UNIVERSIDADE SÃO FRANCISCO

CURSO DE ENGENHARIA ELÉTRICA

"TRAVESSIA TRANQÜILA" – SISTEMA DE CONTROLE DE SEMÁFOROS PARA PESSOAS COM DEFICIÊNCIA FÍSICA

Área de Microcontroladores

por

Thiago Couto Zanin
RA: 3210005

Jorge Salomão Pereira, Msc.
Orientador

Itatiba (SP), novembro de 2004

UNIVERSIDADE SÃO FRANCISCO

CURSO DE ENGENHARIA ELÉTRICA

"TRAVESSIA TRANQÜILA" – SISTEMA DE CONTROLE DE SEMÁFOROS PARA PESSOAS COM DEFICIÊNCIA FÍSICA

Área de Microcontroladores

por

Thiago Couto Zanin
RA: 3210005

Relatório apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Engenharia Elétrica para análise e aprovação.
Orientador: Jorge Salomão Pereira, Msc.

Itatiba (SP), novembro de 2004

AGRADECIMENTOS

Agradeço este trabalho imensamente ao meu orientador Msc. Jorge Salomão Pereira pela ajuda e incentivo.

Agradeço aos amigos que participaram direta e os que participaram indiretamente para a conclusão deste trabalho.

Agradeço a minha noiva pela cooperação neste período de dedicação ao projeto.

Mas quando deres um banquete,
convida os pobres, os aleijados,
os mancos e os cegos;
e serás bem-aventurado;
porque eles não têm com que te retribuir;
pois retribuído te será na ressurreição dos justos.
Lucas 14:13-14

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Zanin, Thiago Couto. “Travessia Tranqüila” – Sistema de Controle de Semáforos para Pessoas com Deficiência Física**.** Itatiba, 2004. Trabalho de Conclusão de Curso, Universidade São Francisco, Itatiba, 2004.

Neste trabalho apresenta-se todo o desenvolvimento de um sistema capaz de controlar semáforos em um cruzamento. Neste, também foi planejada uma forma de pessoas com deficiência física acessarem e alterarem eletronicamente os tempos de travessia, sendo assim beneficiadas.
A parte principal do controle é realizada por um microcontrolador, que segue uma rotina programada, também apresentada aqui. Um segundo microcontrolador faz o papel de alterar os tempos de comutação das lâmpadas dos semáforos, ficando com as pessoas que utilizarão o serviço.
O componente utilizado é o MC68HC908QY4 da família Motorola M68HC08, comumente denominada HC08. Este microcontrolador possui uma grande gama de aplicações, se enquadrando no foco do projeto, além de ter um custo bem reduzido, o que justifica a sua escolha.
A programação destes CI’s é toda descrita neste documento, incluindo o desenvolvimento das rotinas em linguagem Assembly e também a gravação nos chip’s.
As saídas do microcontrolador principal são observadas em corrente contínua, precisando de um circuito capaz de acionar as lâmpadas em corrente alternada que irão compor os semáforos. Este circuito de saída é a última parte descrita neste projeto.

Palavras-chave: Controle de Semáforos. Deficiência Física. Microcontrolador HC08.

ABSTRACT

Zanin, Thiago Couto. "Travessia Tranqüila" – Sistema de Controle de Semáforos para Pessoas com Deficiência Física. Itatiba, 2004. Trabalho de Conclusão de Curso, Universidade São Francisco, Itatiba, 2004.

It will be presented the development of a system that is able to control traffic lights at the cross. Also, a way to help people with physical deficiency electronically change the time to cross is implemented.
A microcontroller (MCU), based on a routine, programmed into flash technology, is the main device in this project. A second microcontroller, owned by the deficient people, changes the light time commutation.
This MCU is the MC68HC908QY4, from MOTOROLA M68HC08 family, usually called HC08. It is used in several applications, which is the aim of this project. Moreover, due to the low cost and reduced size its choice was justified.
The MCU programming, including source code development, is also described.
For last, but not least, the traffic light controls, based on power electronics, will be mentioned.

Keywords: Control traffic lights. Physical Deficiency. Microcontroller HC08.

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 maquinas estão 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.

Tendo em vista este desenvolvimento e na tentativa de implementar uma aplicação que auxilie as pessoas com deficiência física, foi idealizado então o: “Travessia Tranqüila” – sistema de controle de semáforos para pessoas com deficiência física, o qual terá sua completa descrição no decorrer deste documento.

Para a realização deste projeto foi utilizada a família Motorola M68HC08, comumente denominada HC08, que contém microcontroladores de propósito geral com largas possibilidades de aplicação. Este documento fornecerá aos seus leitores uma introdução a 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.

Por se tratar de um material técnico que proporcione a outros pesquisadores uma fonte de pesquisa fiel, serão abordados tópicos referentes ao modelo de programação, modos de endereçamento e o conjunto de instruções dos microcontroladores da família HC08.

1.1. OBJETIVOS

1.1.1. Objetivo Geral

Diante da situação de que o tempo dos semáforos é fixo, e em muitos casos esse tempo é curto demais para a travessia do pedestre, as pessoas devem atravessar a via rapidamente, antes que o semáforo abra e o fluxo de veículos recomece.

Como principal foco, este projeto visa dar um maior conforto para os pedestres que realmente necessitam.

A idéia é desenvolver um circuito eletrônico responsável pela interrupção do tráfego de veículos na via, possibilitando à pessoa uma travessia mais tranqüila, ou seja, devido às suas limitações físicas, o pedestre em questão precisa de um tempo mais prolongado para atravessar.

A pessoa interessada na utilização desta funcionalidade deve ser cadastrada, e assim recebe um “cartão” ou “chave” que o habilitará a utilizar o sistema empregado nos semáforos das vias escolhidas.

1.1.2. Objetivos Específicos

A idéia é desenvolver um circuito eletrônico responsável pela interrupção do tráfego de veículos na via, possibilitando:

- Uma travessia mais tranqüila aos pedestres; e
- Diminuição de acidentes e atropelamentos.

1.2. METODOLOGIA

Para a realização deste projeto foi preparada uma seqüência dos acontecimentos, ajudando a controlar o cronograma das atividades a serem cumpridas.

Abaixo estão listados os principais pontos:

1. Identificação do problema/necessidade
2. Definição do que precisa ser estudado
3. Estudo do Microcontrolador

4. Programação Inicial (microcontrolador principal)
5. Testes via software
6. Testes via hardware
7. Programação Final (dos dois microcontroladores)
8. Testes via software
9. Testes via hardware
10. Desenho da placa de circuito impresso – Chave Eletrônica
11. Montagem da placa – Chave Eletrônica
12. Testes via software
13. Testes via hardware
14. Maquete

Todos estes tópicos foram trabalhados durante o projeto, demandando um tempo inicial no estudo do microcontrolador e também na programação do mesmo, que exigem maior nível de dedicação, já que são a base do projeto.

1.3. ESTRUTURA DO TRABALHO

O primeiro aspecto detalhado no trabalho é sobre a teoria do microcontrolador utilizado, o que dispensa maiores comentários.

Toda a seqüência mostrada na metodologia tem um objetivo final que é a implementação do projeto, e por isso não será descrito item por item neste documento, mesmo porque existem itens repetidos e de melhorias.

O documento consistirá de todos estes tópicos direcionados para o final do projeto, reunindo em um único item o que foi realizado diversas vezes, como por exemplo, a programação e os testes.

2. FUNDAMENTAÇÃO TEÓRICA

A grande demanda de tempo e esforço neste projeto tem relação com o microcontrolador em questão, e por isso, o estudo de suas características e funções se faz necessário neste primeiro momento.

Toda a teoria vista nesta etapa será utilizada para a realização do trabalho, por isso segue uma descrição do microcontrolador com um foco nas funcionalidades requisitadas.

2.1. O MICROCONTROLADOR

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 desenvolvimento de rotinas.

2.1.1. 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.

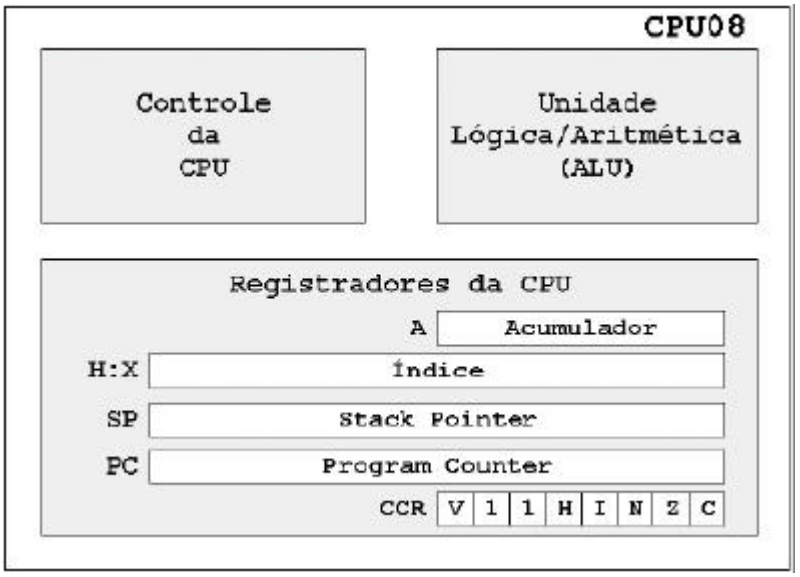


Figura 1. Diagrama de blocos da CPU08
Fonte: Adaptado de CNZ Engenharia (2003)

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 é baseada em algoritmos de adição e subtração (adição com o valor negativo). A multiplicação e 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 seqüeciamento 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 à 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).

