão alta)
- ✓ 416 bytes de código do monitor ROM
- ✓ Modo de entrada no Monitor sem necessidade de tensão alta (V_{TST}) se o vetor de reset estiver apagado ($FFFE e $FFFF contendo $FF)
- ✓ Modo de entrada padronizado se tensão alta for aplicada no pino /IRQ

O Monitor ROM recebe e executa comandos vindos de um computador central através de uma interface de comunicação (padrão físico RS232).

Os comandos do Monitor podem acessar qualquer endereço da memória. No modo Monitor, a CPU pode executar um código descarregado na RAM por um computador, enquanto a maioria dos pinos continua com suas funções normais de operação. Toda a comunicação entre o computador e a CPU é feita através do pino PTA0, que necessita de um conversor de níveis para poder ser conectado ao computador.

A entrada no modo Monitor se dá após a energização (POR) ou um Reset quando determinadas condições (Tabela 6) forem satisfeitas.

Tabela 6 – Entrada em Modo Monitor/Modo Usuário após um Reset

Modo	/IRQ	/RST	Vetor Reset	PTA1	PTA4	Clock (MHz) / Baud (bps)	Comentário
Modo Monitor V_{TST}	V_{TST}	V_{DD}	x	1	0	Externo (f) / f / 1024	Tensões nos pinos PTA1 e PTA4 são necessárias. As funções /RST e OSC1 ativas.
Modo Monitor Forçado	V_{DD}	x	$FF (apagado)	x	x	Externo (f) / f / 1024	Função OSC1 ativa. /RST e /IRQ estão disponíveis apenas se configurados depois.
Modo Monitor Forçado	V_{SS}	x	$FF (apagado)	x	x	Interno (3.2) / 9600 bps	As funções /RST, /IRQ, e OSC1 disponíveis apenas se configurados depois.
Modo Usuário	V_{DD} ou V_{SS}	x	≠ $FF (programado)	x	x	x	Entra no modo usuário. Pino /RST disponível apenas se configurado depois.

No modo monitor, a CPU utiliza vetores para Reset, interrupção de software (SWI), e interrupção Break diferentes das utilizadas para o modo usuário. Os vetores alternativos estão localizados na página $FE, ao invés da página $FF, em outras palavras permitem a execução do código relativo ao firmware do monitor interno, ao invés de executar o código do usuário. A Tabela 7 apresenta as diferenças entre os vetores do modo usuário e modo monitor.

Tabela 7 – Vetores do Modo Usuário x Modo Monitor

Modos	Vetor					
	Reset High	**Reset Low**	**Break High**	**Break Low**	**SWI High**	**SWI Low**
Usuário	$FFFE	$FFFF	$FFFC	$FFFD	$FFFC	$FFFD
Monitor	$FEFE	$FEFF	$FEFC	$FEFD	$FEFC	$FEFD

A taxa de comunicação entre o computador e a CPU é controlada pela *freqüência* externa dividida por 1024. Dessa forma, ao utilizar um cristal externo de 9.8304 MHz o *baud rate* da comunicação serial será de 9600 bps.

Pode-se também utilizar o oscilador interno, que neste caso estará ajustado para a freqüência interna de 3.2 MHz para gerar a taxa de 9600 bps. Neste caso é importante que o endereço $FFC0 do mapa de memória do microcontrolador contenha o valor de *trimming* (ajuste) do oscilador interno. Se este valor estiver errado a comunicação do Modo Monitor com o computador pode ser inviabilizada devido a desvios de frequência.

O firmware do Monitor ROM utiliza os comandos listados a seguir na comunicação e controle das atividades do código que será executado para depuração:

- ✓ READ – Leitura de um byte da memória
- ✓ WRITE – Escrita de um byte na memória
- ✓ IREAD – Leitura dos próximos 2 bytes da memória a partir do último endereço acessado
- ✓ IWRITE – Escrita de um byte na memória no último endereço acessado + 1
- ✓ READSP – Leitura do conteúdo do *Stack Pointer*
- ✓ RUN – Executa as instruções PULH e RTI

A CPU executa as instruções SWI e PSHH quando entra no modo Monitor. O comando RUN informa que a CPU deve executar as instruções PULH e RTI. Antes de enviar o comando RUN, o computador pode modificar o conteúdo dos registradores que estão na pilha para prepará-los a executar o programa do usuário.

O módulo Monitor ROM possui uma característica de segurança muito importante, que previne usuários não autorizados a ler o conteúdo da memória FLASH. Por essa característica, o programador só tem acesso ao conteúdo da FLASH depois de enviar 8 bytes de segurança que devem coincidir com bytes armazenados nos endereços $FFF6-$FFFD. Estes endereços contém dados definidos pelo usuário.

Após a energização (POR), a CPU aguarda o recebimento serial de 8 bytes de segurança no pino PTA0, para entrar no modo Monitor. Se os bytes recebidos coincidirem com os bytes localizados nos endereços $FFF6-$FFFD, todas as funções do Monitor estarão disponíveis para uso. Resets posteriores, não desabilitarão as funções do Monitor, isto é, esses bytes de segurança só serão verificados novamente quando ocorrer outro POR.

Se os bytes de segurança não coincidirem com os bytes dos endereços $FFF6-$FFFD, a CPU entra no modo Monitor mas não permite que a FLASH seja lida. Depois de uma seqüência imprópria (dados não coincidentes), o módulo Monitor pode ainda executar uma rotina de apagamento total da memória FLASH.

4.6. Módulo de Interface do Temporizador (TIM)

O módulo TIM contém um temporizador com 2 canais que fornecem referências de tempo, captura de entrada (*input capture*), comparação de saída (*output compare*) e PWM (*Pulse Width Modulation*).

A principais características do módulo TIM são as seguintes:

- ✓ 2 canais para captura de entrada/ comparação de saída
  - ✓ Captura do sinal por borda de subida, borda de descida ou qualquer borda
  - ✓ Setar, resetar ou mudar o estado de uma saída nas ações de comparação de sinais
- ✓ Geração dos sinais de PWM com ou sem buffer
- ✓ Clock de entrada no módulo TIM programável com 7 seleções de pré-escala.
- ✓ Operação em contagem livre (*free-running*) ou contador de módulo pré-determinado
- ✓ Mudança no estouro de contagem de qualquer pino do canal
- ✓ Bits de reset e parada do contador do módulo TIM

O módulo TIM compartilha 2 pinos de entrada/saída com 2 pinos de I/O do Port A. Os pinos são: PTA0/TCH0 e PTA1/TCH1.

O componente central do módulo TIM é um contador de 16 bits que pode operar como temporizador (*free-running counter*) ou como contador (*modulo up-counter*). O contador TIM fornece referências de tempo para funções de captura de entrada e/ou comparação de saída. O contador TIM utiliza os registradores TMODH:TMODL para configurar o valor máximo de contagem do módulo. O software pode ler o valor do contador TIM a qualquer hora sem afetar a seqüência de contagem. Os dois canais do módulo TIM são programáveis independentemente como canais de captura de entrada ou comparação de saída.

A fonte de clock do módulo TIM é o *clock* do barramento interno dividido por uma das 7 opções de saída disponíveis pelo circuito de pré-escala. Os bits de seleção PS[2:0] do registrador de controle e status do módulo TIM (TSC) selecionam a pré-escala, e conseqüentemente, a fonte de clock do módulo TIM.

Com a função de captura de entrada, o módulo TIM pode determinar precisamente o tempo no qual um evento externo ocorre. Quando uma determinada borda ocorre no pino de um canal de captura de entrada, o módulo TIM armazena o conteúdo do contador nos registradores TCHxH:TCHxL. Esta captura pode, também, gerar uma requisição de interrupção.

Com a função de comparação de saída, o módulo TIM pode gerar um pulso periódico com uma polaridade, duração e freqüência programável. Quando o contador alcança o valor programado de contagem, o módulo TIM pode setar, resetar ou mudar o estado do pino daquele canal. Esta comparação pode, também, gerar uma requisição de interrupção.

Se for utilizada a característica de mudança no estouro da contagem (*toggle-on-overflow*) com um canal configurado com comparação de saída, o módulo TIM pode gerar um sinal de PWM. Maiores detalhes são apresentados no data sheet da família MC68HC908QT/QY.

4.7. Conversor Analógico/Digital (ADC)

As principais características do módulo ADC são:

- ✓ 4 canais com multiplexação das entradas
- ✓ Conversão por aproximação sucessiva linear
- ✓ Resolução de 8 bits
- ✓ Conversão simples ou contínua
- ✓ *Flag* indicativa de conversão completada ou interrupção por conversão completada
- ✓ Clock do ADC selecionável

Os 4 canais do ADC são disponíveis para amostrar sinais externos nos pinos PTA0, PTA1, PTA4 e PTA5. Um multiplex analógico interno permite que a seleção de um dos canais seja amostrada e armazenada no registrador ADCVIN, com resolução de 8 bits. Quando a conversão é completada, o módulo ADC coloca o resultado no registrador de dados do ADC e seta um *flag* ou gera uma interrupção.

4.8. Portas de Entrada/Saída (I/O)

Os microcontroladores MC68HC908QT1/QT2/QT4 tem 5 pinos bidirecionais programáveis como entrada/saída (I/O) e 1 pino de entrada. Os microcontroladores MC68HC908QY1/ QY2/QY4 tem 13 pinos bidirecionais programáveis como entrada/saída e 1 pino de entrada.

> ***NOTA**: Conecte todos os pinos de I/O não utilizados a um determinado nível lógico, V_{DD} ou V_{SS}. Embora os ports de I/O não requeiram terminação para operar adequadamente, uma terminação reduz o consumo excessivo de corrente e a possibilidade de falha por eletrostática.*

4.8.1. Port A

O Port A tem 6 funções especiais que compartilham todos os 6 pinos com o módulo de interrupção de teclado (KBI). Cada pino do port A também tem pull-ups internos configuráveis se o correspondente pino estiver configurado com port de entrada.

4.8.2. Port B

O Port B tem 8 bits de entrada/saída de propósito geral. O Port B está disponível apenas nos seguintes microcontroladores: MC68HC908QY1/QY2/QY4. Cada pino do port B também tem pull-ups internos configuráveis se o correspondente pino estiver configurado como entrada.

4.9. Interrupção Externa (IRQ)

O pino /IRQ, compartilhado com PTA2 e interrupção do teclado (KBI), fornece uma entrada a uma interrupção mascarável. Entre as características do módulo IRQ encontram-se:

- ✓ Pino de interrupção externa - /IRQ
- ✓ Bits de controle da interrupção /IRQ
- ✓ Buffer de histerese
- ✓ Programação da interrupção por borda exclusivamente, ou borda e nível
- ✓ Reconhecimento de interrupção automático
- ✓ Resistor de pull-up interno selecionável

A habilitação da interrupção é feita pela configuração correta do bit IRQEN, presente no registrador CONFIG2. Se um nível lógico 0 for aplicado ao pino /IRQ, uma requisição de interrupção fica armazenada (*latch*) até que uma das seguintes ações ocorra:

- ✓ *Vetor Fetch* – Uma busca ao vetor da interrupção é realizada, iniciado o serviço de atendimento a interrupção.
- ✓ *Software Clear* – O software pode limpar o *latch* da interrupção pela escrita no bit de reconhecimento presente no registrador de controle e status da interrupção (ISCR).
- ✓ *Reset* – Um reset automaticamente limpa o *latch* da interrupção.

4.10. Módulo de Interrupção do Teclado (KBI)

O módulo de interrupção do teclado (KBI) fornece 6 interrupções externas mascaráveis independentemente, que são acessíveis através dos pinos PTA0-PTA5, mais uma interrupção mascarável controlada pela lógica de auto-despertar. As principais características do módulo de interrupção do teclado incluem:

- ✓ 6 pinos de interrupção de teclado, com bits de habilitação de interrupção do teclado separados e uma máscara de interrupção do teclado.
- ✓ 1 interrupção interna controlada pela lógica de auto-despertar, com bit de habilitação da interrupção separado, compartilhado com a mesma máscara de interrupção do teclado.
- ✓ Pull-ups configuráveis por software, quando os pinos forem configurados como entrada.
- ✓ Interrupção programável por exclusivamente por borda ou borda e nível.
- ✓ Saída dos modos de baixo consumo.

Uma interrupção de teclado é armazenada quando um ou mais entradas da interrupção do teclado vai para nível lógico 0 depois de todas estarem em nível lógico 1. O bit MODEK no registrador de controle e status do teclado controla o modo de disparo da interrupção de teclado. Maiores detalhes são apresentados no *data sheet* da família MC68HC908QT/QY.

4.11. Módulo Computador Operando Corretamente (COP)

O módulo COP (*Computer Operating Properly*) contém um contador que gerar um reset quando estourar sua contagem. O módulo COP ajuda o software a se recuperar de quando o sistema passa a executar, por exemplo, um código espúrio. Para prevenir o reset do COP seu contador deve ser zerado periodicamente. O módulo COP pode ser desabilitado através do bit COPD presente no registrador de configuração 1 (CONFIG1).

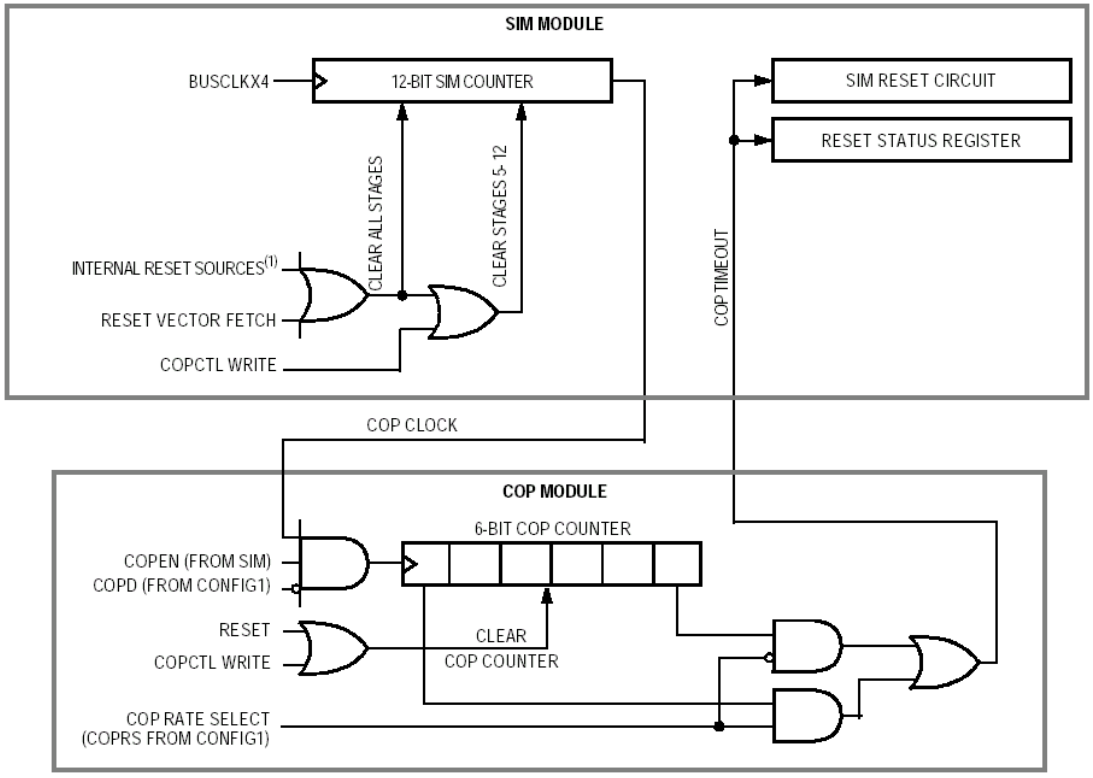

Figura 16 – Diagrama de blocos do módulo COP

O módulo COP contém um contador de 6 bits que roda livre precedido pelo contador de 12 bits do módulo SIM. Se não for zerado por software, o contador do COP estoura e gera um reset assíncrono depois de (2^{18}-2^4) ou (2^{13}-2^4) BUSCLKX4 ciclos; dependendo do estado do bit de seleção da taxa do COP (COPRS), no registrador de configuração1 (CONFIG1).

Um ciclo de escrita de qualquer valor no endereço $FFFF antes da ocorrência de estouro, zera o contador COP e os estágios 12-5 do contador SIM. O registrador localizado no endereço $FFFF, também é denominado registrador de controle do COP (COPCTL).

Um reset do COP coloca o pino /RST em nível lógico 0 por 32 x BUSCLKX4 ciclos e seta o bit COP no registrador de status do reset (RSR).

O módulo COP é desabilitado no modo monitor quando V_{TST} estiver presente no pino /IRQ.

As instruções WAIT e STOP colocam a CPU em modo de baixo consumo. No modo WAIT o COP continua a operar, e para prevenir-se um reset, periodicamente o contador COP deve ser zerado. No modo STOP o sinal de clock BUSCLKX4 é desligado, portanto não é capaz de gerar um reset, porém, na saída do modo STOP assegure-se de ter o período de timeout completo do COP, para prevenir-se de um possível reset durante a execução do software.

4.12. Módulo de Inibição por Tensão Baixa (LVI)

O módulo LVI (*Low-Voltage Inhibit*) tem a função de monitorar a tensão do pino de alimentação (V_{DD}) e pode forçar um reset quando a tensão V_{DD} cair abaixo da tensão de desligamento - V_{TRIPF}.

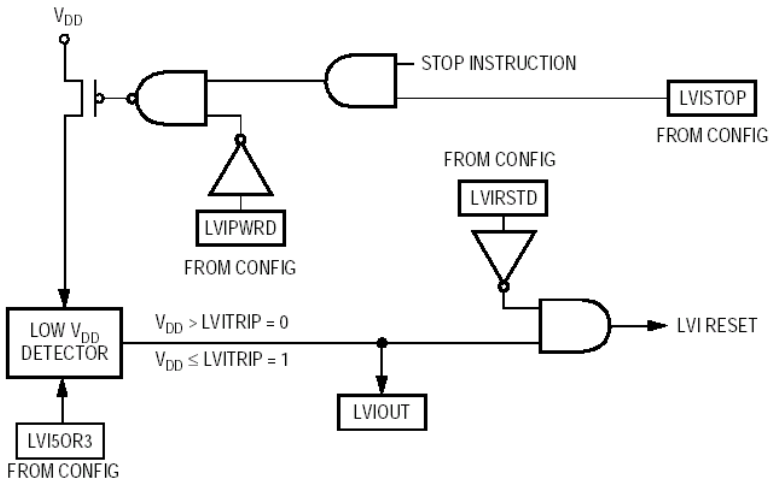

Figura 17 – Diagrama de blocos do módulo LVI

O módulo LVI contém um circuito de referência com zona morta e um comparador. Os bits de configuração do módulo LVI encontram-se no registrador CONFIG1.

Se LVIPWRD = 0, a tensão V_{DD} passa a ser monitorada pelo módulo LVI.

Se LVIRSTD = 0, o módulo LVI gera um reset quando V_{DD} cair abaixo da tensão V_{TRIPF}. Se LVISTOP = 1, o módulo LVI passa a operar também no modo STOP.

Se LVI5OR3 = 1, a tensão V_{TRIPF} é configurada para operação de 5V. Se LVI5OR3 = 0, a tensão V_{TRIPF} é configurada para operação de 3V.

Uma vez ocorrido um reset por LVI, o microcontrolador permanece em reset até que V_{DD} suba acima da tensão V_{TRIPR}. A tensão V_{TRIPF} é menor do que V_{TRIPR}, dessa forma previne-se que a CPU entre e saia continuamente do reset.

Em aplicações que possam operar com V_{DD} em níveis abaixo de V_{TRIPF} , o usuário pode monitor por software o bit LVIOUT. No registrador de configuração, o bit LVIPWRD deve estar em nível lógico 0 para habilitar o módulo LVI, e o bit LVISTD deve estar em nível lógico 1 para desabilitar o reset por LVI.

O bit LVIOUT está presente no registrador de status do LVI (LVISR).

4.13. Módulo Break (BREAK)

O módulo Break pode gerar uma interrupção BREAK que para o fluxo normal do programa em um endereço definido para entrar em um programa alternativo (em "background"). Entre as principais características do módulo Break estão as seguintes:

- ✓ Registradores de I/O acessíveis durante a interrupção Break
- ✓ CPU gera interrupções Break
- ✓ Software gera interrupções Break
- ✓ COP é desabilitado durante interrupções Break

Os seguintes eventos podem causar a ocorrência de uma interrupção Break:

- ✓ A CPU gera um endereço (o endereço no *Program Counter*) que coincide com o conteúdo dos registradores de endereço de Break;
- ✓ O software escreve um nível lógico 1 no bit BRKA no registrador de controle e status de Break.

Quando um endereço do barramento interno coincide com o valor escrito nos registradores de endereço de Break, o módulo Break implementa um sinal de *breakpoint* (/BKPT) no módulo SIM. Este sinal é responsável pela execução, no registrador de instrução da CPU, da instrução SWI (*software interrupt*), depois de completada a instrução corrente.

Ao encontrar a instrução RTI (*return from interrupt*) a rotina de Break é finalizada e o microcontrolador retorna para sua operação normal.

5. CONJUNTO DE INSTRUÇÕES

5.1. Introdução

A CPU08, por ser de arquitetura CISC (*Complex Instruction Set Computer*), apresenta um conjunto de instruções poderoso que pode ser utilizado de forma simples e imediata. Por possuir instruções abrangentes o programador consegue desenvolver programas de forma mais eficiente e sem muito esforço.

Como exemplo de instruções poderosas da CPU da família HC08 encontram-se a multiplicação e a divisão. Uma multiplicação de dois operandos de 8 bits é realizada em 625ns, se o *clock* do barramento interno for de 8MHz. Uma divisão de um operando de 16 bits por outro de 8 bits é executada em 825ns.

Aliado a uma potente CPU, a família HC08 possui 16 modos de endereçamento, alguns deles implementados para geração de código eficientes com linguagens de alto nível, como por exemplo, a linguagem C.

5.2. Nomenclatura

Registradores da CPU

A	Acumulador
CCR	Registrador *Condition Code*
H	Registrador de índice, 8 bits mais significativos
X	Registrador de índice, 8 bits menos significativos
PC	*Program Counter*
PCH	*Program Counter*, 8 bits mais significativos
PCL	*Program Counter*, 8 bits menos significativos
SP	*Stack Pointer*

Bits do Registrador CCR

V	*Overflow* (estouro em operações com sinal) – bit 7
H	*Half carry* (estouro em operações em BCD) – bit 4
I	Mascara interrupções – bit3
N	Negativo (indicador de número negativo) – bit 2
Z	Zero (indicador de resultado zero) – bit 1
C	*Carry/Borrow* (estouro em operações sem sinal) – bit 0

5.3. Conjunto de Instruções Resumido

Movimentação de Dados

Instrução	*Descrição*
LDA	Carrega o acumulador
LDHX	Carrega o par de registradores (H:X)
LDX	Carrega o registrador X
STA	Armazena o acumulador na memória
STHX	Armazena os registradores (H:X) na memória
STX	Armazena o registrador X na memória
TAX	Transfere o acumulador para o registrador X
TXA	Transfere o registrador X para o acumulador
MOV	Movimenta dados da memória
PSHA	Insere o conteúdo do acumulador na pilha
PSHH	Insere o conteúdo do registrador H na pilha
PSHX	Insere o conteúdo do registrador X na pilha
PULA	Insere o conteúdo da pilha no acumulador
PULH	Insere o conteúdo da pilha no registrador H
PULX	Insere o conteúdo da pilha no registrador X

Aritméticas

Instrução	*Descrição*
ADD	Adiciona sem *carry*
ADC	Adiciona com *carry*
SUB	Subtrai sem *carry*
SBC	Subtrai com *carry*
MUL	Multiplica sem sinal
DIV	Divide
DAA	Ajuste decimal do acumulador

Manipulação de dados

Instrução	*Descrição*
AIS	Adiciona valor imediato ao *Stack Pointer*
AIX	Adiciona valor imediato ao Registrador de índice
ASLA **ASLX** **ASL** **LSLA** **LSLX** **LSL**	*Shift* aritmético à esquerda do acumulador *Shift* aritmético à esquerda do registrador X *Shift* aritmético à esquerda da memória *Shift* lógico à esquerda do acumulador *Shift* lógico à esquerda do registrador X *Shift* lógico à esquerda da memória
ASRA **ASRX** **ASR**	*Shift* aritmético para a direita do acumulador *Shift* aritmético para a direita do registrador X *Shift* aritmético para a direita da memória
CLRA **CLRH** **CLRX** **CLR**	Limpa acumulador Limpa registrador H Limpa registrador X Limpa memória
DECA **DECX** **DEC**	Decrementa acumulador Decrementa registrador X Decrementa memória
INCA **INCX** **INC**	Incrementa acumulador Incrementa registrador X Incrementa memória
LSRA **LSRX** **LSR**	*Shift* lógico à direita do acumulador *Shift* lógico à direita do registrador X *Shift* lógico à direita da memória
NEGA **NEGX** **NEG**	Negativo do acumulador (complemento de 2) Negativo do registrador X (complemento de 2) Negativo da memória (complemento de 2)
ROLA **ROLX** **ROL**	Rotação para a esquerda com *carry* do acumulador Rotação para a esquerda com *carry* do registrador X Rotação para a esquerda com *carry* da memória
RORA **RORX** **ROR**	Rotação para a direita com *carry* do acumulador Rotação para a direita com *carry* do registrador X Rotação para a direita com *carry* da memória

Bit Manipulação de Bits

Instrução	*Descrição*
BCLR	Limpa o bit ***n*** na memória
BSET	Seta o bit ***n*** na memória

Lógica

Instrução	*Descrição*
AND	AND lógico
COMA **COMX** **COM**	Complementa acumulador (complemento de 1) Complementa registrador X (complemento de 1) Complementa memória (complemento de 1)
EOR	Exclusive-OR do acumulador com memória
ORA	OR lógico
NSA	Troca nibbles do acumulador

Teste de dados

Instrução	*Descrição*
BIT	Testa bits (comparação bit a bit)
BRCLR	Desvia se bit ***n*** na memória está limpo
BRSET	Desvia se bit ***n*** na memória está setado
CMP	Compara acumulador com memória
CPHX	Compara registrador de índice com memória
CPX	Compara X com memória
TSTA **TSTX** **TST**	Testa se acumulador negativo ou zero Testa se registrador X negativo ou zero Testa se memória negativa ou zero

Desvios condicionais

Instrução	*Descrição*
BCC	Desvia se bit C = 0
BHS	Desvia se conteúdo do acumulador for maior ou igual ao conteúdo da memória (operandos sem sinal)
BCS	Desvia se bit C = 1
BLO	Desvia se conteúdo do acumulador for menor do que o conteúdo da memória(operandos sem sinal)
BEQ	Desvia se igual
BGE	Desvia se conteúdo do acumulador for maior ou igual ao conteúdo da memória (operandos com sinal)
BGT	Desvia se conteúdo do acumulador for maior do que o conteúdo da memória (operandos com sinal)
BHCC	Desvia se bit H = 0
BHCS	Desvia se bit H = 1
BHI	Desvia se conteúdo do acumulador for maior do que o conteúdo da memória (operandos sem sinal)
BIH	Desvia se pino /IRQ = 1
BIL	Desvia se pino /IRQ = 0
BLE	Desvia se conteúdo do acumulador for menor ou igual ao conteúdo da memória (operandos com sinal)
BLS	Desvia se conteúdo do acumulador for menor ou igual ao conteúdo da memória (operandos sem sinal)
BLT	Desvia se conteúdo do acumulador for menor do que o conteúdo da memória (operandos com sinal)
BMC	Desvia se bit I = 0
BMI	Desvia se negativo
BMS	Desvia se bit I = 1
BNE	Desvia se diferente
BPL	Desvia se positivo
BRA	Desvia sempre
BRN	Nunca desvia
BSR	Desvia para subrotina
CBEQA	Compara acumulador com valor imediato e desvia se igual
CBEQX	Compara registrador X com valor imediato e desvia se igual
CBEQ	Compara acumulador com memória e desvia se igual
DBNZA	Decrementa acumulador e desvia se diferente de zero
DBNZX	Decrementa registrador X e desvia se diferente de zero
DBNZ	Decrementa memória e desvia se diferente de zero

Desvios incondicionais

Instrução	*Descrição*
JMP	Salto
JSR	Salta para subrotina
RTS	Retorno de subrotina

Controle

Instrução	*Descrição*
CLC	Limpa bit C
CLI	Limpa bit I (habilita interrupções)
NOP	Sem operação
SEC	Seta bit C
SEI	Seta bit I (desabilita interrupções)
RSP	Reseta o *Stack Pointer*
RTI	Retorno da interrupção
STOP	Habilita o pino /IRQ e para o oscilador
SWI	Interrupção de software
TAP	Transfere acumulador para CCR
TPA	Transfere CCR para acumulador
TSX	Transfere SP para Registrador de índice
TXS	Transfere Registrador de índice para SP
WAIT	Habilita interrupções e para o processador

5.3.1. Conjunto de instruções completo

O conjunto de instruções completo encontra-se no manual do microcontrolador (ver documento MC68HC908QY4\D).

5.4. Características elétricas

A características/especificações elétricas dos componentes da família M68HC908QT/QY encontram-se no manual do microcontrolador (ver documento MC68HC908QY4\D).

6. PROGRAMAÇÃO ASSEMBLER

Para um microprocessador/microcontrolador executar um programa, o conteúdo do arquivo do código fonte deve ser convertido em uma seqüência de instruções binárias (*opcodes*), e então carregadas na memória de programa do microcontrolador. Esta conversão do código fonte em *opcodes* é realizada por um aplicativo denominado montador assembler, ou simplesmente assembler, e os códigos de máquina (*opcodes*) resultantes são armazenados em um arquivo no formato .S19 da Motorola, que pode ser carregado/gravado na memória de programa do microcontrolador.

6.1. Sintaxe

O arquivo fonte de um programa deve ser escrito com uma sintaxe correta, de tal forma que o assembler consiga gerar os *opcodes* corretamente. Cada linha do arquivo fonte contém os seguintes elementos básicos:

- ✓ Rótulos (Labels)
- ✓ Instruções Mnemônicas ou pseudo-instruções
- ✓ Operandos
- ✓ Comentários

A ordem e posição desses elementos é importante. Em particular, os rótulos sempre começam na coluna um, mnemônicos devem começar a partir da coluna dois (normalmente utiliza-se uma tabulação a partir da coluna um), operando sempre seguem os mnemônicos, e o caracter ponto-e-vírgula (;) sempre precede comentários.