2.1.2. Sistema de Sincronismo - *Clock*

Todo sistema computacional utiliza um relógio de sincronismo, clock, para fornecer à 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 usada para controlar o sequeciamento das instruções da CPU. Normalmente as CPU's dividem a freqüência básica do cristal por 2 ou mais para chegar ao clock do barramento interno. Cada ciclo de leitura ou escrita na memória leva um ciclo de clock do barramento interno, também denominado ciclo de barramento (bus cycle).

2.1.3. 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 a EPROM quanto à 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 apagado e reprogramado 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 diferente da CPU.

2.1.4. Sinais de Entrada

Dispositivos de entrada fornecem informação para a CPU processar, vindas do mundo externo. A maioria das entradas que os microcontroladores processam são denominadas sinais de entrada digitais, e utilizam níveis de tensão compatíveis com a fonte de alimentação do sistema. O sinal de 0V (GND ou Vss) indica o nível lógico 0 e o sinal de fonte positiva, que tipicamente é +5V DC (VDD) 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 VDD e Vss. Outros dispositivos de entrada convertem os sinais analógicos em sinais digitais (valores binários formados por 0's e 1's) que a CPU pode entender e manipular. Alguns microcontroladores incluem circuitos conversores analógicos/digitais (ADC) encapsulados no mesmo componente.

2.1.5. Sinais de Saída

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

2.1.6. 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.

2.1.7. 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 é usado.

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.

2.2. DESCRIÇÃO FUNCIONAL

Os microcontroladores que fazem parte da família MC68HC908QT/QY têm 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

Fonte: Adaptado de CNZ Engenharia (2003)

2.2.1. Características principais

?? Núc1eo da CPU de alto desempenho HC08

?? Tensão de operação (VDD) de 5V e 3V

?? 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

?? Programação "in-system" 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

?? 5 ou 13 linhas de entrada/saída (I/O) bidirecionais e mais 1 entrada

- Capacidade de alta corrente dreno/fonte em todos os pinos
- Pull-ups selecionáveis individualmente em todos os pinos

?? Registradores de I/O mapeados em memória

2.2.2. Pinagem

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

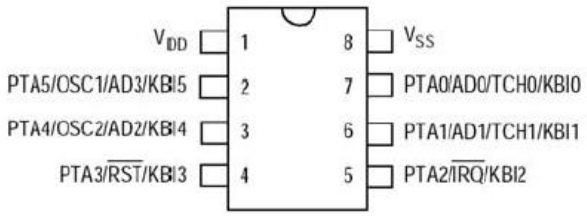


Figura 2. Pinagem dos microcontroladores MC68HC908QTlIQT2/QT4 - PDIP/SOIC
Fonte: Adaptado de CNZ Engenharia (2003)

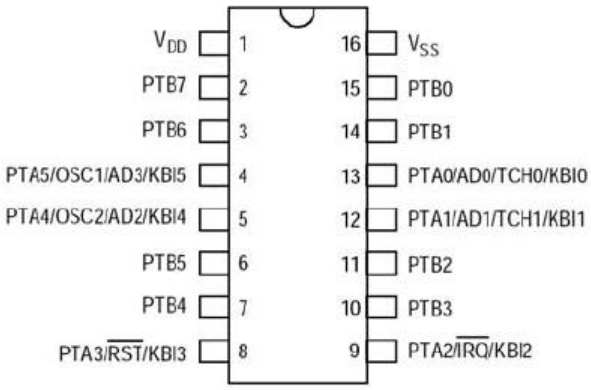


Figura 3. Pinagem dos microcontroladores MC68HC908QYlIQY2/QY4 - PDIP/SOIC
Fonte: Adaptado de CNZ Engenharia (2003)

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

Tabela 2. Descrição dos pinos dos microcontroladores MC68HC908QT/Y

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

Fonte: Adaptado de CNZ Engenharia (2003)

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

2.2.3. Memória

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

Figura 4. Mapeamento da memória da família MC68HC908QT/QY

Fonte: Adaptado de CNZ Engenharia (2003)

2.2.3.1. 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 sub-rotina a CPU utiliza 2 bytes da pilha para guardar o endereço de retorno.

2.2.3.2. 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/QYl e para o MC68HC908QT2/QY2
- $FFDO - $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.

2.2.4. 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 freqüência de um cristal oscilador externo, então o ciclo de execução será um quarto da freqüência do cristal. Isto é denominado ciclo de barramento ou ciclo da CPU. Todas as instruções tem sua execução especificada em número de ciclos da CPU. Dessa forma, se o hardware utilizar um cristal externo de 32MHz, a freqüência do barramento será de 8 MHz. Portanto, instruções de um ciclo são executadas em 125ns.

Este módulo é utilizado para fornecer urna 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 OSCI.
- ?? RC externo: Esta opção utiliza um resistor externo (R) para gerar urna freqüência. O capacitor é interno ao chip.
- ?? Cristal externo: Módulo oscilador interno ao chip que necessita um cristal externo ou ressonador cerâmico.

2.2.4.1. 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.

2.2.5. 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 Code Warrior 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 = freqüência/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 (VTST) 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.

2.2.6. Portas de Entrada/saída (I/O)

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

2.2.6.1. 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 como entrada.

2.2.6.2. 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.

3. PROJETO

Com o advento da indústria automobilística e sua modernização, o número de veículos trafegando pelas ruas e avenidas aumentou inesperadamente. Neste momento foi necessário implementar um controle de tráfego de veículos para permitir que pedestres e automóveis pudessem circular juntos e ainda promover a segurança dos veículos nos cruzamentos, evitando acidentes.

Desta forma, surgiu o semáforo, que a partir de um código de cores controla a circulação dos automóveis pelas ruas e avenidas, manipulando o fluxo entre elas e também zelando pela segurança de pedestres que desejam atravessar de um lado para outro das vias de circulação.

Hoje o semáforo faz parte da rotina de todo motorista, estando este sujeito à punições legais em casos de desrespeito ao semáforo.

Sobre o código de cores que o semáforo utiliza, a cor vermelha indica fluxo não permitido na via de circulação, ou seja, os veículos devem parar perante esta situação. Neste caso, a travessia de pedestres na rua é alta. A cor verde indica fluxo permitido na via, permitindo que os automóveis prossigam em seu trajeto e os pedestres não devem atravessar. A cor amarela significa que o motorista deve ter atenção, pois em poucos instantes o semáforo irá interromper o fluxo de veículos pela via de circulação, ou seja, retornará à cor vermelha.

O tempo que cada cor aparece para o motorista depende da necessidade de controle de tráfego da via de circulação. Uma avenida muito movimentada precisa ter mais tempo de semáforo aberto para o fluxo e menos tempo de semáforo fechado, para evitar grandes congestionamentos. Este tempo é definido no circuito do semáforo e não pode ser alterado.

Abaixo, um diagrama de estados mostra como é o funcionamento de um semáforo tradicional.

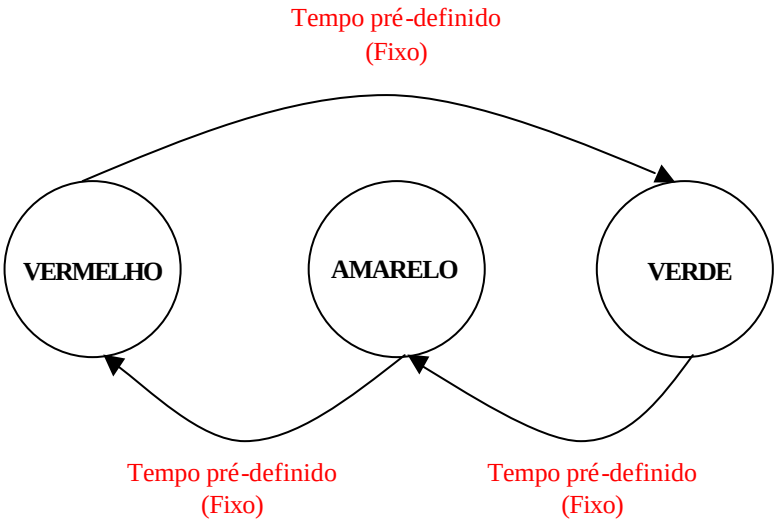


Figura 5. Diagrama de um semáforo tradicional

O projeto consistirá no controle dos semáforos do cruzamento, bem como a permissão de um maior tempo de passagem para o pedestre, quando possuir uma "chave eletrônica" que acesse o circuito principal.