Rótulos são nomes dados aos endereços iniciais de: instruções de desvio, subrotinas ou subrotinas de serviço de interrupções. Sempre que possível deve-se utilizar nomes que tenham algum significado para o sistema.

Mnemônicos são abreviações textuais significativas dos *opcodes*, e podem conter ou não **operandos** (depende da instrução).

Finalmente, para facilitar o entendimento, as linhas contendo instruções mnemônicas podem ser freqüentemente comentadas. Isto é permitido no arquivo fonte sempre que houver um ponto-e-vírgula precedendo um texto explicativo (**comentário**). O assembler irá ignorar o conteúdo da linha depois que encontrar um ponto-e-vírgula, e o conteúdo não fará parte do código executável (arquivo no formato .S19).

A seguir estão alguns exemplos de linhas válidas de arquivos de código fonte:

```
Rótulo    Mnemônico   Operando     Comentários
; Isto eh uma linha de comentario
          MOV         #$FF,PORTB   ; PORTB = saida
          CLRA
          JSR         Dly_1s       ; Subrotina de 1 segundo
          BCLR        1,PORTA
VOLTA4:   LDA         PORTA
```

Notas:

1. *Sempre coloque um caracter de tabulação antes das instruções ou pseudo-instruções;*
2. *Para instruções que envolvem 2 ou mais operandos, não deixe espaços em branco após a vírgula que separa os operandos;*
3. *Não utilize nomes de variáveis ou rótulos iniciando com números (p.ex.: 3_desloc);*
4. *Não utilize caracteres acentuados da língua portuguesa.*

6.2. Diretivas

Além do conjunto de instruções da família HC08 apresentado anteriormente, o assembler contém diversas pseudo-instruções, também denominadas **diretivas.** As diretivas do assembler são declarações que fornecem instruções para o assembler, e não para a CPU. Neste documento serão discutidas somente algumas diretivas do assembler incluído no *CodeWarrior*. Maiores detalhes podem ser vistos na documentação do *CodeWarrior*. As diretivas mais importantes são listadas na Tabela 8:

Tabela 8 – Principais Diretivas do Assembler

BASE	Descrição:	Seleciona a base de representação numérica de números sem prefixo.
	Sintaxe:	**BASE** <n> onde: <n> = 2, 8, 10 ou 16
	Exemplo:	**BASE** 10
DC	Descrição:	Define valores constantes na memória.
	Sintaxe:	[<label>:] **DC** [.<tamanho>] <expressão> [, <expressão>]... onde: <tamanho> = **B** (default), **W** ou **L**.
	Exemplo:	Mens_CNZ: **DC.B** "CNZ Engenharia" Tempo_Sem: **DC.W** $0124, $437F, $003E, $0000
	Sinônimo:	**FCB** (= **DC.B**), **FDB** (= 2 **DC.B** = **DC.W**), **DCL** (= 4**DC.B** = **DC.L**)
DCB	Descrição:	Aloca um bloco de memória inicializado com o valor especificado. O tamanho do bloco é dado por <tamanho> * <quantidade>.
	Sintaxe:	[<label>:] **DCB** [.<tamanho>] <quantidade>, <valor> onde: <tamanho> = **B** (default), **W** ou **L**.
	Exemplo:	**DCB.W** 3, $FFFE ; Aloca 3 words com conteúdo $FFFE

DS	Descrição:	Reserva memória para variáveis. O conteúdo da memória reservada não é inicializada. O tamanho do bloco é dado por <*tam.*> * <*qtde*>.
	Sintaxe:	[<label>:] **DS** [.<tamanho>] <quantidade> onde: <tamanho> = **B** (default), **W** ou **L**.
	Exemplo:	Contador: **DS.B** 2 ; 2 bytes contínuos na memória
	Sinônimo:	**RMB** (= **DS.B**), **RMD** (= 2**DS.B** = **DS.W**), **RMQ** (= 4**DS.B** = **DS.L**)
END	Descrição:	Fim do código fonte.
	Sintaxe:	**END**
	Exemplo:	**END**
EQU	Descrição:	Aloca ao símbolo (label) um valor permanente.
	Sintaxe:	<label>: **EQU** <expressão>
	Exemplo:	RAM: **EQU** $0040 Estado0: **EQU** %00100110
INCLUDE	Descrição:	Insere arquivo especificado no arquivo fonte. O nome do arquivo deve estar entre aspas.
	Sintaxe:	**INCLUDE** "<nome do arquivo>"
	Exemplo:	**INCLUDE** "M68HC908QT/QY.INC" ; Definições dos uCs QT/QY.
ORG	Descrição:	Marca a origem do seção a partir do endereço especificado.
	Sintaxe:	**ORG** <expressão>
	Exemplo:	**ORG** $F800 ; Endereço inicial da FLASH (M68HC908QT1)
SECTION	Descrição:	Define uma seção relocável e inicia um contador para o código imediatamente após a sua declaração
	Sintaxe:	<nome>: **SECTION** [SHORT][<número>]
	Exemplo:	Variáveis: **SECTION** **DS.W** 6 ; Reversa 12 bytes (6 words) na RAM.
XDEF	Descrição:	Define os rótulos definidos no módulo que poderão ser referenciados em outros módulos
	Sintaxe:	**XDEF** [.<tamanho>] <label>[,<label>]... Onde: <tamanho> = W(default)
	Exemplo:	**XDEF** _Startup, Inicio
	Sinônimo:	**GLOBAL**, **PUBLIC**

6.3. Linker

O projeto de software pode ser desenvolvido em uma estrutura de módulos (conjunto de rotinas que executam funções pré-determinadas), em arquivos fonte independentes. Cada arquivo fonte pode ser construído e testado separadamente, e posteriormente ligados ("*linkados*") de tal forma a implementar todas a funcionalidade requerida por uma aplicação. Nesse estágio de desenvolvimento, não será necessário a utilização do ***Linker*** disponível no CodeWarrior.

6.4. Arquivo formato S19

Para microcontroladores da Motorola, a forma mais comum de arquivo código binário é conhecido como formato **S19**. Um arquivo no formato **S19** é um arquivo texto ASCII que pode ser visualizado por um editor de texto ou processador de texto. Cada linha é um registro. Cada registro começa com a letra maiúscula S seguida de um número de 0 a 9. Para aplicações com microcontroladores únicos códigos importantes são S0, S1 e S9. O registro S0 é um cabeçalho opcional que contém o nome do arquivo para benefício dos programadores que necessitam manter essas informações. Os registros S1 contém os todos os dados (*opcodes*) da aplicação. O registro S9 é usado para marcar o fim do arquivo no formato **S19**. Todos os números em um registro (linha) estão em hexadecimal.

Um registro é composto dos seguintes campos:

- ✓ ***Tipo:*** pode ser S0 (cabeçalho), S1 (dados) ou S9 (fim de arquivo);
- ✓ ***Comprimento***: par de digitos hexadecimal (2 caracteres ASCII) que indica o número de bytes restantes no registro (excluídos o tipo e o comprimento do campo);
- ✓ ***Endereço da memória***: composto de 16 bits (4 caracteres ASCII) a partir de onde serão armazenados os dados do registro (do tipo S1);
- ✓ ***Dados do código binário***: pares de dígitos hexadecimal localizados após o endereço e antes do *checksum.* Contém os códigos de máquina representados em valores de 8 bits de dados para serem armazenados em endereços sucessivos na memória;
- ✓ ***Checksum***: campo de 8 bits que representa a somatório em complemento de um de todos os bytes no registro, exceto os campos de tipo e de *checksum*. Este *checksum* é usado durante a transferência do arquivo no formato **S19** para verificar se os dados estão completos e corretos para cada registro.

A seguir é listado um exemplo de arquivo no formato **S19**:

```
S0200000433A5C50726F6A65746F735F434E51534D5F504544656275672E616273BE
S123F800450100948C9A5FA600B71EA619B71FA602B7041300B600A40427FA1200A614CDE7
S11EF820F82F4A26FA1300B600A40426FA20E687A69B5F5A26FD4A26FA86818C
S105FFFEF80005
S9030000FC
```

O arquivo gerado no formato **S19** é utilizado posteriormente para gravar (programar) a memória FLASH do microcontrolador, e também é utilizado como o arquivo de entrada para ferramentas de depuração, seja ela uma simulação por software ou depuração *In-Circuit*.

7. DESENVOLVIMENTO DE SISTEMAS

Entende-se por sistema, mais especificamente sistemas eletrônicos microprocessados (*embedded systems*), ao conjunto formado pelo hardware com microprocessador ou microcontrolador mais periféricos, integrado ao software básico[3] que implementa as funções determinadas para a aplicação.

O desenvolvimento de hardware é executado por um engenheiro ou técnico eletrônico que normalmente pesquisa, estuda e projeta os circuitos eletrônicos, de preferência com componentes disponíveis no mercado. Esta atividade depende basicamente de estudo, experiência acumulada e de componentes disponíveis e, uma vez projetado e testado dificilmente sofrerá alterações.

O desenvolvimento de software, ao contrário do hardware, depende muito da criatividade do programador e, pode ser alterado com maior freqüência para, por exemplo, estar inserindo novas funções na aplicação. No desenvolvimento de softwares com microcontroladores é interessante iniciar o aprendizado com uma linguagem bem próxima aos códigos de máquina (*Linguagem Assembly*). Para tanto é necessário ter uma compreensão razoável do conjunto de instruções e dos recursos disponíveis pelo assembler.

Para o desenvolver sistemas eletrônicos com microcontroladores, o projetista deve seguir os seguintes passos:

- ✓ Especificar e documentar o sistema (problema ou idéia);
- ✓ Projetar e documentar o hardware;
- ✓ Descrever implementação do software do sistema através de fluxogramas ou outra forma de representação gráfica ou textual;
- ✓ Editar um arquivo com instruções mnemônicas (arquivo fonte) cada bloco apresentado no fluxograma, utilizando um editor de textos.
- ✓ Utilizar um assembler (software) para transcrever as instruções mnemônicas em códigos de máquina necessários a execução do programa pelo microcontrolador.
- ✓ Simular, emular, testar e/ou depurar as funcionalidades do programa desenvolvido.

Para exemplificar o desenvolvimento de um sistema eletrônico completo é apresentado a seguir a implementação um problema simples com as etapas descritas anteriormente.

---

[3] *Software básico – programa que é gravado/programado em memória do tipo ROM/EPROM/FLASH, também denominado FIRMWARE.*

7.1. Especificação do Sistema

Desenvolver um sistema eletrônico capaz de acender um LED por 1 segundo toda vez que um botão for pressionado. O LED não deverá piscar novamente, enquanto o botão for mantido pressionado e até que o botão seja pressionado novamente.

Embora este sistema seja simples, ele demonstra os elementos mais comuns de qualquer aplicação como microcontroladores:

- ✓ Como configurar os sinais de I/O (entradas e saídas);
- ✓ Como um programa pode sentir (ler) sinais digitais de entrada (Botão);
- ✓ Como um programa pode acionar (escrever) sinais digitais de saída (LED);
- ✓ Implementa uma rotina rudimentar de temporização de eventos (LED piscar).

O objetivo piscar um LED pode parecer um tanto quanto "tolo", porém se ao invés do LED for inserido um circuito a relé no pino de saída do microcontrolador, a aplicação sem mudança nenhuma será a de um relé temporizado que é acionado por um sinal de entrada, que pode ser um botão ou um sensor digital qualquer. Este tipo de relé temporizado é encontrado comercialmente, e bastante utilizado para aplicações em automação industrial.

7.2. Projeto de hardware

Apesar de ser muito simples, a solução desse problema com circuitos convencionais requer vários componentes eletrônicos. Se for utilizado um microcontrolador (p.ex.: o MC68HC908QT1) serão necessários: o microcontrolador, um botão (SW), um resistor e um LED (Figura 18). O resto é desenvolvimento de software.

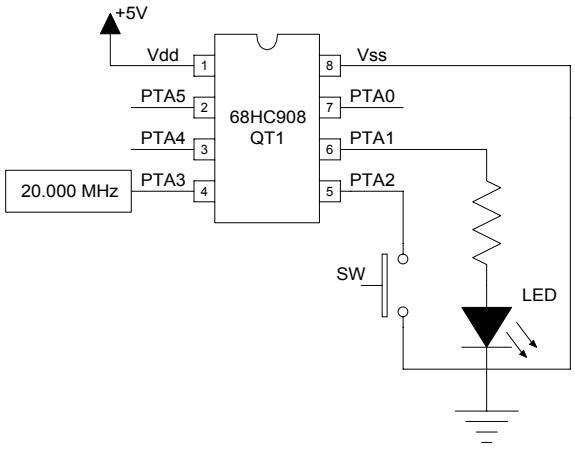

Figura 18 – Esquema elétrico LED+Botão

7.3. Descrição do software

Uma das formas mais utilizadas para descrever um sistema, do ponto de vista de uma posterior implementação de software, é o fluxograma. O fluxograma dessa aplicação é apresentado na Figura 19, que também contém uma legenda (quadro à direita da figura) explicativa de cada tipo de bloco utilizado.

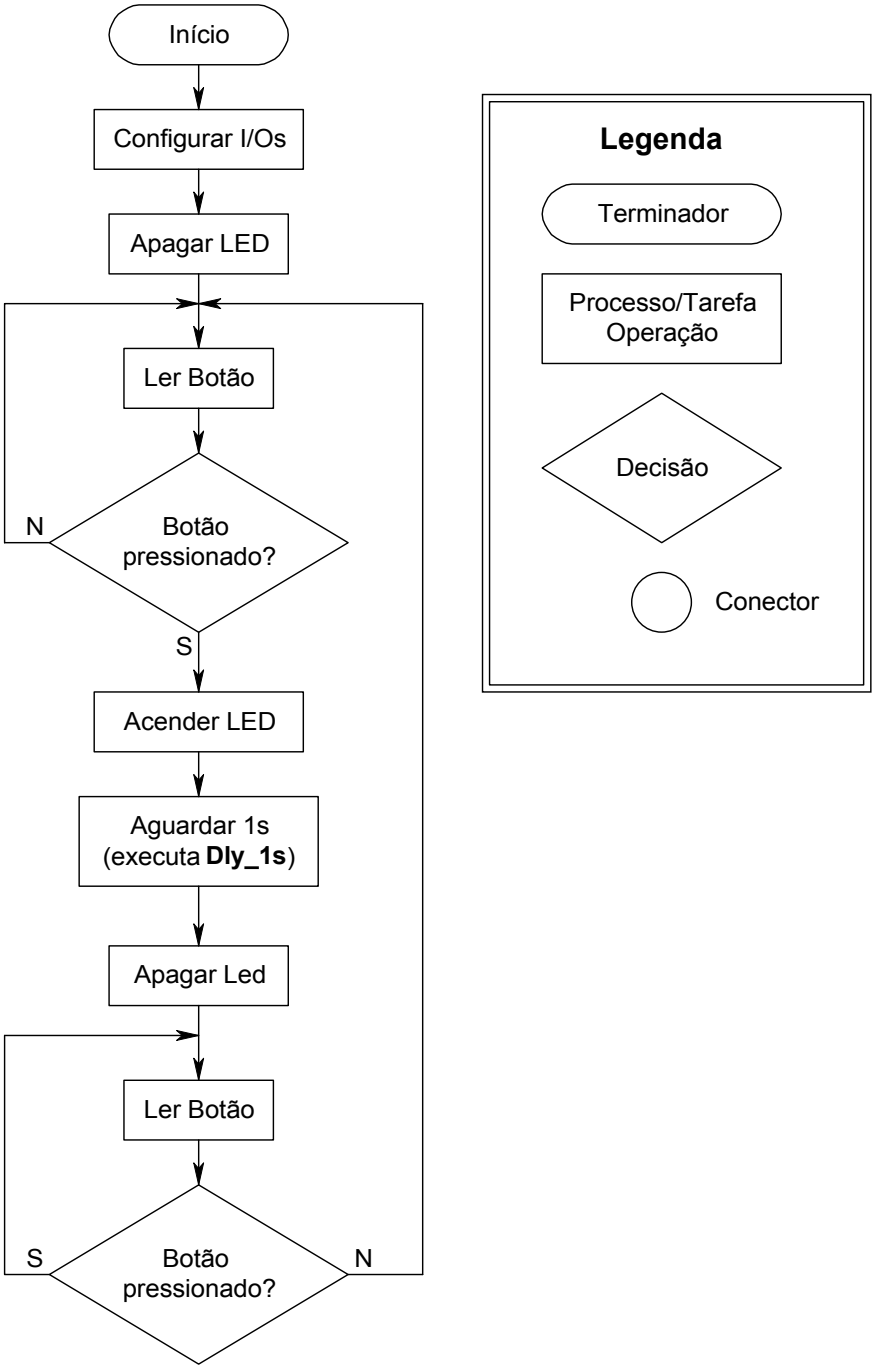

Figura 19 – Fluxograma da aplicação LED+Botão

O processo/tarefa "aguardar 1 segundo" representado na Figura 19 não é uma tarefa trivial, e pode ser expandido através de um fluxograma denominado **Dly_1seg** (Figura 20) .