Abaixo se encontra um modelo para o projeto:

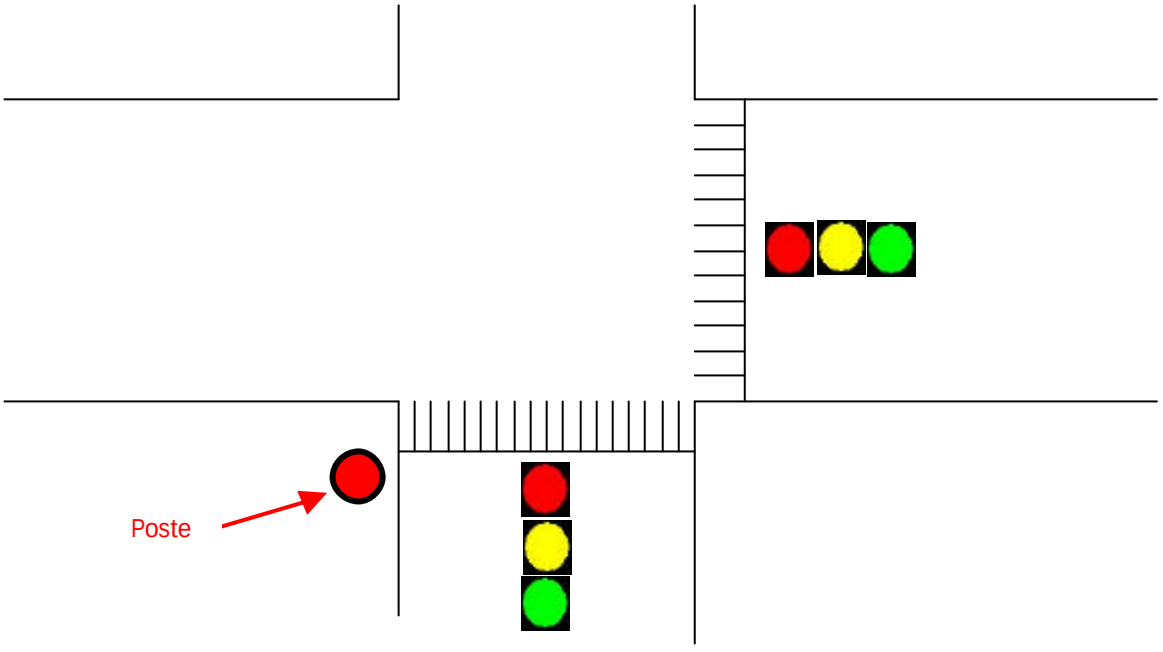


Figura 6. Vista do cruzamento / Circuito principal no poste

3.1. DIAGRAMA DE BLOCOS

Abaixo se encontra o diagrama de blocos do projeto:

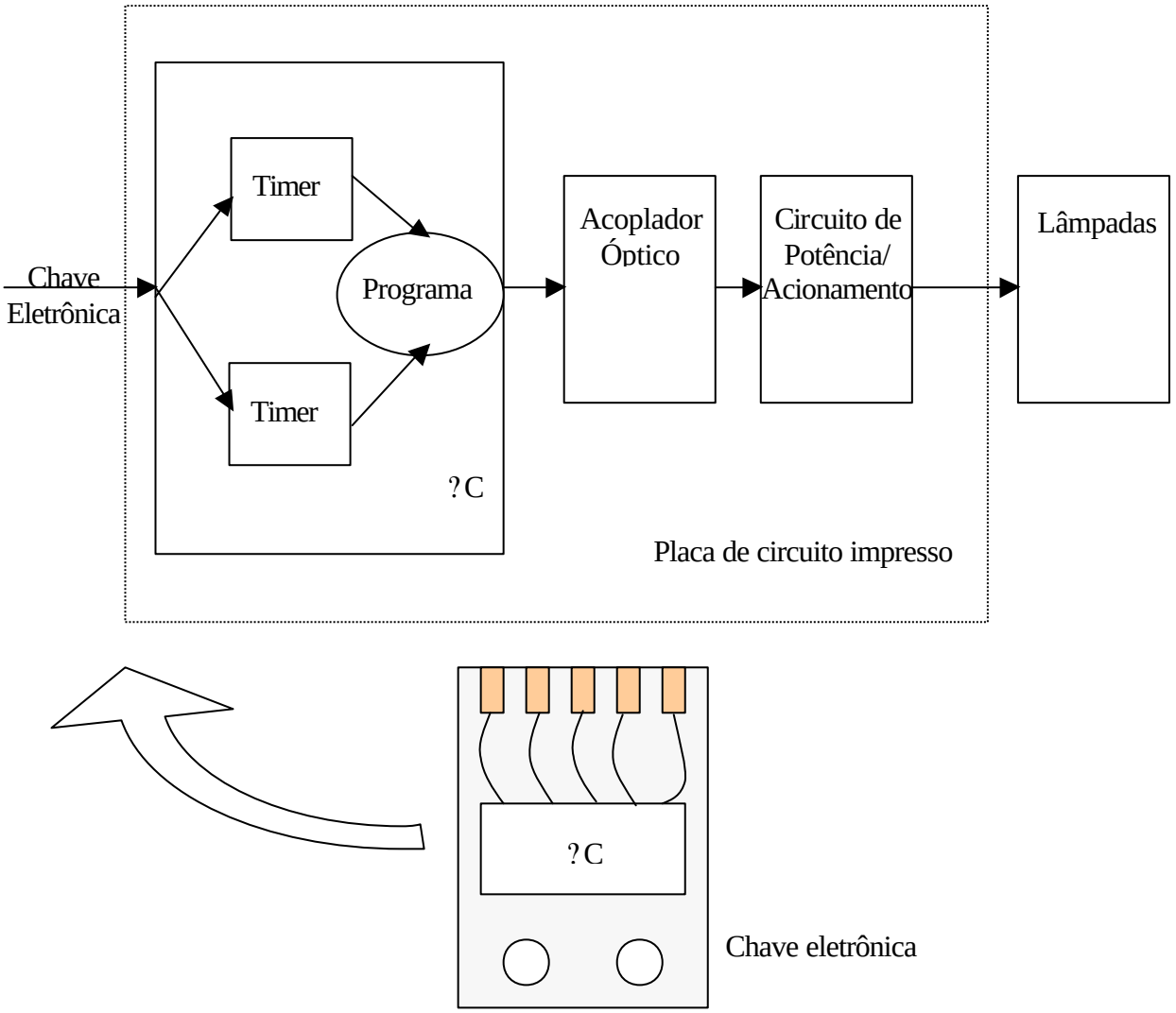


Figura 7. Diagrama de blocos do projeto

O microcontrolador principal é o que controla todas as comutações das lâmpadas dos semáforos, e pode ser visto com dois temporizadores (Timer). Um temporizador indica o funcionamento normal dos semáforos, e ou outro, a alteração dos tempos com a utilização da "chave eletrônica".

A saída do microcontrolador principal segue por um acoplador óptico de modo a isolar o CI do circuito, eliminando assim ruídos inconvenientes e por conseqüência os erros.

O circuito de potência é responsável por acionar as lâmpadas em corrente alternada, visto que a tensão até então era em corrente contínua (5VDC).

3.2. CONFIGURAÇÃO DO PROJETO

Para entender o programa serão necessárias algumas informações como:

3.2.1. As portas

As portas utilizadas para saída/entrada e as respectivas lâmpadas são:

1. PTB0 – Verde (lado 1)
2. PTB1 – Amarelo (lado 1)
3. PTB2 – Vermelho (lado 1)
4. PTB3 – Verde (lado 2)
5. PTB4 – Amarelo (lado 2)
6. PTB5 – Vermelho (lado 2)
7. PTB6 – Entrada “Travessia Tranqüila” 1
8. PTB7 – Entrada “Travessia Tranqüila” 2

A configuração de I/O das portas são programadas através do comando DDRA e DDRB, que possuem 8 bits. Os bits setados em nível alto garantem a respectiva porta PTA e PTB a função de saída.

3.2.2. Registradores

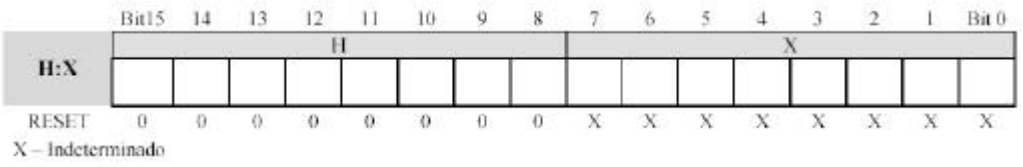
O acumulador (A) é um registrador de 8 bits de uso geral. A CPU utiliza o acumulador para armazenar os operandos e resultados de operações aritméticas e não aritméticas.

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

X - Indeterminado

O HX é um registrador de 16 bits de índice que 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.



3.3. PROGRAMANDO O MICROCONTROLADOR PRINCIPAL

O microcontrolador principal deve conter uma rotina que controle as comutações entre as lâmpadas: vermelha, amarela e verde. O programa desenvolvido segue abaixo, já com as considerações sobre cada comando Assembly, facilitando assim o entendimento do mesmo. Pelo mesmo motivo, foi feita uma divisão entre o programa principal e as subrotinas.

Como ponto de apoio, nos apêndices I e II pode-se verificar os modos de endereçamento e também um conjunto de instruções do microcontrolador em uso.

3.3.1. Programa Principal

```
---------------------------------------------------------------------------------------------------------------------
*****************************************************************
* Main_Init – Este é o ponto onde o código começa depois do RESET
*****************************************************************
Entry:
main:
    mov   #$00,PTB          ;Limpa PTB
    mov   #$3F,DDRB         ;Pinos de saída (os pinos 6 e 7 devem ser de entrada - 3F)
    bset  6,PTBPUE          ;Habilita pull-up PTB6
    bset  7,PTBPUE          ;Habilita pull-up PTB7
    mov   #$FF,DDRA         ;Habilita os pinos de PTA como saída

Ciclo:                      ;Marco - Ciclo
    bclr  4,PTB             ;Apaga luz amarela lado2
    bclr  2,PTB             ;Apaga luz vermelha lado1
    bset  0,PTB             ;Liga luz verde lado1
    bset  5,PTB             ;Liga luz vermelha lado2
    clrh                    ;Limpa o registrador h
    clrx                    ;Limpa o registrador x
    jsr   DelayV            ;Definir tempo de ligado/desligado (Pula para subrotina DelayV)
    cmp #$88                ;Compara A com 88
    beq Volta               ;Se for igual vai para 'Volta'
    bclr  0,PTB             ;Apaga luz verde lado1
    bset  1,PTB             ;Liga luz amarela lado1
    jsr   DelayA            ;Tempo do amarelo ligado/desligado (Pula para subrotina DelayA)
    bclr  1,PTB             ;Apaga luz amarela lado1
    bset  2,PTB         ;Liga luz vermelha lado1
    bclr  5,PTB         ;Apaga vermelha lado2
```

```
    bset  3,PTB                 ;Liga verde lado2
    jsr   DelayV                ; Definir tempo de ligado/desligado (Pula para subrotina DelayV)

Volta:                          ;Marco - Volta
    bclr  3,PTB                 ;Apaga luz verde lado2
    bset  4,PTB             ;Liga luz amarela lado2
    jsr   DelayA                ;Tempo do amarelo ligado/desligado (Pula para subrotina DelayA)
    bra  Ciclo                  ;Salta para Ciclo

END
```

Com este programa pode-se perceber que as os tempos de Vermelho/Verde são definidos na subrotina DelayV, enquanto que o tempo em Amarelo é definido na subrotina DelayA.

As funções BSET e BCLR têm a função de colocar em nível lógico alto ou baixo, respectivamente.

3.3.2. Subrotinas

```
*****************************************************************
* Subrotina de delayV
*****************************************************************
**** Neste caso, o verde/ vermelho fica ligado ~ 60s
DelayV:                     ;Subrotina DelayV
    lda   #$67              ;Carrega o acumulador com 67 em hexa
loop1:                      ;Abre loop1
    ldhx  #$FFFF            ;Carrega registrador hx com FFFF em hexa
loop2: sta $80              ;Guarda o valor do acumulador no endereço 80 da RAM
    lda PTB                 ;Carrega o acumulador com o conteúdo de PTB
    cmp #$A1                ;Compara o acumulador com A1 em hexa
    beq Delay1              ;Pula para Delay1 se for igual
    cmp #$61                ;Compara o acumulador com 61 em hexa
    beq Delay2              ;Pula para Delay2 se for igual
    lda $80                 ;Carrega o acumulador com o conteúdo do endereço 80 da RAM
    aix  #-1                ;Decrementa 1 do registrador hx
    cphx #0                 ;Compara registrador hx com 0
    bne  loop2              ;Pula para loop2 se não for igual
    dbnza loop1             ;Decrementa o acumulador e pula para loop1 se for diferente de 0
    rts                     ;Retorna da subrotina

*****************************************************************
* Subrotina de delayA
*****************************************************************
**** Luz amarela fica ligada por ~ 5s
DelayA:                     ;Subrotina DelayA
    lda   #$EE              ;Carrega o acumulador com EE em hexa
loop3:                      ;Abre loop3
     nop            ;Nenhuma operação
     nop            ;Nenhuma operação
     ldhx  #$3100   ;Carrega registrador hx com FFFF em hexa
loop4:                      ;Abre loop4
    aix  #-1                ;Decrementa 1 do registrador hx
    cphx #0                 ;Compara registrador hx com 0
    bne  loop4              ;Pula para loop4 se não for igual
```

```
    dbnza loop3             ;Decrementa o acumulador e pula para loop3 se for diferente de 0
    rts                     ;Retorna da subrotina

************************************************************
* Subrotina de delay1
************************************************************
**** Delay1 - Tempo do travessia tranquila (1) ~ 90s  ---  Pino 6 (PTB)
Delay1:
    bclr  0,PTB     ;Apaga luz verde lado1
    bset  1,PTB     ;Liga luz amarela lado1
    jsr   DelayA    ;Tempo do amarelo ligado/desligado
    bclr  1,PTB     ;Apaga luz amarela lado1
    bset  2,PTB     ;Liga luz vermelha lado1
    bclr  5,PTB     ;Apaga vermelha lado2
    bset  3,PTB     ;Liga verde lado2
    lda   #$FF              ;Carrega o acumulador com FF em hexa
loop7:                      ;Abre loop7
    ldhx  #$FFFF            ;Carrega registrador hx com FFFF em hexa
loop8:                      ;Abre loop8
    nop                     ;Nenhuma operação
    aix  #-1                ;Decrementa 1 do registrador hx
    cphx #0                 ;Compara registrador hx com 0
    bne  loop8              ;Pula para loop8 se não for igual
    dbnza loop7             ;Decrementa o acumulador e pula para loop7 se for diferente de 0
    lda   #$FF              ;Carrega o acumulador com FF em hexa
loop9:                      ;Abre loop9
    ldhx  #$FFFF            ;Carrega registrador hx com FFFF em hexa
loop10:                     ;Abre loop10
    aix  #-1                ;Decrementa 1 do registrador hx
    cphx #0                 ;Compara registrador hx com 0
    bne  loop10             ;Pula para loop10 se não for igual
    dbnza loop9             ;Decrementa o acumulador e pula para loop9 se for diferente de 0
    lda  #$88               ;Carrega o acumulador com 88 em hexa
    rts                     ;Retorna da subrotina

************************************************************
* Subrotina de delay2
************************************************************
**** Delay2 - Tempo do travessia tranquila (2) ~ 120s  ---  Pino 7 (PTB)

Delay2:
    bclr  0,PTB     ;Apaga luz verde lado1
    bset  1,PTB     ;Liga luz amarela lado1
    jsr   DelayA    ;Tempo do amarelo ligado/desligado
    bclr  1,PTB     ;Apaga luz amarela lado1
    bset  2,PTB     ;Liga luz vermelha lado1
    bclr  5,PTB     ;Apaga vermelha lado2
    bset  3,PTB             ;Liga verde lado2
    lda   #$FF              ;Carrega o acumulador com FF em hexa
loop11:                     ;Abre loop11
    ldhx  #$FFFF            ;Carrega registrador hx com FFFF em hexa
loop12:                     ;Abre loop12
    aix  #-1                ;Decrementa 1 do registrador hx
    cphx #0                 ;Compara registrador hx com 0
    nop                     ;Nenhuma operação
    nop                     ;Nenhuma operação
    nop                     ;Nenhuma operação
    bne  loop12             ;Pula para loop12 se não for igual
    dbnza loop11    ;Decrementa o acumulador e pula para loop11 se for diferente de 0
```

```
    lda   #$FF          ;Carrega o acumulador com FF em hexa
loop13:                 ;Abre loop13
    ldhx  #$FFFF        ;Carrega registrador hx com FFFF em hexa
loop14:                 ;Abre loop14
    aix  #-1            ;Decrementa 1 do registrador hx
    cphx #0             ;Compara registrador hx com 0
    nop                 ;Nenhuma operação
    nop                 ;Nenhuma operação
    nop                 ;Nenhuma operação
    bne  loop14         ;Pula para loop14 se não for igual
    dbnza loop13        ;Decrementa o acumulador e pula para loop13 se for diferente de 0
    lda  #$88           ;Carrega o acumulador com 88 em hexa
    rts                 ;Retorna da subrotina
```

No final de cada subrotina existe o comando RTS que faz retornar ao programa principal.

No caso deste projeto, a subrotina:

- ?? DelayA faz a lâmpada Amarela de ambos os lados permanecerem acesas durante cinco segudos;
- ?? DelayV faz as lâmpadas Vermelha (lado1) e Verde (lado2), e vice-versa, permanecerem um minuto acesas;
- ?? Delay1 é o primeiro acesso ao “Travessia Tranqüila”, garantindo noventa segundos para o pedestre atravessar; e
- ?? Delay2 é o segundo acesso ao “Travessia Tranqüila”, garantindo cento e vinte segundos para o pedestre atravessar.

Na subrotina DelayV existe uma leitura da porta PTB para verificar a existência do microcontrolador externo, podendo assim habilitar o “Travessia Tranquila” (Delay1 ou Delay2), dependendo da porta acessada pela “chave eletrônica”.

3.4. PROGRAMANDO O MICROCONTROLADOR EXTERNO

O microcontrolador externo estará de posse de uma pessoa com deficiência física, e poderá atuar em uma das duas portas preparadas no microcontrolador principal. O programa gravado neste CI pode ser visto abaixo.

3.4.1. Programa Principal

```
---------------------------------------------------------------------------------------------------------------------------------------------------
*****************************************************************
* Main_Init – Este é o ponto onde o código começa depois do RESET
*****************************************************************
Entry:
main:
      lda   #$00                    ;Carrega o acumulador com 00
      sta   PTA                     ;Grava 00 em PTA
      lda   #$FF                    ;Carrega FF no acumulador
      sta   DDRA                    ;Coloca todos os pinos PTA como saída
      lda   #$10                    ;Carrega 10 em PTA
      sta   PTA                     ;Coloca PTA4 em nível alto

Ciclo:                              ;Marco - Ciclo
      bset  1,PTA             ;Liga led vermelho (Ligado)
      jsr   Acionar                 ;Código para acessar o “Travessia Tranqüila” (Pula para subrotina Acionar)
      bset  5,PTA             ;Liga led verde (Acionou)
loop:  nop                          ;Nenhuma operação
       bra  loop                    ;Salta para loop

END
---------------------------------------------------------------------------------------------------------------------------------------------------
```

3.4.2. Subrotinas

```
**********Acionar**********
Acionar:
      ldhx  #$00FF            ;Carrega registrador hx com 00FF em hexa
      bclr  4,PTA                   ;Primeiro digito do código (Pino 4)

Atrasa1: aix  #-1                   ;Gasta #$FF com o primeiro dígito
      cphx #0                       ;Compara registrador hx com 0
      bne  Atrasa1            ;Pula para Atrasa1 se não for igual
      ldhx  #$00FF            ;Carrega registrador hx com 00FF em hexa
      bset  4,PTA                   ;Segundo digito do código (Pino 4)


Atrasa2: aix  #-1                   ;Gasta #$FF com o segundo digito
      cphx #0                       ;Compara registrador hx com 0
      bne  Atrasa2            ;Pula para Atrasa2 se não for igual
      ldhx  #$00FF            ;Carrega registrador hx com 00FF em hexa
      bclr  4,PTA                   ;Terceiro digito do código (Pino 4)
      rts                           ;Retorna da subrotina
---------------------------------------------------------------------------------------------------------------------------------------------------
```

3.5. ENTENDENDO OS PROGRAMAS

O programa principal em seu início habilita os pinos PTB0 ao PTB5 como saídas. Estes são os que fazem o acionamento dos transistores, e conseqüentemente das lâmpadas, através dos relés.