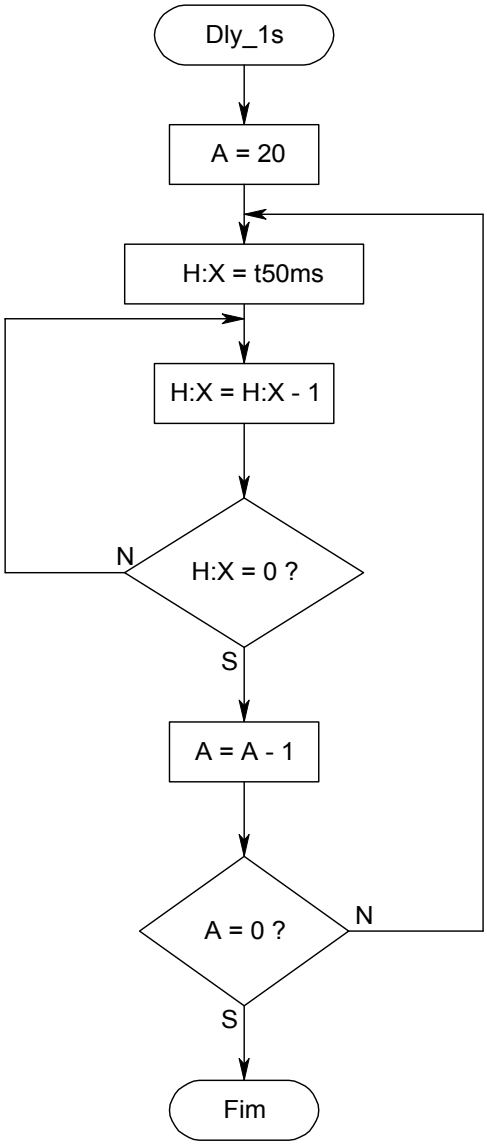

Figura 20 – Fluxograma da rotina Dly_1seg

A Figura 20 apresenta uma solução para a rotina de tempo (por instruções) que é executada em 1 segundo (20 x 50 ms), conforme especificado no início do projeto. Nesse momento é impossível dizer com exatidão qual é o valor da variável **t50ms** que fornecerá o tempo de 50ms. Esse valor depende, do tempo e do número de ciclos das instruções, do número de instruções e do número de vezes que um determinado *loop*[4] é executado (ver item 7.4).

[4] Loop – *trecho de um fluxograma/programa que é executado com retorno a um ponto/ instrução inicial.*

7.4. Codificação (Arquivo Fonte)

Após a conclusão do fluxograma, deve-se iniciar o processo denominado Codificação, que nada mais é do que escrever textualmente, através das instruções da linguagem *assembly*, os processos, tarefas e decisões representados no fluxograma, e que uma vez traduzido para códigos de máquina, deverá funcionar no hardware especificado.

Nesse ponto do desenvolvimento já foi definido o microcontrolador a ser utilizado, e conseqüentemente seus registradores, mapa de memória, etc., mas o esforço na codificação será feito com base nos fluxogramas apresentados. A utilização de diretivas do assembler serão inseridas no arquivo fonte quando o mesmo for implementado com a ferramenta de desenvolvimento *CodeWarrior*, e isso tem um motivo, como será visto no item 8.3.

A codificação do projeto proposto é apresentada pela concatenação das instruções listadas abaixo, e tem como base os fluxogramas apresentados anteriormente.

```
;*************************************************************
;                  Programa principal
;*************************************************************
INICIO:     LDA   #00001000B  ; Configura 68HC908QT1:
            STA   CONFIG2     ;  - pino /IRQ inativo
                              ;  - oscilador externo (20 MHz)
                              ;  - pino /RST inativo
            LDA   #00011001B  ;  - COP desabilitado
            STA   CONFIG1     ;  - LVI desabilitado
                              ;  - recuperacao do STOP lenta
            LDA   #$FF
            STA   PORTA
            LDA   #$02
            STA   DDRA        ; Configura I/O: PTA1 - Saida
                              ; PTA0, PTA2 a PTA5 - Entrada
            BSET  1,PORTA     ; Apaga o LED

VOLTA:
            LDA   PORTA
            AND   #$04        ; Le botao
            BNE   VOLTA       ; Se botao nao pressionado, VOLTA

            BCLR  1,PORTA     ; Acende o LED

            JSR   Dly_1s      ; Aguarda 1seg (subrotina Dly_1s)

            BSET  1,PORTA     ; Apaga o LED

FICA:       LDA   PORTA
            AND   #$04        ; Le botao
            BEQ   FICA        ; Se botao pressionado, FICA

            BRA   VOLTA       ; Senao VOLTA
```

O processo/tarefa **Dly_1s** (subrotina) é um programa relativamente pequeno, conforme código abaixo.

```
;*************************************************************
;           Subrotina de Tempo (1s) - Dly_1s
;*************************************************************
Dly_1s:     LDA   #20          ; Inicia delay de 1s (20 x 50ms)
Loop0:      LDHX  #t50ms       ; H:X = contador de tempo 50ms (NN)
Loop1:      AIX   #-1          ; H:X = H:X - 1
            CPHX  #0           ; Verifica se H:X = 0
            BNE   Loop1        ; Se H:X <> 0, volta para Loop1
            DECA
            BNE   Loop0        ; Se executou NNx20 vezes, continua
            RTS                ; Fim da subrotina
```

A rotina **Dly_1s** envolve um *loop* interno (50ms) dentro de outro *loop* externo, pois não é possível com um único *loop* conseguirmos tempos da ordem de dezenas de milissegundos. O *loop* interno consiste em instruções que decrementam o registrador H:X um determinado número de vezes, que ao final (valor $0) tenha decorrido aproximadamente 50 ms. O *loop* externo é executado 20 vezes o *loop* interno, dessa forma tem-se um tempo decorrido final de 20 x 50ms = 1 segundo. Um detalhamento de como é feito o cálculo está apresentado na Figura 21, onde o valor no interior dos círculos representa o número de ciclos de máquina para que cada instrução seja executada.

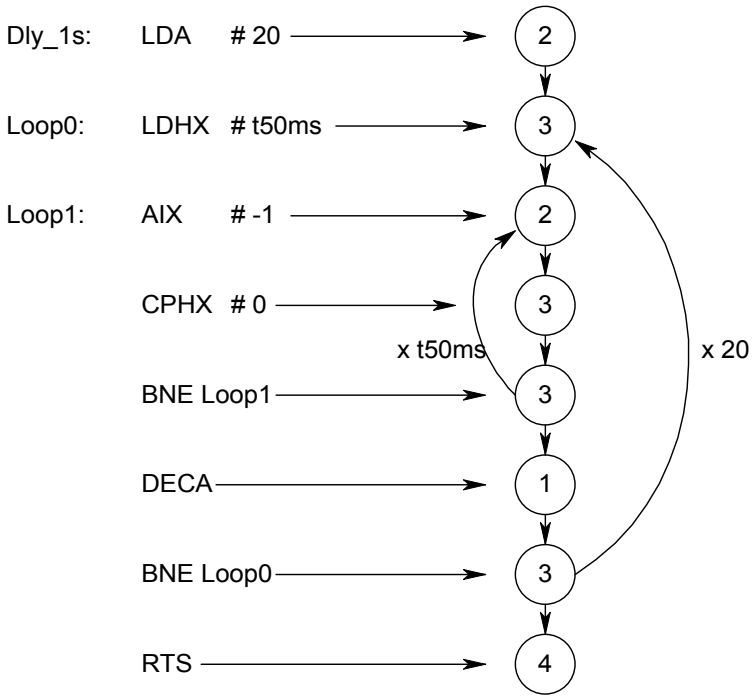

Figura 21 – Número de ciclos de máquina da subrotina Dly_1s

Pelo projeto proposto não foi definido o tipo de oscilador, portanto será utilizado um oscilador externo de freqüência igual a 20 MHz (igual ao utilizado no Kit de Desenvolvimento M68EVB908Q – ver item 8.6.2). O tempo denominado ciclo de máquina é igual ao inverso dessa freqüência vezes 4 (ou 4 períodos da freqüência dada).

O que está faltando é calcular o valor **t50ms**, tal que forneça um tempo de execução da subrotina de aproximadamente 1 segundo, e que pode ser obtido a partir da seguinte equação:

$$t = (2 + (3 + (2 + 3 + 3) \times t50ms + 1 + 3) \times 20 + 4) \times \left(\frac{4}{20 \times 10^6} \right) = 1 segundo$$

Isolando **t50ms** e fazendo os cálculos chega-se ao valor:

$$t50ms = 31249{,}0875 \cong 31249$$

Como esse valor está dentro da faixa de números de 16 bits (valor máximo: 65535) possíveis, então ele será o valor inserido no arquivo fonte no lugar de **t50ms** para que a subrotina seja executada em 1s. Se a freqüência ou a implementação das instruções forem alteradas, o cálculo do tempo de execução deverá ser refeito.

7.5. Montador Assembler

Depois de totalmente codificado, o software desenvolvido em linguagem *assembly* deve ser traduzido para seus códigos de máquina, respectivos. Esse processo é realizado por um programa computacional chamado montador assembler (ou macro assembler). Detalhes do código gerado, e que também servirá para depuração do sistema serão vistos no item 8.

7.6. Simulação, depuração e testes do sistema

Com o código executável (formato **S19** ou **ABS**) em mãos, o programador estará pronto para verificar se o código gerado executa o que idealizado inicialmente para a aplicação. Para tanto, o programador pode utilizar uma série de ferramentas de desenvolvimento que estão detalhadas no item 8.

8. FERRAMENTAS DE DESENVOLVIMENTO - *CodeWarrior*

8.1. Introdução

O *CodeWarrior* é um ambiente de desenvolvimento de software poderoso que suporta várias linguagens (*assembly*, C, C++, Java), múltiplos sistemas operacionais (Windows, PowerPCs, Solaris, Linux), para múltiplos processadores/microcontroladores (x86, Sparc, Mcore, 68HC16, 68HC12, 68HC08, 68HC05, etc). Entre as características mais importantes do *CodeWarrior* incluem-se:

- ✓ Ambiente de Desenvolvimento Integrado (IDE – *Integrated Development Environment*) que fornece acesso a todos os componentes de forma simples;
- ✓ Interface Gráfica do Usuário (GUI – *Graphical User Interface*) simples de usar;
- ✓ Gerenciador de projeto – sistema de armazenamento para todos os arquivos e opções dentro de um arquivo de projeto simples;
- ✓ Editor de código fonte – colorido, direcionado para a sintaxe, multi-janelas para edição de arquivos;
- ✓ Navegador (*browser*) – fornece acesso a vários elementos do código fonte, tais como, variáveis, rotinas, classes, etc;
- ✓ Compilador C/C++ otimizado e poderoso montador assembler;
- ✓ *Linker* inteligente – liga apenas os objetos que estão atualmente em uso no projeto;
- ✓ Bibliotecas (*librarian*)– permite ao programador construir bibliotecas customizadas;
- ✓ Depurador (*debugger*) – suporta simulação e depuração de aplicações ou projetos de hardware.

8.2. Iniciando com o *CodeWarrior* (Windows)

Na primeira vez que for utilizar o *CodeWarrior*, ou quando quiser começar um novo projeto encontre a pasta denominada Metrowerks, via botão **Iniciar > Programas > Metrowerks > *CodeWarrior* CW08 V2.1**. Dentro desta pasta, encontre e selecione o ícone "***CodeWarrior* IDE**" (Figura 22). Isto vai levar o usuário para o ambiente de desenvolvimento do *CodeWarrior*.

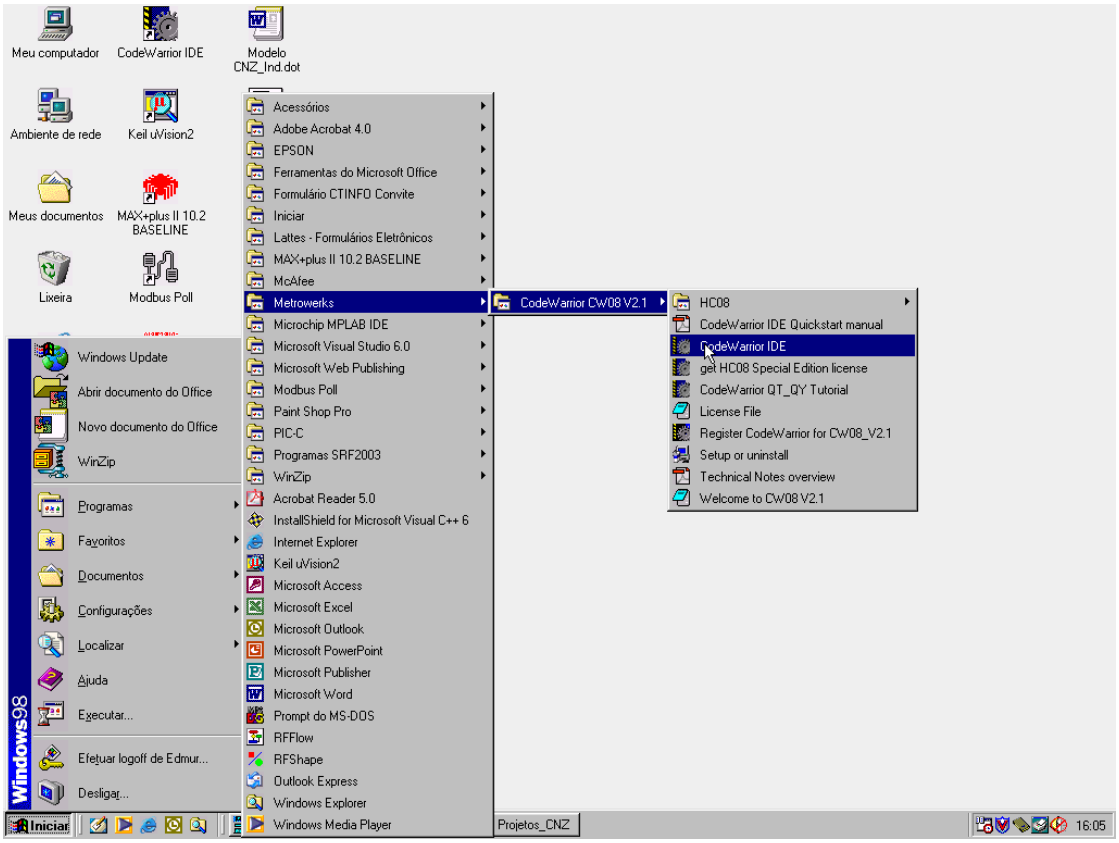

Figura 22 – Iniciando o *CodeWarrior*

A partir dessa seleção, o *CodeWarrior* será executado em uma janela com a maioria dos ícones da barra de ferramentas indisponíveis, pois ainda não existe um projeto aberto (Figura 23).

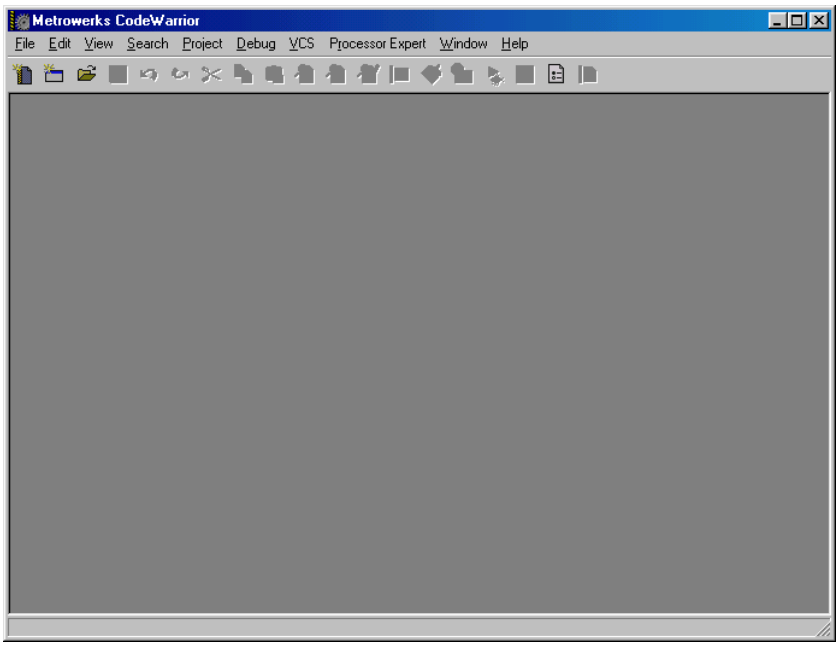

Figura 23 – Janela inicial do *CodeWarrior*

O *CodeWarrior* pode ser inicializado pela execução (duplo-click) de um arquivo de projeto (extensão .**mcp**) já existente. Esta ação abrirá o *CodeWarrior* IDE e carregará automaticamente o projeto.

A barra de ferramentas do *CodeWarrior* (Figura 24) contém botões que ativam os principais comandos/funções do ambiente de desenvolvimento, e que também estão disponíveis na barra de menus.

Figura 24 – Barra de ferramentas do *CodeWarrior*

Legenda dos botões:

1 – ***New Text File*** **(Novo arquivo texto)**
2 – ***New*** **...(Novo ... – Projeto/Arquivo/Objeto)**
3 – ***Open*** **... (Abrir ...)**
4 – ***Save*** **(Salvar)**
5 – ***Undo/Can't Undo*** **(Desfazer/Não pode desfazer)**
6 – ***Redo/Can't Redo*** **(Refazer/Não pode refazer)**
7 – ***Cut*** **(Recortar)**
8 – ***Copy*** **(Copiar)**
9 – ***Paste*** **(Colar)**
10 – ***Find*** **(Encontrar)**
11 – ***Find Next*** **(Encontrar a próxima)**
12 – ***Replace Selection*** **(Substituir seleção ...)**
13 – ***Compile*** **(Compilar código)**
14 – ***Make*** **(Construir código binário)**
15 – ***Stop Build*** **(Parar construção do código)**
16 – ***Debug*** **(Depurar)**
17 – ***Errors and Warnings*** **(Erros e Avisos)**
18 – ***Preferences*** **... (Preferências ...)**
19 – ***P&E PEDebug FCS-ICS-ICD Settings*** **... (Configuração ...)**

8.3. Criando um Projeto

Para criar um novo projeto siga o seguinte roteiro:

A. Dentro do *CodeWarrior* selecione ***File > New*** na barra de menus. Isto apresentará a caixa de diálogo ***NEW*** (Figura 25). Tenha certeza que o painel ***Project*** está ativo, clicando na aba correspondente se necessário.

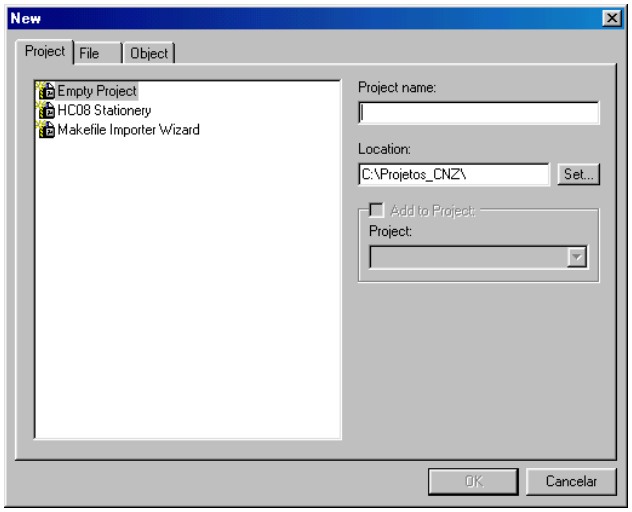

Figura 25 – Caixa de diálogo *NEW*

B. Na lista de projetos, selecione ***HC08 Stationery*** (Figura 26). Esta opção prepara o *CodeWarrior* para trabalhar com o microcontrolador e linguagem de programação selecionadas, facilitando o trabalho do programador na configuração do ambiente de trabalho.

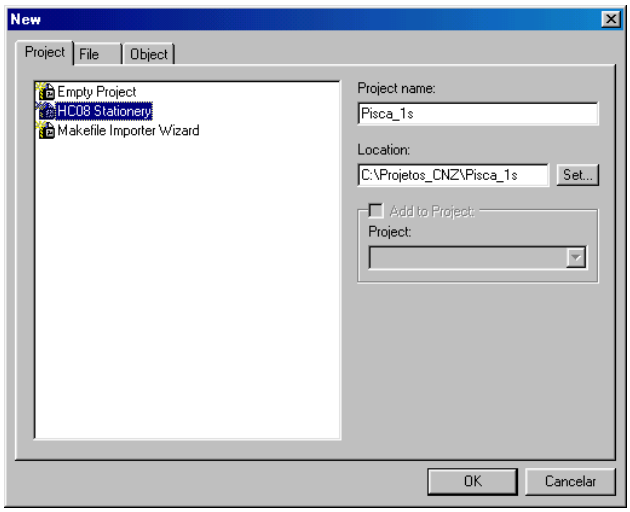

Figura 26 – Nome e pasta do novo projeto

C. No campo de ***Project name***, digite o nome do projeto novo – **Pisca_1s** (Figura 26).

D. No campo ***Location***, especifique a pasta onde estarão armazenados os arquivos utilizando o botão **Set...** (Figura 26), até criando uma pasta nova se for necessário.

E. Clique em **OK** e prossiga com a caixa de diálogos ***New Project*** (Figura 27).

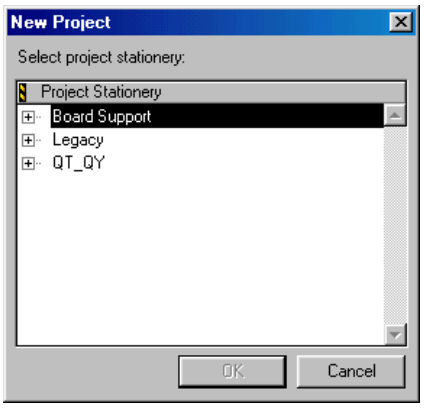

Figura 27 – Caixa de diálogo *NEW PROJECT*

F. Selecione o microcontrolador e a linguagem de programação a ser utilizada. Para o exemplo inicial selecione: **QT_QY > QT1 > *Assembly*** e pressione **OK** (Figura 28).

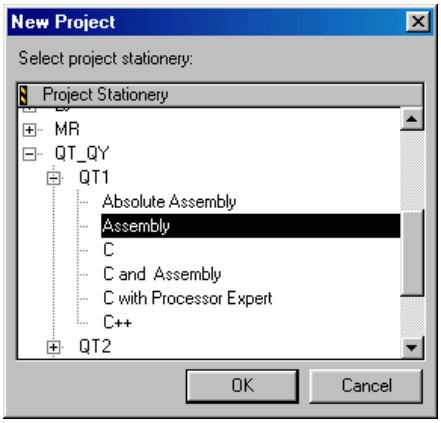

Figura 28 – *New Project - Assembly*

O *CodeWarrior* irá preparar todo o ambiente de trabalho e apresentar uma janela de arquivos e pastas do projeto recém-criado, denominado **Pisca_1s.mcp** (Figura 29). Maiores detalhes a respeito da utilidade e função de cada arquivo presente nessa janela podem ser vistos no arquivo **readme.txt** (presente na mesma janela).

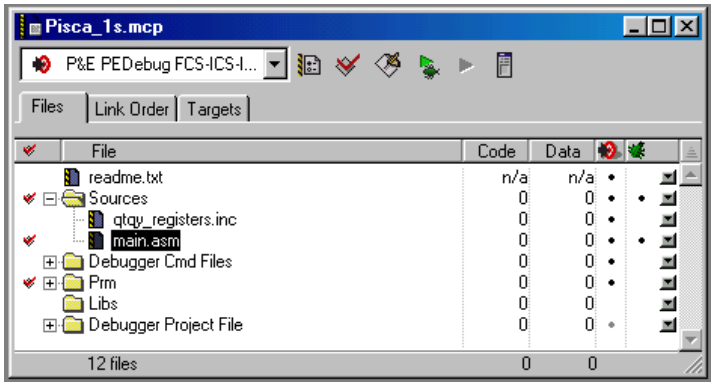

Figura 29 – Controle de arquivos e pastas do projeto (Pisca_1s.mcp)

Expandindo (sinal +) a pasta "***Sources***", confirme que um arquivo denominado "**main.asm**" está presente. Este arquivo contém um modelo (*template*) de irá auxiliar o programador com algumas instruções e pseudo-instruções necessárias e comuns a todos os projetos, que no caso são utilizados pela linguagem *assembly*. O arquivo **main.asm** está listado a seguir, e o programador tem liberdade total para alterar as linhas de código e comentários apresentados.

```
;*****************************************
;*          HC08 Demo Program            *
;*****************************************

            Include 'qtqy_registers.inc'    ; For the QTx,QYx

            XDEF Entry, main, _Startup

initStack   EQU    $00FF
Limit:      EQU    23

MyData:     SECTION  SHORT
;
; Insert here your data definition
;

MyCode:     SECTION
;
; Insert here your source code
;

_Startup:
Entry:      LDHX   #initStack   ; initialize Stack
            TXS
            CLRH
            CLI

main:       CLRX                ; REPEAT

            NOP                 ; Insert here you own code

            BRA    main         ; UNTIL Forever
            END
```

8.4. Editando um arquivo fonte

Com o arquivo **main.asm** aberto, o programador pode inserir as instruções mnemônicas que fazem parte do seu projeto. Vamos utilizar o exemplo que foi desenvolvido no item 7.4, inserindo no lugar correto e de forma adequada no arquivo fonte **main.asm (***template***)**. Ao final o arquivo fonte deverá ficar com a seguinte formatação:

```
;*************************************************************
;**********             Pisca_1s Led+Botao             **********
;*************************************************************

            Include 'qtqy_registers.inc'    ; Definicoes do QT1

            XDEF Entry, main, _Startup

initStack   EQU    $00FF

t50ms       EQU    31249        ; Valor calculado de 50ms para Dly_1s

_Startup:
Entry:      LDHX   #initStack   ; Inicializa Stack Pointer
            TXS
            CLRH
            SEI                 ; Desabilita interrupcoes

main:       CLRX                ; REPEAT

;===============================================================
;                     Programa principal
;===============================================================
INICIO:     LDA   #00001000B  ; Configura 68HC908QT1:
            STA   CONFIG2     ;  - pino /IRQ inativo
                              ;  - oscilador externo (20 MHz)
                              ;  - pino /RST inativo
            LDA   #00011001B  ;  - COP desabilitado
            STA   CONFIG1     ;  - LVI desabilitado
                              ;  - recuperacao do STOP lenta
            LDA   #$FF
            STA   PORTA
            LDA   #$02
            STA   DDRA        ; Configura I/O: PTA1 - Saida
                              ; PTA0, PTA2 a PTA5 - Entrada

            BSET  1,PORTA     ; Apaga o LED

VOLTA:
            LDA   PORTA
            AND   #$04        ; Le botao
            BNE   VOLTA       ; Se botao nao pressionado, VOLTA

            BCLR  1,PORTA     ; Acende o LED

            JSR   Dly_1s      ; Aguarda 1seg (subrotina Dly_1s)

            BSET  1,PORTA     ; Apaga o LED

FICA:       LDA   PORTA
            AND   #$04        ; Le botao
            BEQ   FICA        ; Se botao pressionado, FICA

            BRA   VOLTA       ; Senao VOLTA
```

```
;*************************************************************
;                Subrotina de Tempo (1s) - Dly_1s
;*************************************************************
Dly_1s:     LDA   #20          ; Inicia delay de 1s (20 x 50ms)
Loop0:      LDHX  #t50ms       ; H:X = contador de tempo 50ms (NN)
Loop1:      AIX   #-1       ; H:X = H:X - 1
            CPHX  #0           ; Verifica se H:X = 0
            BNE   Loop1        ; Se H:X <> 0, volta para Loop1
            DECA
            BNE   Loop0        ; Se executou NNx20 vezes, continua
            RTS                ; Fim da subrotina

            END
```

8.5. Construindo o Projeto

Durante o processo de criação de um novo projeto foi inserido no modelo de arquivo fonte (**main.asm)** o código da aplicação que deverá ser traduzido (montado) para código de máquina do microcontrolador escolhido como segue:

A. Na barra de ferramentas da janela ***Project***, clique no botão ***Make*** (ou pressione **F7**).

B. O resultado do processo de construção será mostrado na janela ***Errors& Warnings***.

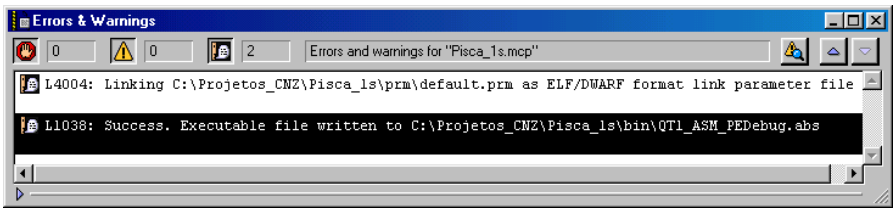

Figura 30 – *Errors & Warnings*

C. Se forem detectados erros, o programador através das mensagens na janela ***Errors & Warnings*** (Figura 30) deverá: interpretar os erros, voltar ao código fonte, corrigi-los e construir novamente o código executável. Essas etapas devem ser repetidas enquanto o processo de construção indicar a existência de erros. Ao final a janela deverá apresentar 0 erros (*errors*), 0 avisos (*warnings*), se possível e N mensagens de não erros.