Os pinos PTB6 e PTB7 estão configurados como entrada, permitindo através deles, o acesso à rotina do "travessia tranqüila", garantindo um maior tempo para o pedestre atravessar a rua. Estas duas portas estão configuradas com "pull-up", ou seja, têm um nível de tensão fixo, evitando o acionamento através de ruídos, que é indesejável.

O código fonte do microcontrolador principal aciona a luz verde do lado 1 e a vermelha do lado 2 antes de seguir pela subrotina delayV. A subrotina delayV gasta um tempo de 1 minuto para retornar ao programa principal, sendo este então, o tempo de permanência dessas lâmpadas acesas.

Ao retornar da subrotina, a lâmpada verde é apagada e a amarela acesa. Segue-se então para a subrotina DelayA, onde se gasta cerca de 5 segundos para retornar. Logo após isso, a lâmpada amarela é apagada e a vermelha acesa, acionando também a lâmpada verde do lado 2.

A rotina é semelhante para os dois semáforos, o que também acontece com as subrotinas, pois os tempos são idênticos para as lâmpadas vermelha, amarela e verde.

Na subrotina DelayV existe uma leitura da porta PTB, que faz uma varredura buscando verificar a presença da chave eletrônica. Esta leitura ocorre através do comando lda PTB, sendo capaz de perceber em qual pino de entrada o "travessia tranqüila" foi acessado, no PTB6 ou no PTB7. Se for pelo PTB6 o programa segue pela subrotina Delay1, e se for pelo PTB7 o programa segue pela subrotina Delay2.

As subrotinas Delay1 e Delay2 gastam um tempo de 90 e 120 segundos, respectivamente. Estes são os tempos programados para a travessia dos pedestres com deficiência física.

O programa do microcontrolador externo, o da chave eletrônica, ao acessar o poste com o circuito principal, aciona um led indicando que foi alimentado. Logo após, segue para a subrotina "Acionar", que envia níveis alternados de tensão, alto e baixo, para o microcontrolador principal. Isto é o necessário para acionar a função "travessia tranqüila". Terminada a subrotina, retorna ao programa principal e aciona o outro led, indicando assim o fim da rotina e a permissão de retirada da chave eletrônica.

3.6. GRAVAÇÃO

Após o programa ser desenvolvido pode-se gravar o microcontrolador com a utilização da placa de gravação. O microcontrolador requer alguns sinais em seus pinos para que seja gravado, sendo possível fazer a gravação sem a utilização da placa oficial da Motorola. Para isso, necessita-se de um circuito montado com algumas características específicas.

Segue abaixo o circuito necessário para gravação do microcontrolador, incluindo o CI MAX232, responsável pela conversão de níveis entre PC e Microcontrolador.

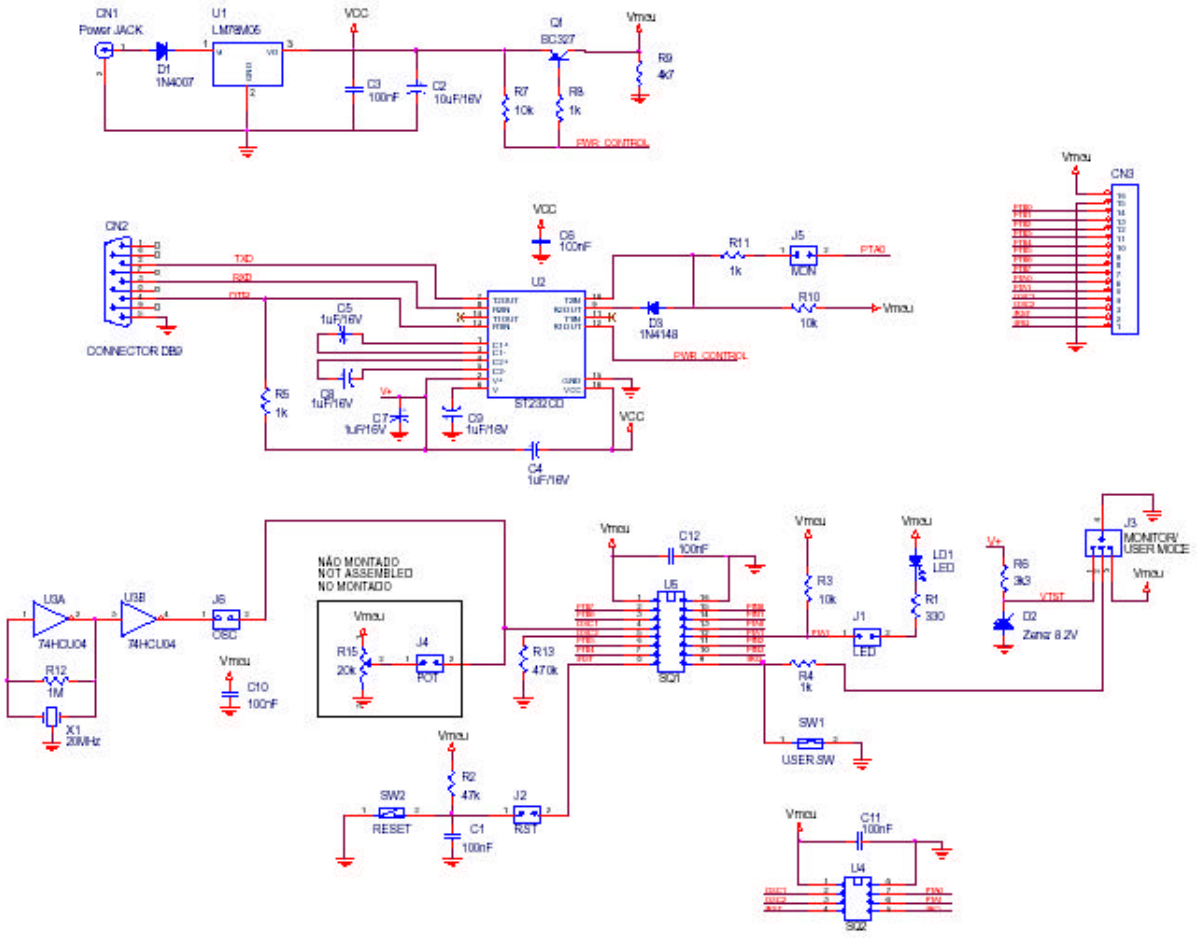


Figura 8. Placa de gravação da família MC68HC908QT/QY
Fonte: Manual do Kit de Programação

Abaixo está a configuração da comunicação entre o PC e a placa em que se encontra o microcontrolador, permitindo a configuração da porta serial do micro e a velocidade da mesma.

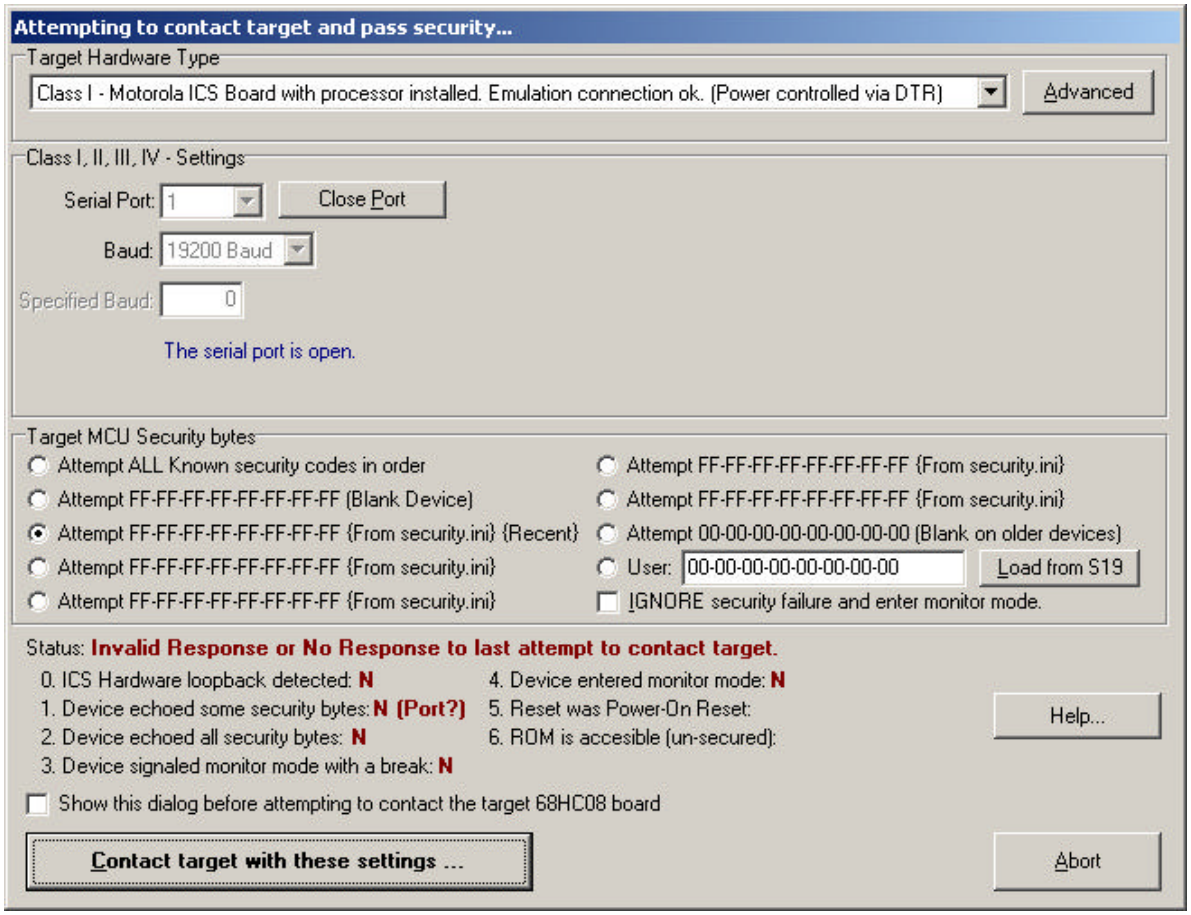


Figura 9. Configuração de comunicação (PC – Placa)