Nota: Um programa construídos sem erros (0 Errors) não indica que ele esteja correto. No processo de construção do código executável o assembler verifica apenas a sintaxe das linhas do arquivo fonte. Os erros/falhas de software são percebidos durante a simulação/depuração/execução do código executável.

8.6. Simulando e Depurando um Projeto

Um projeto construído com sucesso gerará um arquivo do código objeto absoluto (normalmente no formato **S19**) que poderá ser carregado nas ferramentas de simulação/emulação para avaliação, testes e depuração do código. Para simular um código executável utilizando as ferramentas de simulação do *CodeWarrior*, o programador deverá seguir o seguinte procedimento:

A. Mantenha o projeto aberto ou abra-o se ele estiver fechado. No exemplo desenvolvido, mantenha o projeto **Pisca_1s.mcp** aberto e construído sem erros.

B. Ative o "*True-Time Simulator & Real-Time Debugger"* clicando no botão ***Debug*** da barra de ferramentas do projeto (ou pressione **F5**).

Se a mensagem da Figura 31 surgir durante o processo de execução das ferramentas de simulação, significa que não foi instalada a licença de avaliação (*evaluation license*) solicitada pela Metrowerks. Contudo, o software vai rodar no modo demonstração (*demo mode*) com todos os componentes habilitados, mas para um código de no máximo 1 Kb. Maiores informações sobre a licença de uso do *CodeWarrior* e as formas de obtê-la estão disponíveis no arquivo **welcome_CW08_V2_1_1.txt,** que pode ser acessado via botão **Iniciar > Programas > Metrowerks > *CodeWarrior* CW08 V2.1 > Welcome to CW08 2.1**.

Figura 31 – Aviso sobre licença das ferramentas de simulação

Independente de ter uma licença registrada ou não, o simulador é carregado com o código executável do projeto que estava aberto, conforme Figura 32.

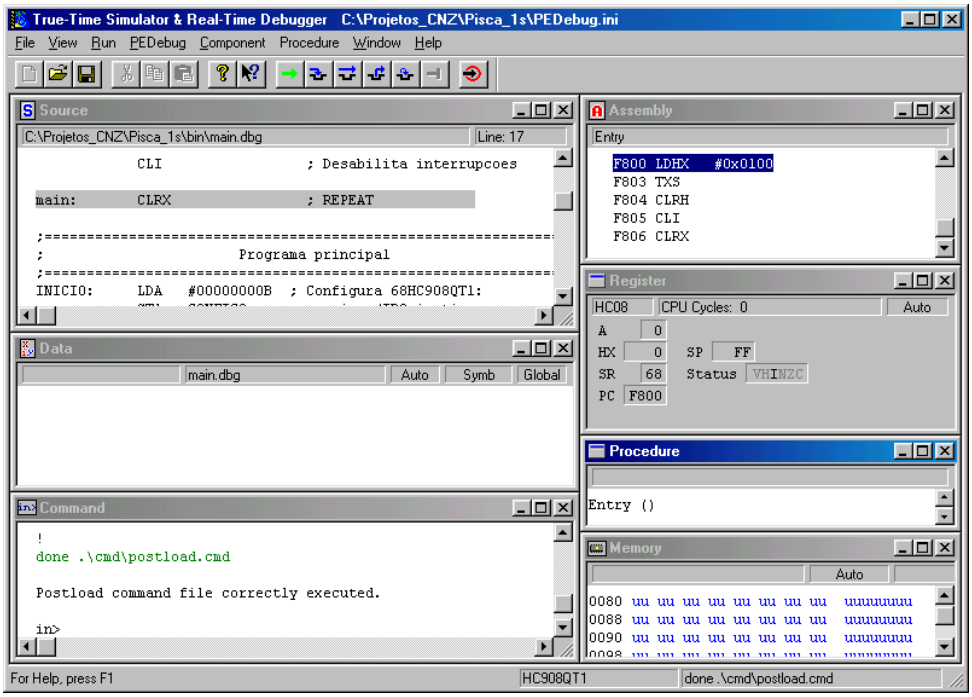

Figura 32 – Janela inicial do simulador

O software do simulador é composto de múltiplas janelas que podem ser organizadas a qualquer momento e da forma mais conveniente pelo programador. A aparência padrão dessas janelas estão apresentadas na Figura 32. São elas:

- ✓ **Source** – Arquivo fonte do projeto a ser simulado.
- ✓ **Data** – Variáveis utilizadas pelo projeto.
- ✓ **Command** – Comandos implementados pelo simulador durante a simulação.
- ✓ ***Assembly*** – Código de máquina do projeto a ser simulado.
- ✓ ***Register*** – Registradores do modelo de programação da CPU08 com conteúdo atualizável durante a simulação.
- ✓ **Procedure** – Função/subrotina que está sendo executada momentaneamente pelo simulador.
- ✓ **Memory** – Mapa de memória do microcontrolador com conteúdo atualizável durante a simulação.

Na primeira vez que um projeto é simulado, o programador deve se certificar que o microcontrolador que é alvo de simulação, esteja selecionado corretamente. Nas seções de simulação subseqüentes do mesmo projeto, a configuração do ambiente de simulação obtida a partir da última seção de simulação realizada.

Para verificar/atualizar o microcontrolador utilizado na simulação, o usuário deve selecionar na barra de menus do simulador: **PEDebug > Device:<***dispositivo***> > 68HC08 > QT/QY Family > HC908QT1** (Figura 33).

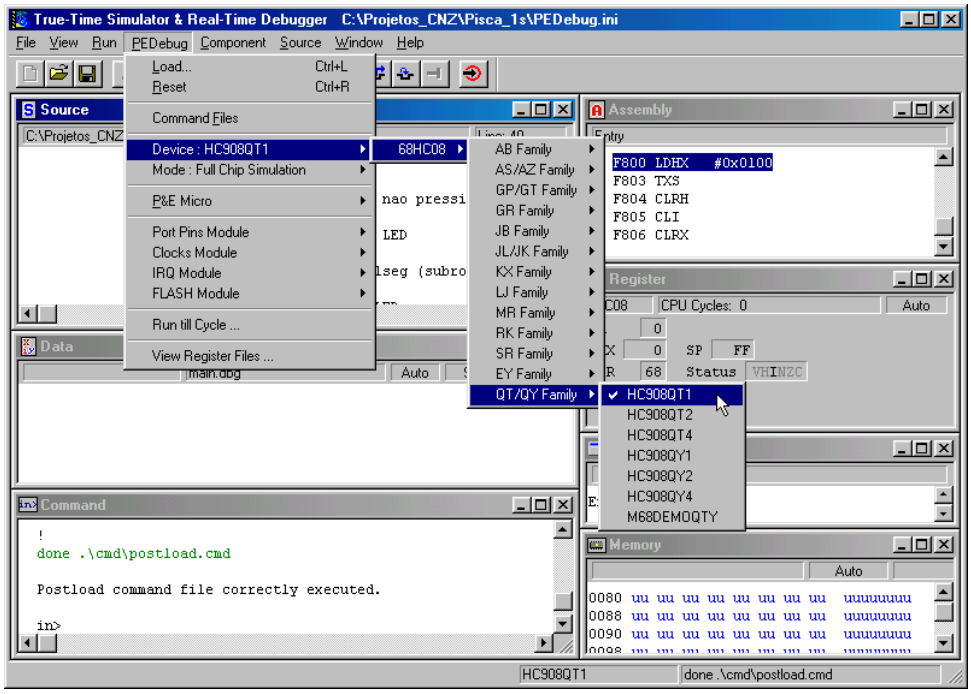

Figura 33 – Simulador – Menu de escolha do dispositivo.

Uma vez selecionado o dispositivo correto, o usuário pode iniciar a tarefa de simulação/depuração de código. Por se tratar de um software poderoso, a empresa PE Systems (proprietária do simulador instalado no *CodeWarrior*) disponibilizou para os usuários três modos diferentes de simulação de sistema. São eles:

- ✓ Simulação Completa do Chip (*FCS – Full Chip Simulation*)
- ✓ Simulação *In-Circuit* (*ICS – In-Circuit Simulation*)
- ✓ Depuração/Programação *In-Circuit* (*ICD – In-Circuit Debug/Programming*)

O modo de simulação/depuração pode ser selecionado na barra de menus do simulador: **PEDebug > Modo:<***modo de simulação***>** (Figura 34).

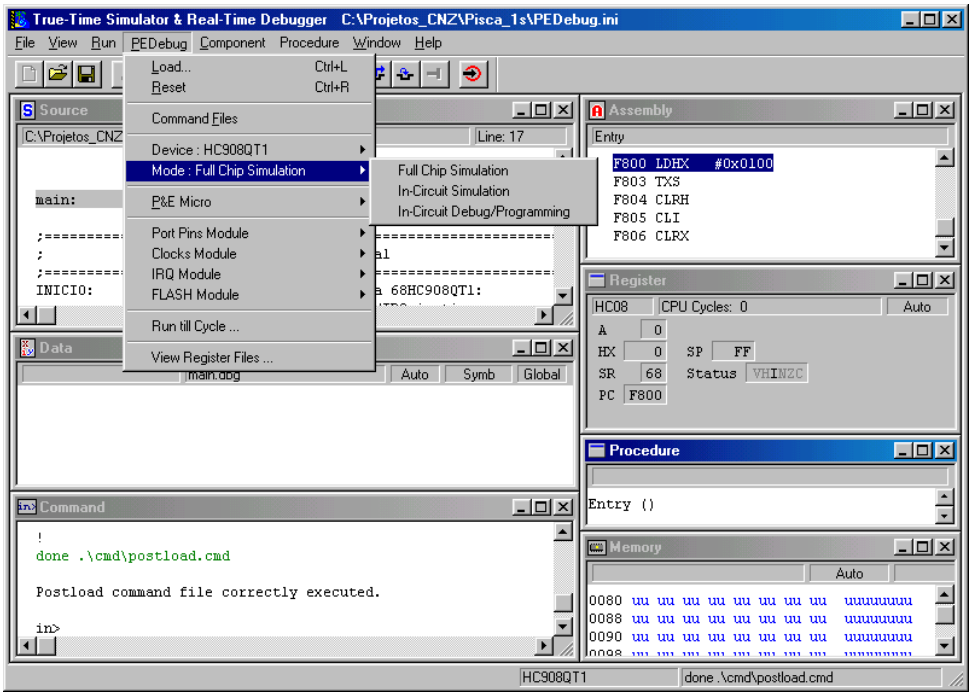

Figura 34 – Simulador – Modos de simulação

No modo FCS, a simulação do software do projeto é todo totalmente realizado pelo simulador (software aplicativo), não existindo comunicação com hardware.

No modo ICS, a simulação do software do projeto é feita pelo microcontrolador disponível no hardware alvo (na memória RAM), através de uma interface de comunicação serial com o modo Monitor ROM. Cada instrução é transferida para o hardware, executada pelo microcontrolador, e o resultado dos registradores/memória é atualizado no simulador.

No modo ICD, a depuração/programação é realizada na memória de programa do microcontrolador (hardware), ou seja, o software do projeto é totalmente transferido (programado) para a memória FLASH do microcontrolador disponível no hardware. O usuário através das janelas do simulador/depurador controla as atividades de depuração no hardware. O hardware recebe comandos através da interface serial controlado pelo modo Monitor ROM, executa o comando pedido utilizando o programa na FLASH, e retorna dados dos registradores/memória para atualizar nas janelas do simulador. A grande vantagem do modo ICD é a execução do software em **tempo real.**

Todos os modos de simulação/depuração disponíveis utilizam a mesma interface gráfica de usuário. A barra de menus contém os principais comandos do simulador, conforme apresentado na Figura 35. Maiores detalhes podem ser vistos no menu Ajuda do simulador.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Figura 35 – Simulador – Barra de Ferramentas

Legenda dos botões:

1 – ***New*** **...(Novo ...)**
2 – ***Open*** **... (Abrir ...)**
3 – ***Save*** **(Salvar)**
4 – ***Cut*** **(Recortar)**
5 – ***Copy*** **(Copiar)**
6 – ***Paste*** **(Colar)**
7 – **Help Topics (Tópicos de Ajuda)**
8 – **Help (Ajuda)**
9 – **Start/Continue (Iniciar/Continuar) – F5**
10 – **Single Step (Passo simples) – F11**
11 – **Step Over (Passo sobre) – F10**
12 – **Step Out (Passo para fora) – Shift+F11**
13 – ***Assembly*** **Step (Passo do *assembly*) – Ctrl+F11**
14 – **Halt (Parar) – F6**
15 – **Reset Target (Reseta Target) – Ctrl+R**

Os botões numerados de 9 a 15 executam funções específicas do simulador que refletem sobre o funcionamento do microcontrolador, de acordo com os passos do software implementado. Uma descrição mais detalhada de cada botão é apresentada a seguir:

- ✓ ***Start/Continue*** – Esta função inicia a execução da aplicação (software) a partir do endereço dado pelo PC (linha que está ativa na janela ***Source***) até uma das seguintes condições seja satisfeita: encontrar um *breakpoint*, detectar um erro de *runtime*, ou a aplicação ser paralisada (suspensa) pelo usuário com a função ***Halt***.
- ✓ ***Single Step*** – Se a aplicação estiver suspensa, este comando realiza a execução de uma única declaração do código fonte em linguagem de alto nível ou uma instrução quando o código fonte estiver em *assembly*. Se a declaração corrente for uma chamada a uma função (subrotina), o simulador entra (step into) na função. O comando ***Single Step*** não trata uma função como um comando, portanto executa o passo entrando na função.

- ✓ ***Step Over*** – Comando similar ao ***Single Step***, mas não entra dentro de funções chamadas. Uma chamada de função (subrotina) é tratada como um único comando, porém todos os registradores/memória afetados são atualizados no simulador.
- ✓ ***Step Out*** – Se a aplicação for suspensa dentro de uma função, este comando continua a execução, e então para na instrução seguinte a chamada da função (executa a chamada da função até o seu retorno a instrução subseqüente à chamada). Se não houver chamadas de função presentes, então a função ***Step Out*** não é realizada.
- ✓ ***Assembly Step*** – Se a aplicação estiver suspensa, este comando realiza um passo correspondente a uma instrução em *assembly*. Este comando é similar ao comando ***Single Step***, mas executa apenas uma instrução em assembly ao invés de uma declaração em linguagens de alto nível.
- ✓ ***Halt*** – Interrompe e suspende a execução da aplicação. Quando a aplicação se encontrar nesse estado, o usuário poderá examinar o estado de cada variável da aplicação, setar *breakpoints*, e inspecionar o código fonte.
- ✓ ***Reset Target*** – Reseta o ambiente de Simulação/Depuração.