Fonte: Software CodeWarrior

3.7. SIMULAÇÃO

Com a utilização da placa de gravação foram realizados dois testes:

1. **Passo-a-passo**: Pulando linha a linha de comando e verificando o que ocorreu.
2. **Run to cursor**: Soltando o programa para rodar até a instrução desejada, verificando o tempo da rotina.

3.7.1. Passo-a-passo

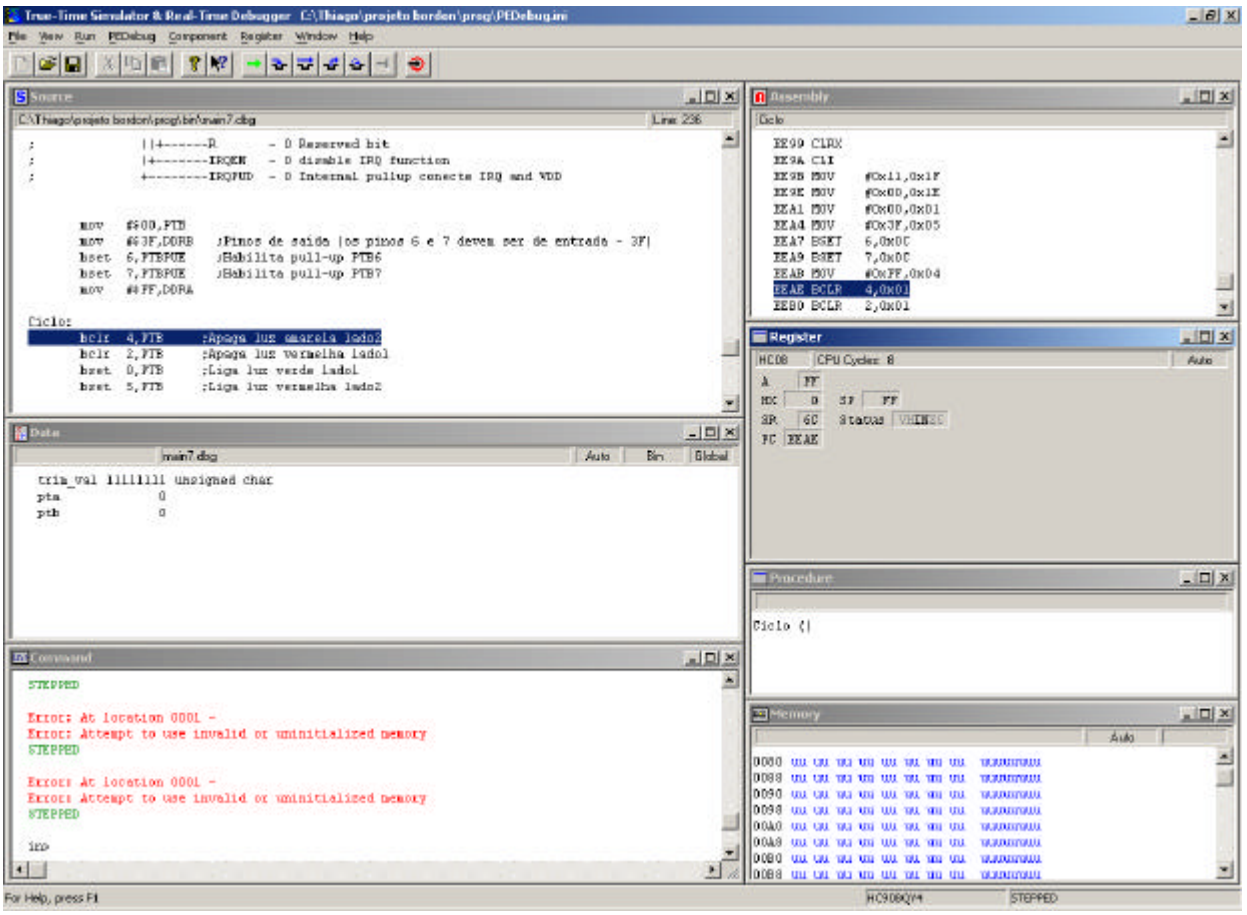


Figura 10. Simulação passo-a-passo

Fonte: Software CodeWarrior

Este modo de simulação é eficiente para a detecção de erros, onde se verificam todos os registradores e portas após cada comando da rotina.

3.7.2. Run to cursor

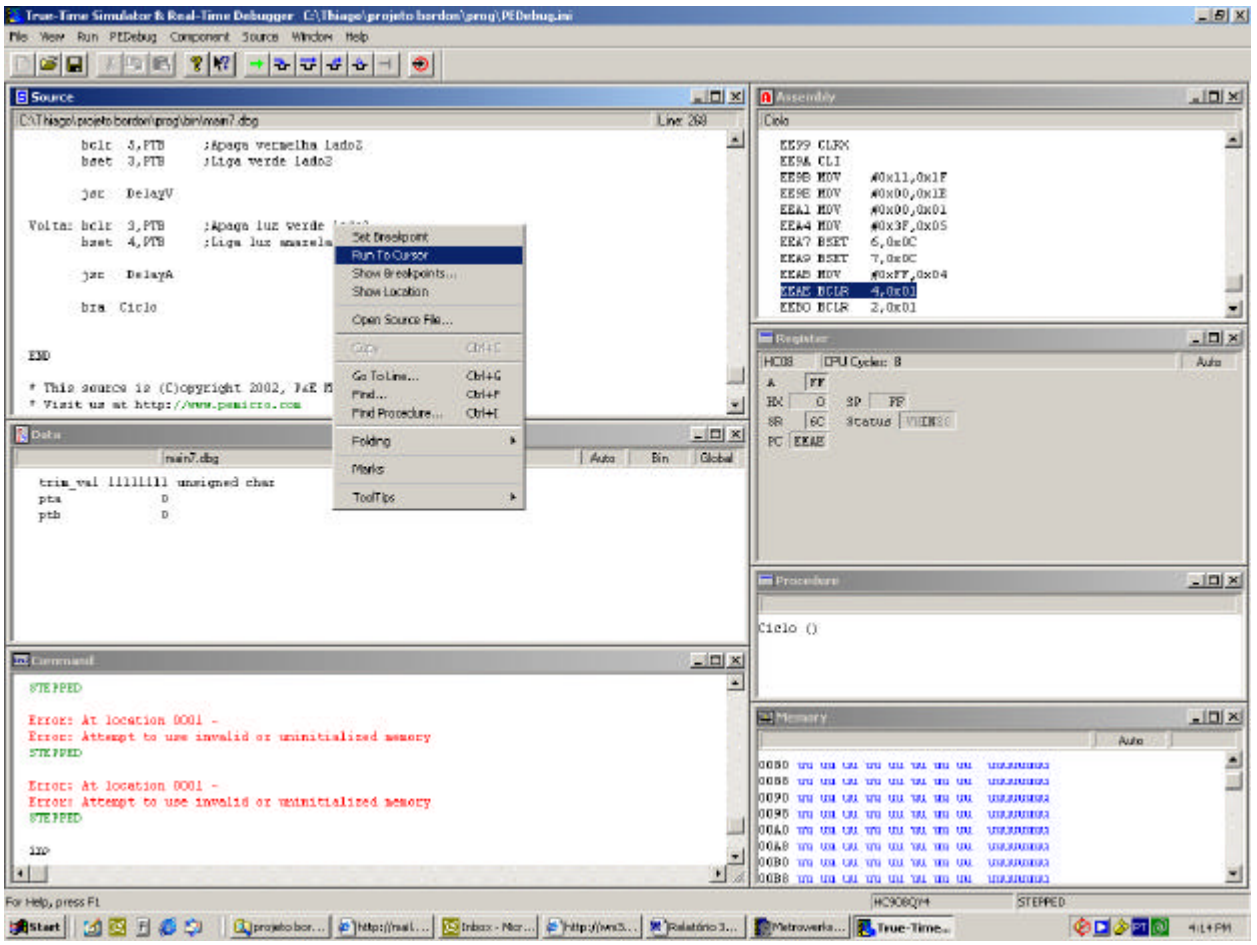


Figura 11. Simulação “run to cursor”

Fonte: Software CodeWarrior

Este modo de simulação é muito útil para se testar uma determinada parte da rotina, sem precisar passar por outras partes passo-a-passo, sabendo que já estão funcionando corretamente.