Além dos comandos listados acima, existe um comando muito útil na simulação/depuração de códigos denominado *breakpoint*. Uma breve descrição é apresentada a seguir:

- ✓ ***Breakpoints ...*** – *Breakpoints* são pontos de controle associado ao valor do PC, isto é, a execução do programa é suspensa assim que o PC alcançar o valor definido no *breakpoint*. O modo mais simples para setar um *breakpoint* é clicar o botão da direita do mouse sobre a linha na qual o usuário quer o ponto de parada (na janela ***Source***), e no menu selecionar ***Set Breakpoints***. Uma linha que contenha um *breakpoint* contém uma marca em vermelho (Figura 36) indicando o ponto de parada. Para remover o *breakpoint* basta clicar com o botão da direita do mouse sobre a linha na qual o usuário quer remover o ponto de parada (na janela ***Source***), e no menu selecionar ***Delete Breakpoint***.

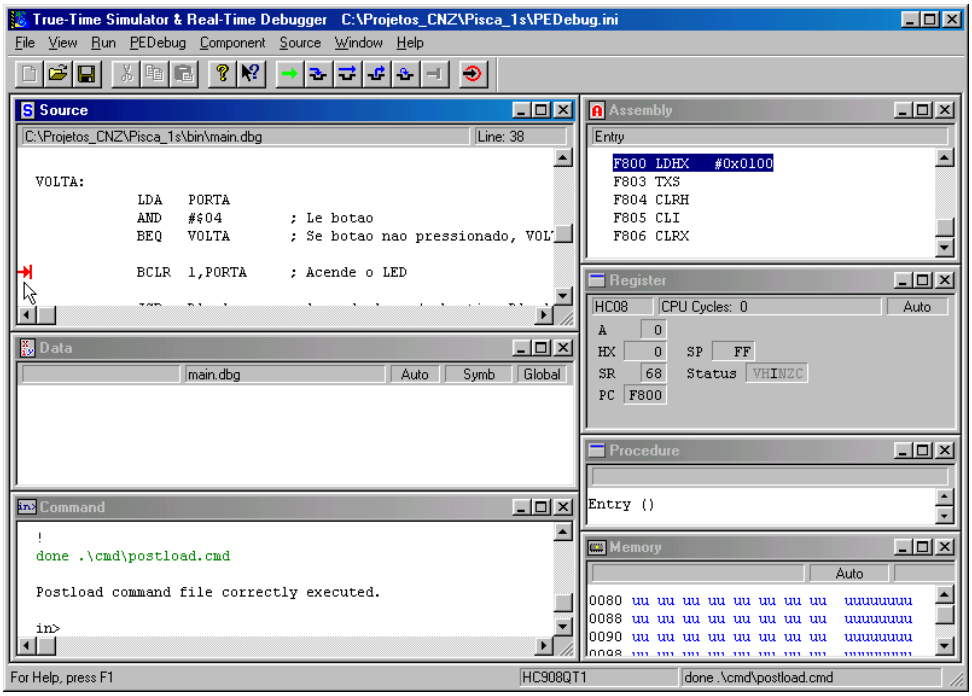

Figura 36 – *Breakpoint* (marca em vermelho) na janela *Source*

8.6.1. Simulação Completa do Chip (FCS)

A Simulação Completa do Chip (FCS) é totalmente executada pelo software do simulador. Para iniciar no modo FCS siga o seguinte roteiro:

A. Selecione o modo: *In-Circuit Simulation* (ICS) como apresentado no item 8.6.

B. Na barra de ferramentas dessa nova janela, clique no botão ***Reset Target*** (ou pressione **Ctrl-R**). Note que a janela ***Procedure*** mostrará a função "***Entry* ()"** e a janela ***Source*** mostrará o começo do código da função ***Entry***.

C. Através dos botões de simulação descritos na Figura 35 vá descobrindo as mudanças/atualizações (marcadas em vermelho) que cada instrução executada na janela ***Source*** ou ***Assembly*** provocam no simulador. A janela ***Register*** será atualizada refletindo o fato do código ter sido executado uma instrução**.**

D. Valores dos registradores (janela ***Register***) e/ou conteúdo de posições de memória (janela ***Memory***) podem ter seus valores modificados, de acordo com as necessidades da simulação/depuração. Para tanto, o usuário deverá selecionar a posição/registrador a ser alterado (duplo-click), digitar o valor requerido,pressionar <*Enter*>, e continuar o processo de simulação.

8.6.2. Simulação In-Circuit (ICS)

A Simulação *In-Circuit* (ICS – *In-Circuit Simulation*) utiliza a funcionalidade do modo Monitor que é um modo especial da família HC08. Este modo de funcionamento permite a um computador externo controlar o microcontrolador através de uma interface serial assíncrona. Esta característica permite ao computador solicitar e modificar o estado do processador, bem como carregar, depurar e programar o código executável – dando aos microcontroladores M68HC908QT/QY a capacidade de serem completamente testados e programados através de uma interface simples sem serem removidos do sistema ao qual estão embarcados. Entre as principais características do Monitor ROM:

- ✓ Funcionalidade dos pinos do microcontrolador normal durante o modo usuário.
- ✓ 1 pino dedicado para comunicação serial entre o Monitor e o computador externo.
- ✓ Execução do código em RAM ou FLASH
- ✓ Capacidade de proteção da memória FLASH
- ✓ Interface de programação da memória FLASH
- ✓ Execução do código em RAM ou FLASH
- ✓ Características de segurança da memória FLASH
- ✓ Interface de programação da memória FLASH

As características de segurança nos microcontroladores da família HC08 existem para proteger um dispositivo programado de serem lidos e disassemblados por pessoas não autorizadas. Este mecanismo é constituído de 8 bytes de código de segurança que estão armazenados nos endereços de $FFF6 a $FFFD. Se um usuário conhece o que está programado no dispositivo, ele implicitamente conhece o código para desbloquear a segurança necessário para poder ter acesso ao modo Monitor e ter acesso as memórias ROM/FLASH. Contudo, para os usuários que não conhecem o código de segurança, eles podem entrar no modo monitor mas sem acesso as memória ROM/FLASH.

A verificação do software e a correspondência/compatibilidade do hardware podem ser facilmente desempenhadas, uma vez que os efeitos das instruções de software no ambiente de simulação pode ser observável diretamente no hardware. Isto faz da depuração um processo simples, e a placa de desenvolvimento M68EVB908Q torna tudo isso possível.

Uma vez que o código fonte de um programa foi construído com sucesso (existência de um arquivo S19), o simulador carrega o arquivo na RAM do microcontrolador depois de entrar no modo monitor. A comunicação com o simulador é realizada através de um canal serial (padrão RS232) com o chip através do pino PTA0 e vice-versa, portanto cada instrução executada no ambiente de simulação é imediatamente refletida no hardware.

Para realizar a Simulação *In-Circuit*, conecte o computador e a placa M68EVB908Q através de um cabo serial, e siga o seguinte roteiro:

A. Selecione o modo: *In-Circuit Simulation* como apresentado na Figura 34.

B. Na primeira vez e/ou sempre que houver algum problema de comunicação, o simulador abrirá uma janela com o conteúdo apresentado na Figura 37. Se não for a primeira vez, e não houver problemas de comunicação o simulador prepara todo o ambiente para iniciar a simulação, sem abrir essa janela.

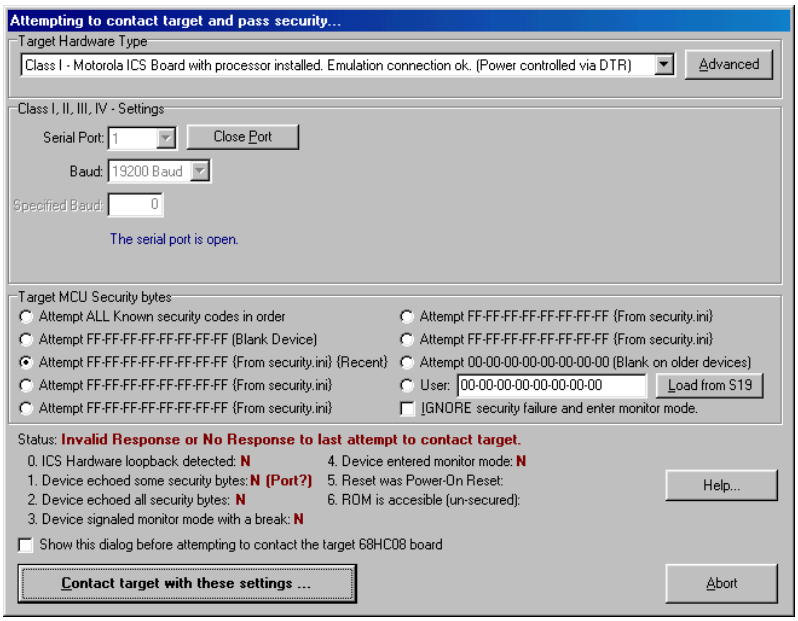

Figura 37 – ICS – Conexão serial e código de segurança

C. Para utilizar a placa de desenvolvimento M68EVB908Q no modo ICS, o usuário deverá selecionar o *Target Hardware Type* - **CLASS I** (*Motorola ICS Board with processor installed. Emulation connection OK. Power controlled via DTR*).

D. Selecionar a comunicação serial através de um port disponível (COM1, COM2, ...) e uma taxa de comunicação de 19200 bps. Na placa M68EVB908Q os jumpers J5 e J6 deverão estar fechados, e o jumper J3 deverá estar na posição 1-2 (Maiores detalhes podem ser encontrados no manual do usuário da placa M68EVB908Q.

E. Se o usuário não programou os bytes de segurança de código, selecionar uma das opções com FF-FF-FF-FF-FF-FF-FF-FF (8 bytes com $FF).

F. Pressionar o botão '*Contact Target with these settings ...*'. Isto permitirá a entrada no modo ICS, e na janela de *Commands* o usuário verá entre outras, as seguintes mensagens:

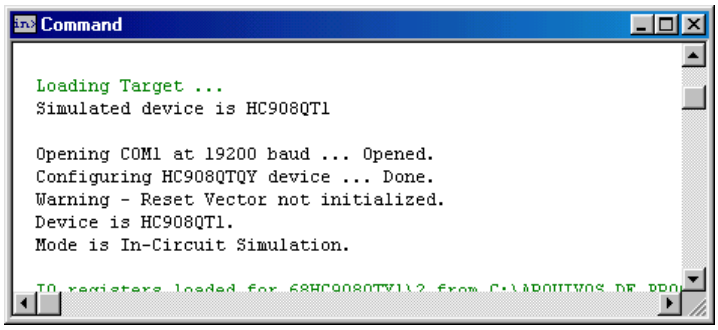

Figura 38 – Janela de Comandos – Simulação *In-Circuit*

G. A partir desse ponto tudo é idêntico ao descrito na Simulação Completa do Chip (*Full Chip Simulation*), exceto as interrupções e os valores dos pinos que não podem ser setados/resetados pela linha de comando, pois todos os valores são determinados pelo hardware. O pino /IRQ (ou o botão SW1) da placa de desenvolvimento M68EVB908Q pode ser utilizado para iniciar uma interrupção depois de executado o protocolo inicial.

Se contudo o usuário não consegui passar pelo modo de segurança, observe que na caixa de diálogos *Target Connection and Security* tem uma seção denominada "STATUS" descrevendo os diferentes tipos de falhas, e o que verificar em cada caso. As razões mais comuns para não passar pelos códigos de segurança são:

- ✓ Não foi escolhido o código de segurança correto;
- ✓ Na re-energização, a tensão no microcontrolador não desceu abaixo de 0.1volts (durante o desligamento).
- ✓ A classe do hardware (*Target hardware type*) não foi escolhida corretamente.

8.6.3. Depuração/Programação In-Circuit (ICD)

A Depuração/Programação *In-Circuit* (*ICD – In-Circuit Debug/Programming*) é muito semelhante a Simulação *In-Circuit*, com a seguinte diferença básica: no modo ICD o código executável (formato S19) é programado na memória FLASH do microcontrolador e toda a depuração do código é feita utilizando o hardware alvo.

Para realizar a depuração/programação no modo ICD, conecte o computador e a placa M68EVB908Q através de um cabo serial, e siga o seguinte roteiro:

A. Selecione o modo: *In-Circuit Debug/Programming* como apresentado na Figura 34.

B. Na primeira vez e/ou sempre que houver algum problema de comunicação, o simulador abrirá uma janela com o conteúdo apresentado na Figura 39. Se não for a primeira vez, e não houver problemas de comunicação o simulador prepara todo o ambiente para iniciar a simulação, sem abrir essa janela.

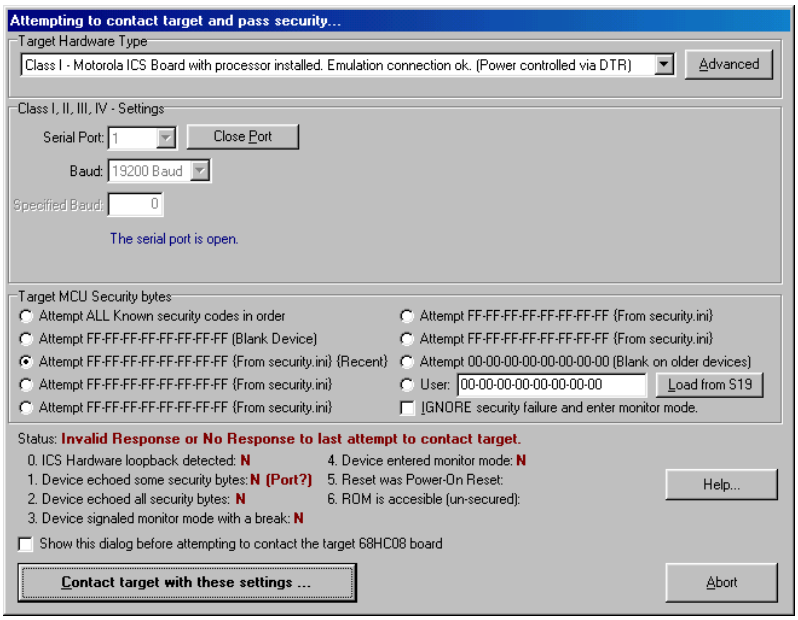

Figura 39 – ICS – Conexão serial e código de segurança

C. Para utilizar a placa de desenvolvimento M68EVB908Q no modo ICS, o usuário deverá selecionar o *Target Hardware Type* - **CLASS I** (*Motorola ICS Board with processor installed. Emulation connection OK. Power controlled via DTR*).

D. Selecionar a comunicação serial através de um port disponível (COM1, COM2, ...) e uma taxa de comunicação de 19200 bps. Na placa M68EVB908Q os jumpers J5 e J6 deverão estar fechados, e o jumper J3 deverá estar na posição 1-2 (Maiores detalhes podem ser encontrados no manual do usuário da placa M68EVB908Q).

E. Se o usuário não programou os bytes de segurança de código, selecionar uma das opções com FF-FF-FF-FF-FF-FF-FF-FF (8 bytes com $FF).

F. Pressionar o botão '*Contact Target with these settings ...*'. Isto iniciará a comunicação para entrar no modo ICD.

G. Se o ambiente do simulador estava utilizando outro modo de simulação, surgirá uma caixa de diálogo (Figura 40) perguntando ao usuário se ele deseja recarregar os dados do projeto. O usuário deverá pressionar o botão "*Yes*".

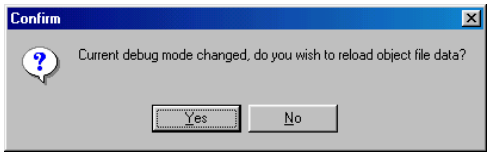

Figura 40 – Mudança no modo de simulação

H. Uma nova caixa de diálogo (Figura 41) surgirá perguntando ao usuário se ele deseja apagar e programar a memória FLASH. O usuário deve responder "*Yes*", e a partir desse ponto o código executável é gravado na memória FLASH.

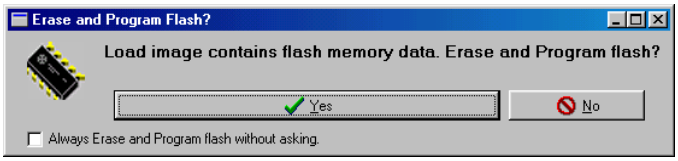

Figura 41 – Apagar e Programar a memória FLASH

I. O processo de apagamento e programação da memória FLASH é iniciado. O usuário pode verificar a seqüência de comandos transferidos entre o computador e o hardware alvo, conforme apresentado na Figura 42.

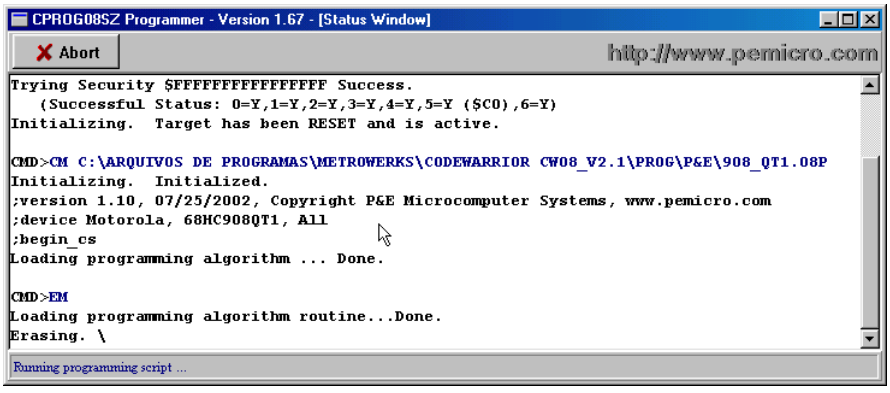

Figura 42 – Apagando e programando a memória FLASH

J. A partir desse ponto tudo é idêntico ao descrito na Simulação Completa do Chip (*Full Chip Simulation*), exceto as interrupções e os valores dos pinos que não podem ser setados/resetados pela linha de comando, pois todos os valores são determinados pelo hardware. O pino /IRQ (ou o botão SW1) da placa de desenvolvimento M68EVB908Q pode ser utilizado para iniciar uma interrupção depois de executado o protocolo inicial.

Se contudo o usuário não consegui passar pelo modo de segurança, observe que na caixa de diálogos *Target Connection and Security* tem uma seção denominada "STATUS" descrevendo os diferentes tipos de falhas, e o que verificar em cada caso. As razões mais comuns para não passar pelos códigos de segurança são:

- ✓ Não foi escolhido o código de segurança correto;
- ✓ Na re-energização, a tensão no microcontrolador não desceu abaixo de 0.1volts (durante o desligamento).
- ✓ A classe do hardware (*Target Hardware Type*) não foi escolhida corretamente.

A partir desse ponto, o usuário poderá seguir o caminho sozinho utilizando o *CodeWarrior* para desenvolvimento e depuração de seus projetos. Para maiores informações, consulte os arquivos de documentação incluídos na subpasta ***Documentation*** da pasta ***Metrowerks***. Os seguintes arquivos em PDF são de particular interesse:

- ✓ *CodeWarrior IDE Quickstart manual*
- ✓ *Motorola M68HC08 manuals overview*

9. REFERÊNCIAS BIBLIOGRÁFICAS

[1] CPU08 – Reference Manual – Rev. 1
Motorola Inc.

[2] MC68HC908QY4/D – Data Sheet - Rev. 0
Motorola Inc.

[3] MC68HC908QY4/D – Advance Information
Motorola Inc.

[4] Motorola's 8-bit FLASH MCU Seminar Series
Motorola Inc.

[5] Apostila do Curso de Arquitetura de Computadores - FEV/2000
Edmur Canzian

[6] Apostila do Curso Sistemas Digitais Microprocessados – OUT/2001
Edmur Canzian

[7] *CodeWarrior* Development Studio for HC08 Microcontrollers Special Edition
Metrowerks Corp.

[8] Quickstart for Embedded System – Featuring Hiware Technology
Metrowerks Corp.

[9] *CodeWarrior* IDE 4.2 – Users Guide
Metrowerks Corp.

[10] AN2305 – User Mode Monitor Access for MC68HC908QY/QT Series MCUs
Motorola Inc.

[11] AN2317/D – Low-Cost Programming and Debugging Options for M68HC08 MCUs
Motorola Inc.

[12] Placa de Desenvolvimento M68EVB908Q – Manual do Usuário
Motorola Industrial Ltda.

GLOSSÁRIO

#xxxx – Os dígitos que seguem o prefixo "#" indicam um operando imediato.

$xxxx – Os dígitos que seguem o prefixo "$" estão em formato hexadecimal.

ALU – Arithmetic/Logic Unit (ver Unidade Lógica/Aritmética).

Assembly – ver Linguagem *Assembly*.

Barramento (**Bus**) – Uma coleção de linhas lógicas usadas para transferir informações simultaneamente.

Barramento de Dados (**Data Bus**) – Conjunto de linhas utilizadas para transportar informações binárias da CPU para a memória ou da memória para a CPU.

Barramento de endereços (**address bus**) – Conjunto de linhas utilizadas para selecionar uma posição específica na memória, para que a CPU possa escrever informações ou ler seu conteúdo.

BCD – Binary-Coded Decimal (ver Decimal Codificado em Binário).

Bit – Digito binário simples. Um bit pode armazenar os valores 0 ou 1.

Byte – Um conjunto de 8 bits.

CCR – *Condition Code Register* (ver Registrador *Condition Code*).

Chip – Componente eletrônico.

Ciclos da CPU – Período de *clock* do barramento interno, normalmente derivado da divisão de um cristal oscilador por 2 ou mais. O tempo necessário para executar uma instrução é medido em ciclos da CPU.

Clock - Sinal em freqüência utilizado para marcar a seqüência de eventos pela CPU.

Código de máquina – Códigos binários processados pela CPU como instruções. Códigos de máquina incluem os *opcodes* e operandos de dados.

Conjunto de instruções – Conjunto de todas as operações que a CPU pode realizar. Uma instrução é freqüentemente representada por mnemônicos, tais como LDA (Carrega o Acumulador). Outra representação do conjunto de instruções é o conjunto de *opcodes* que são reconhecidos pela CPU.

CPU – Central Processor Unit (ver Unidade Central de Processamento).

CPU08 – Unidade Central de Processamento dos microcontroladores da família HC08

Decimal Codificado em Binário – Notação que utiliza valores binários para representar quantidades decimais. Cada digito BCD é representado por 4 bits, e as 6 últimas representações possíveis das 16 combinações (1010 até 1111) são consideradas ilegais.

Depuração/Programação *In-Circuit* – Ferramenta de depuração e programação onde um software aplicativo se comunica com um microcontrolador, tem a capacidade de gravar sua memória FLASH e executar e depurar comandos no próprio chip.

EEPROM – Electrically Erasable, Programmable Read-Only-Memory. Tipo de memória não volátil que pode ser apagada e reprogramada por instruções de programa.

Entrada/Saída – Sinais que fazem a interface entre o microcontrolador e o mundo externo. A CPU lê uma entrada para sentir o nível de um sinal externo e escreve em uma saída para mudar o nível em um sinal externo.

EPROM – Erasable, Programmable, Read-Only Memory. Tipo de memória não volátil que pode ser apagada pela exposição em luz ultra-violeta.

FCS – *Full Chip Simulation* (ver Simulação Completa do Chip).

Firmware – Software básico que pode ser gravado na memória de programa de microprocessadores ou microcontroladores.

Flag – Bit que tem uma função específica.

Hardware – Circuitos eletrônicos.

I/O – Input/Output (ver Entrada/Saída).

I/O Mapeado em Memória – Neste tipo de sistema, registradores de controle e I/Os são acessados da mesma maneira que na memória RAM ou ROM. As instruções podem acessar a memória, e também podem acessar registradores de I/O.

ICD – *In-Circuit Debug/Programming* (ver Depuração/Programação In-Circuit).

ICS – *In-Circuit Simulation* (ver Simulação *In-Circuit*).

Instruções – Operações que a CPU pode realizar. Instruções são expressadas por programadores como mnemônicos da linguagem *Assembly*.

Interrupção – As interrupções fornecem um meio de suspender temporariamente a execução normal do programa para liberar a CPU a executar um conjunto de instruções (serviço) em resposta ao pedido (interrupção) de um periférico.

Latch – Circuito lógico que mantém uma saída estável mesmo depois que a entrada for removida ou alterada. Uma entrada de *clock* determina quando o *latch* irá capturar o estado da entrada e aplicá-lo na saída.

Linguagem *Assembly* – Método utilizado por programadores para representar instruções de máquina (dados em binário) em uma forma mais conveniente. Para cada instrução de máquina é dado um nome curto e simples, denominado mnemômico.

Mapa de memória – Representação visual de toda a memória endereçável pela CPU.

Microcontrolador – Sistema computacional completo, incluindo a CPU, memória, *clock* , e I/O em um simples circuito integrado.

Mnemônico – Letras que representam uma operação da CPU. Por exemplo, o mnemônico da instrução "Carregar o Acumulador" é LDA.

Modelo de Programação – Registradores de uma CPU em particular.

Modos de endereçamento – Meio pelo qual a CPU obtém (endereça) a informação necessária para completar uma instrução.

Nível lógico 0 – Nível de tensão aproximadamente igual a tensão do terra (V_{SS}).

Nível lógico 1 – Nível de tensão aproximadamente igual a tensão de alimentação (V_{DD}).

Offset – Deslocamento (reposicionamento) de um ponteiro em relação a um endereço base, que normalmente é o Registrador de Índice ou o *Stack Pointer*.

Opcode – Código binário que instrui a CPU a fazer uma operação específica.

Oscilador – Circuito que produz uma freqüência constante que é utilizada pela CPU para temporizações e referência de sequeciamento.

OTP – One Time Programmable. Tipo de memória não volátil que pode ser programada mas não pode ser apagada. A memória recebe esse nome porque não existem formas de apagar o seu conteúdo.

Página Zero – Primeiros 256 bytes de memória ($0000 - $00FF), também chamada de página direta.

PC – Program Counter. Registrador que aponta para o endereço do byte de código a ser buscado na memória. O mesmo que Contador do Programa.

Programa – Conjunto de instruções que fazem o microcontrolador desempenhar uma operação desejada.

RAM – Random Access Memory. Tipo de memória que pode ser lida ou escrita pela CPU. O conteúdo de uma posição de memória RAM permanece armazenado até que a CPU escreva um valor diferente ou até que a energia seja desligada.

Registrador *Condition Code*– Registrador que informa o status de operações lógicas e aritméticas.

Reset – Reset é utilizado para forçar a CPU e periféricos para um estado inicial conhecido.

Reset(ar) - Colocar um sinal/bit em GND (nível lógico 0).

Set(ar) – Colocar um sinal/bit em VCC (nível lógico 1).

Simulação Completa do Chip – Ferramenta de depuração onde o microcontrolador tem toda sua funcionalidade simulada por um software aplicativo em microcomputador pessoal.

Simulação *In-Circuit* – Ferramenta de depuração onde um software aplicativo se comunica com um microcontrolador (hardware), envia comandos, executa comandos no microcontrolador e recebe respostas dos comandos para atualização do simulador.

Software – Conjunto de instruções computacionais ordenadas de tal forma que executar as funções necessárias a uma aplicação.

SP - *Stack Pointer*. Registrador que contém o próximo endereço (ponteiro) livre da pilha, o mesmo que Ponteiro da Pilha.

Unidade Central de Processamento – Unidade que controla a execução de instruções.

Unidade Lógica/Aritmética – Módulo de uma CPU onde são realizadas as operações aritméticas e lógicas.

CNZ

EMPRESA

A CNZ é uma empresa (registrada no CREA-SP) voltada para Projetos, Consultoria, Assessoria e Treinamento nas áreas de Sistemas Eletrônicos Digitais Microprocessados e de Informática Industrial. Sua principal missão é o treinamento de profissionais envolvidos com o desenvolvimento de projetos de sistemas digitais e a prestação de serviços para empresas que atuam na área de Automação e Controle, e também de Equipamentos Biomédicos e Laboratoriais. A CNZ desenvolve integralmente o projeto, fabrica e comercializa produtos de acordo com as necessidades pré-determinadas pelo cliente.

PRODUTOS

Kits de desenvolvimento para microcontroladores 68HC908QT/QY

Sistema de aquisição de dados compactos

Conversor de protocolos seriais

Interfaces de comunicação

Equipamentos biomédicos

TREINAMENTO

Microcontroladores da família HC08 – Módulo Básico

Microcontroladores da família HC08 – Módulo Avançado

Interfaces e protocolos de comunicação serial para microcontroladores

Linguagem C para microcontroladores

CNZ Engenharia e Informática Ltda.
Av. Estácio de Sá, 560 – Cotia – SP
Fone: +55 (11) 4612-2324
Homepage : www.cnz.com.br

CNZ Indústria e Comércio Ltda.
Av. Lineu Prestes, 2242 – São Paulo - SP
Fone: +55 (11) 3039-8343
Homepage : www.cnz.com.br

DISTRIBUIDORES AUTORIZADOS MOTOROLA SEMICONDUTORES

AVNET

Fone : 11- 5079-2150
Fax : 11- 5079-2160
e-mail : vendas@avnet.com
web page : http://www.avnet.com

FARNELL

Fone : 11- 4066-9400
Fax : 11- 4066-9410
e-mail : vendas@farnell.com
web page : http://www.farnell.com

KARIMEX COMPONENTES ELETRONICOS LTDA

Fone : 11- 5189-1900
Fax : 11- 5181-9380
e-mail : vendas@karimex.com
web page : http://www.karimex.com.br

INFORMAT COMPONENTES ELETRONICOS LTDA

Fone : 11- 3350-0200
Fax : 11- 3223-0892
e-mail : gustavo@grupoinformat.com.br
web page : http://www.grupoinformat.com.br/informat/

PANAMERICANA / ARROW

Fone : 11- 3613-9300
Fax : 11- 3613-9355
e-mail : sac@pan-arrow.com.br
web page : http://www.pan-arrow.com.br

RICHARDSON ELETRONICS

Fone : 11 - 3845-6199
Fax : 11 - 3845-6199
e-mail : otavio@rell.com
web page : http://www.rell.com

SILETEC ELETRONICA COM. LTDA

Fone : 11- 5536-4401
Fax : 11- 5535-4997
e-mail : siletec@siletec.com.br
web page : http://www.siletec.com.br

FUTURE ELECTRONICS

Fone : 19 - 3737-4100

Fax : 19 - 3236-9834

e-mail : rafael.armani@future.ca

web page : http://www.futureelectronics.com/