3.8. TESTES

Com a utilização de uma proto-board, foi montado o microcontrolador para acionar os leds.

Foi verificada a eficácia da programação, permitindo a avaliação do sinal a ser utilizado na parte de acionamento das lâmpadas na saída.

3.9. ACIONAMENTO DAS SAÍDAS

Dando continuidade ao projeto, desenvolveu-se a parte de circuito de potência/acionamento para a saída.

Ao verificar com alguns testes, o microcontrolador utilizado respondeu corretamente sem a utilização de um acoplador óptico, pensado inicialmente para evitar ruídos oriundos da saída do sistema. Então o sistema agora irá ser acoplado simplesmente com o uso de um transistor, operando como “chave”.

As saídas utilizadas do microcontrolador são denominadas PTB0 até PTB5 (como visto anteriormente em “Configuração do Projeto”), constituindo o acionamento de cada uma das lâmpadas dos semáforos.

Cada uma dessas saídas deve fazer o chaveamento de um relé, para assim utilizar-se de corrente alternada no acionamento das lâmpadas. Este acionamento requer um transistor operando como “chave”, fornecendo assim os 12V necessários para o funcionamento do relé.

Para verificar o funcionamento, abaixo se vê uma única saída do microcontrolador acionando o relê. Neste caso, quando a PTB estiver acionada o transistor irá opera como uma chave fechada, aparecendo os 12V em cima da bobina do relé. Assim, o contato NA (normal aberto) do relé é conectado ao comum, acionando a carga em AC, que no caso será uma lâmpada.

Tem-se com este circuito, todo o acionameno em DC, mas uma saída em AC, o que é muito interessante, pois a parte lógica do sistema não dificulta em nada o seu manuseio, manutenção e testes.

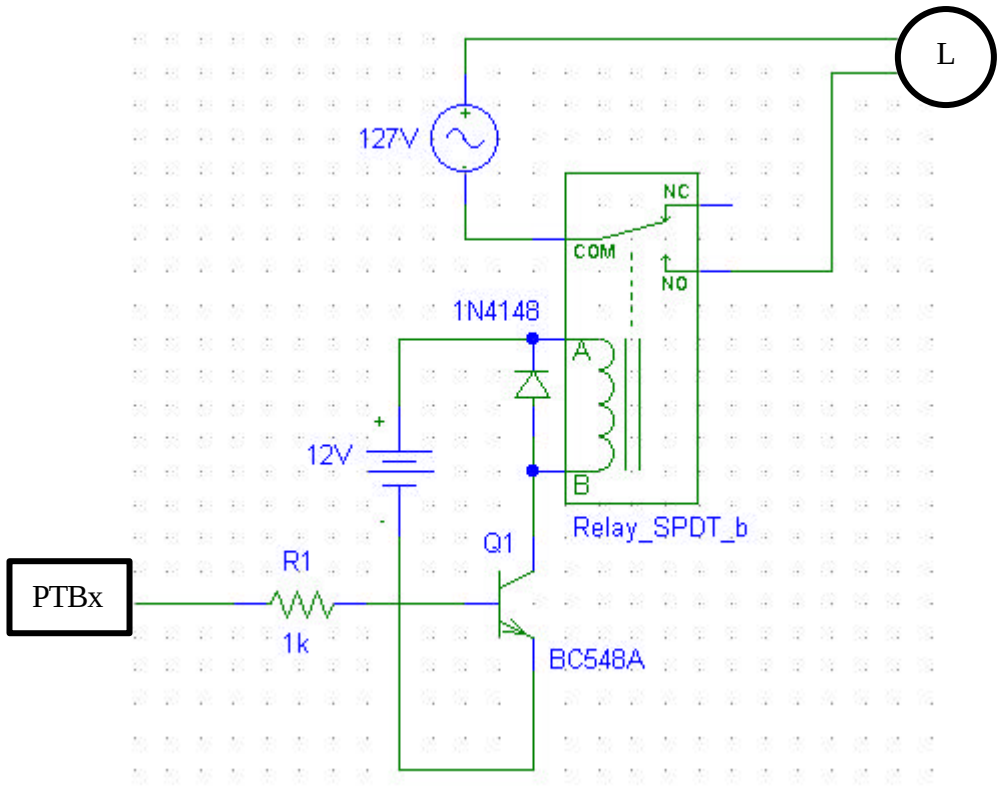


Figura 12. Configuração de uma saída

Um dos problemas encontrados foi justamente o acoplamento entre o microcontrolador e o transistor, que a princípio estava chaveando corretamente, acionando led na saída. Com a inserção do relé, a resposta já não era a mesma. Não estava conseguindo desenergizar o primeiro relé e acionar o segundo. Foi então colocado um diodo entre os terminais da bobina do relé, para assim evitar a FEM (força contra-eletromotriz) na bobina, permitindo que se desenergize corretamente.

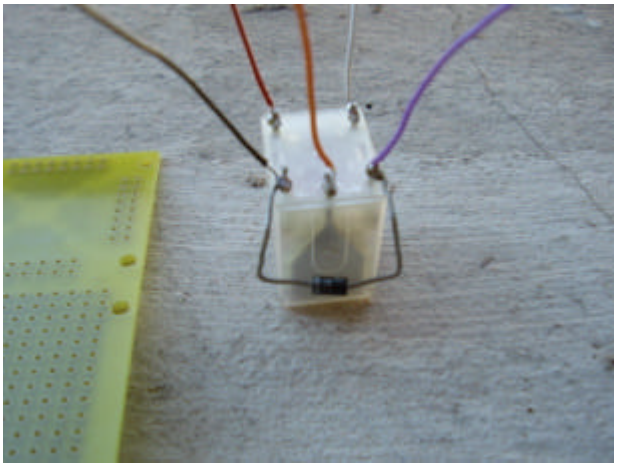


Figura 13. Apresentação de um relé

Com todas as saídas configuradas da mesma forma, pode-se então definir a placa de circuito impresso, que é mostrada abaixo:

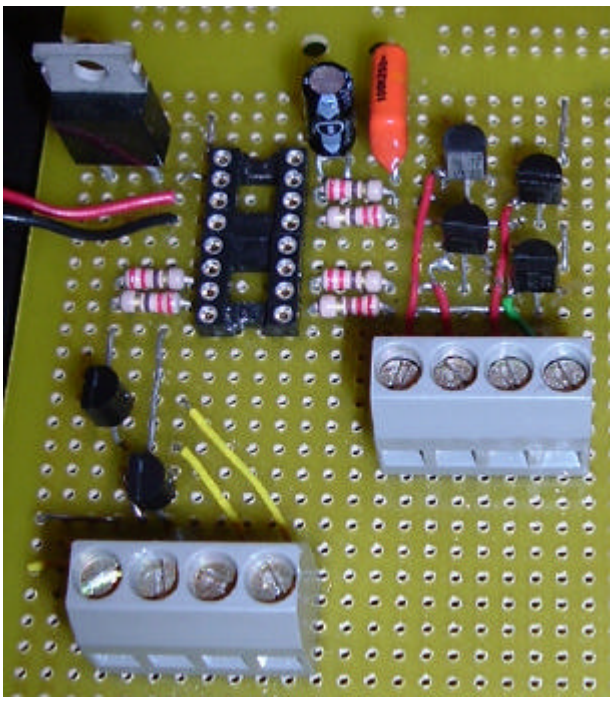
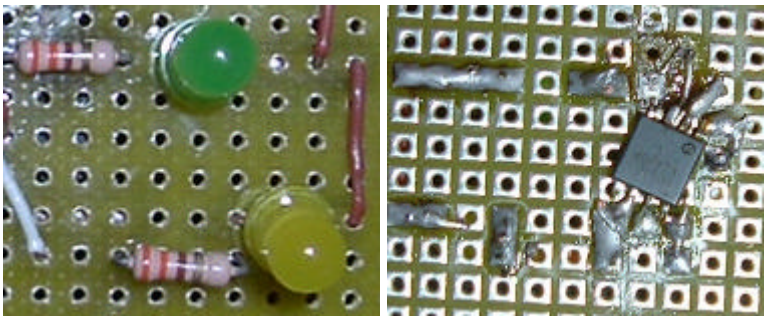


Figura 14. Apresentação da placa com o circuito principal concluído

A placa da chave eletrônica pode ser vista aqui:



(a) (b)

Figura 15. Circuito da chave eletrônica: a) Vista superior; b) Vista inferior

3.10. A MAQUETE

Para a verificação final do projeto foi criada uma maquete com os semáforos, que pode ser vista logo abaixo.

Figura 16. Maquete com os semáforos

4. CONSIDERAÇÕES FINAIS

Com grande importância no mercado atual aparecem os microcontroladores, sendo responsáveis por tarefas que até passam despercebidas, como é o caso do controle de semáforos. O estudo e a realização deste projeto foi muito importante para a revisão desta tecnologia, que tem muito a ser explorada ainda.

Com a tentativa de solucionar o problema dos deficientes físicos no trânsito, pode-se ver o quanto os microcontroladores são úteis, e podem ser empregados em muitas situações ainda carentes tecnologicamente.

Espera-se que este documento seja de real auxílio para outros que necessitem de informações sobre o emprego desse tipo de microcontrolador.

REFERÊNCIAS BIBLIOGRÁFICAS

1. Livro: MICROCONTROLADORES HC908Q: TEORIA E PRÁTICA
   Autor: FÁBIO PEREIRA
   Editora: EDITORA ÉRICA
   Número de páginas: 296

2. DATA SHEET M68HC08 – MICROCONTROLLERS
   Número de páginas: 240
   Ano: 2004

3. MANUAL DO USUÁRIO – PLACA DE DESENVOLVIMENTO M68EVB908Q
   Kit de Programação MOTOROLA M68HC08
   Número de páginas: 15

4. MC68HC908QY4/D MC68HC908QY4 ADVANCED INFORMATION
   http://www.motorola.com/sps/

5. CODEWARRIOR DEVELOPMENT STUDIO FOR HC08 MICROCONTROLLERS SPECIAL EDITION
   http://www.metrowerks.com/MW/Develop/Embedded/HC08/Default.htm

6. AN2305 USER MODE MONITOR ACCESS FOR MC68HC908QY/QT SERIES MCUS
   http://www.motorola.com/sps/

7. APOSTILA HC08 - CNZ
   http://www.eel.ufsc.br/hari/Apostila_HC08_-_CNZ_(Rev_1).pdf
   Páginas: 96
   Pesquisado em: 08/2004

GLOSSÁRIO

Microcontrolador	É um microprocessador e vários periféricos num único componente eletrônico.
Pino de Entrada/Saída (I/O)	Pino de ligação externa do microcontrolador, que pode ser configurado como entrada ou saída. Na maioria dos casos, o pino de entrada e saída permite ao microcontrolador comunicar, controlar ou ler informação.
Software	Informação de que o microcontrolador necessita, para poder funcionar. O software não pode apresentar quaisquer erros se desejar que o programa e o dispositivo funcionem como deve ser. O software pode ser escrito em diversas linguagens tais como: Basic, C, Pascal ou assembler.
Hardware	Microcontrolador, memória, alimentação, circuitos de sinal e todos os componentes ligados ao microcontrolador.
Simulador	Pacote de software para "correr" num PC que simula o funcionamento interno do microcontrolador. É um instrumento ideal para verificar as rotinas de software e todas as porções de código que não implicam ligação com o mundo exterior. Existem opções para observar o código quando se desloca no programa para trás e para a frente ou passo-a-passo, e para detecção de erros.
Assembler	Pacote de software que traduz código fonte em código que o microcontrolador pode compreender. Uma parte deste software destina-se também à detecção dos erros cometidos ao escrever o programa.
ROM, EPROM, EEPROM, FLASH, RAM	São tipos de memórias encontradas quando utiliza-se o microcontrolador. A primeira não pode ser limpa, aquilo que se escreve nela, permanece para sempre e nunca mais pode ser apagada. A segunda pode apagar-se por meio de uma lâmpada de raios ultravioletas. A terceira, pode ser apagada eletricamente usando a tensão que o microcontrolador funciona. A quarta, também é apagável eletricamente, mas ao contrário da memória EEPROM, não implica um grande número de ciclos, ao escrever ou apagar os conteúdos dos endereços de memória. Finalmente, o último tipo, é a memória mais rápida, mas não conserva o seu conteúdo quando ocorre uma falha na alimentação. Por isso, esta memória não é usada para guardar o programa, mas sim para guardar os valores das variáveis e resultados intermédios.
Endereçamento	Determina e designa o local da memória a aceder.

Byte, Kilobyte, Megabyte	Termos relacionados com quantidades de informação. A unidade básica é o byte que corresponde a 8 bits. Um kilobyte são 1024 bytes e um megabyte tem 1024 kilobytes.
OPCODE	OPerativo CODE (do inglês código operativo) é o valor numérico que identifica univocalmante uma instrução do microcontrolador e de seu parâmetro de funcionamento.
Subrotina	É um fraguimento de instrução chamado mais de uma vez dentro um programa. Tipicamente uma subrotina será colocada em um programa seja para separar logicamente a função do resto do código, ou seja, para economizar espaço de memória, escrevendo-se uma só vez o grupo de instruções necessária.

ANEXO I – 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.

1. Inerente

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

Exemplo: DECA - Decrementa conteúdo do acumulador.

2. lmediato

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 - Carrega o acumulador com o valor $20.

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.

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.

5. Indexado

Os modos de endereçamento indexado são a chave para a geração de código eficiente para pesquisa de 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+,TGI - Compara o conteúdo de A com o conteúdo do endereço armazenado em (H:X + $50), salta para TAG quando igual, e posteriormente, incrementa X.

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 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).

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 urna 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.

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.

ANEXO II – CONJUNTO DE INSTRUÇÕES

Segue uma tabela resumo do conjunto de instruções da família M68HC08.

Tabela 3. Conjunto de Instruções (Folha 1 de 7)

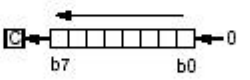
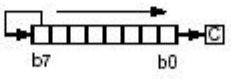
Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
ADC #*opr* ADC *opr* ADC *opr* ADC *opr*,X ADC *opr*,X ADC ,X ADC *opr*,SP ADC *opr*,SP	Add with Carry	A ← (A) + (M) + (C)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A9 B9 C9 D9 E9 F9 9EE9 9ED9	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ADD #*opr* ADD *opr* ADD *opr* ADD *opr*,X ADD *opr*,X ADD ,X ADD *opr*,SP ADD *opr*,SP	Add without Carry	A ← (A) + (M)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	AB BB CB DB EB FB 9EEB 9EDB	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
AIS #*opr*	Add Immediate Value (Signed) to SP	SP ← (SP) + (16 « M)	–	–	–	–	–	–	IMM	A7	ii	2
AIX #*opr*	Add Immediate Value (Signed) to H:X	H:X ← (H:X) + (16 « M)	–	–	–	–	–	–	IMM	AF	ii	2
AND #*opr* AND *opr* AND *opr* AND *opr*,X AND *opr*,X AND ,X AND *opr*,SP AND *opr*,SP	Logical AND	A ← (A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A4 B4 C4 D4 E4 F4 9EE4 9ED4	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ASL *opr* ASLA ASLX ASL *opr*,X ASL ,X ASL *opr*,SP	Arithmetic Shift Left (Same as LSL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
ASR *opr* ASRA ASRX ASR *opr*,X ASR *opr*,X ASR *opr*,SP	Arithmetic Shift Right		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5
BCC *rel*	Branch if Carry Bit Clear	PC ← (PC) + 2 + rel ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3
BCLR *n*, *opr*	Clear Bit n in M	Mn ← 0	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BCS *rel*	Branch if Carry Bit Set (Same as BLO)	PC ← (PC) + 2 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BEQ *rel*	Branch if Equal	PC ← (PC) + 2 + *rel* ? (Z) = 1	–	–	–	–	–	–	REL	27	rr	3

Tabela 3. Conjunto de Instruções (Folha 2 de 7)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BGE *opr*	Branch if Greater Than or Equal To (Signed Operands)	PC ← (PC) + 2 + *rel* ? (N ⊕ V) = 0	–	–	–	–	–	–	REL	90	rr	3
BGT *opr*	Branch if Greater Than (Signed Operands)	PC ← (PC) + 2 + *rel* ? (Z) | (N ⊕ V) = 0	–	–	–	–	–	–	REL	92	rr	3
BHCC *rel*	Branch if Half Carry Bit Clear	PC ← (PC) + 2 + *rel* ? (H) = 0	–	–	–	–	–	–	REL	28	rr	3
BHCS *rel*	Branch if Half Carry Bit Set	PC ← (PC) + 2 + *rel* ? (H) = 1	–	–	–	–	–	–	REL	29	rr	3
BHI *rel*	Branch if Higher	PC ← (PC) + 2 + *rel* ? (C) | (Z) = 0	–	–	–	–	–	–	REL	22	rr	3
BHS *rel*	Branch if Higher or Same (Same as BCC)	PC ← (PC) + 2 + *rel* ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3
BIH *rel*	Branch if $\overline{IRQ}$ Pin High	PC ← (PC) + 2 + *rel* ? $\overline{IRQ}$ = 1	–	–	–	–	–	–	REL	2F	rr	3
BIL *rel*	Branch if $\overline{IRQ}$ Pin Low	PC ← (PC) + 2 + *rel* ? $\overline{IRQ}$ = 0	–	–	–	–	–	–	REL	2E	rr	3
BIT #*opr* BIT *opr* BIT *opr* BIT *opr*,X BIT *opr*,X BIT ,X BIT *opr*,SP BIT *opr*,SP	Bit Test	(A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A5 B5 C5 D5 E5 F5 9EE5 9ED5	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
BLE *opr*	Branch if Less Than or Equal To (Signed Operands)	PC ← (PC) + 2 + *rel* ? (Z) | (N ⊕ V) = 1	–	–	–	–	–	–	REL	93	rr	3
BLO *rel*	Branch if Lower (Same as BCS)	PC ← (PC) + 2 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BLS *rel*	Branch if Lower or Same	PC ← (PC) + 2 + *rel* ? (C) | (Z) = 1	–	–	–	–	–	–	REL	23	rr	3
BLT *opr*	Branch if Less Than (Signed Operands)	PC ← (PC) + 2 + *rel* ? (N ⊕ V) =1	–	–	–	–	–	–	REL	91	rr	3
BMC *rel*	Branch if Interrupt Mask Clear	PC ← (PC) + 2 + *rel* ? (I) = 0	–	–	–	–	–	–	REL	2C	rr	3
BMI *rel*	Branch if Minus	PC ← (PC) + 2 + *rel* ? (N) = 1	–	–	–	–	–	–	REL	2B	rr	3
BMS *rel*	Branch if Interrupt Mask Set	PC ← (PC) + 2 + *rel* ? (I) = 1	–	–	–	–	–	–	REL	2D	rr	3
BNE *rel*	Branch if Not Equal	PC ← (PC) + 2 + *rel* ? (Z) = 0	–	–	–	–	–	–	REL	26	rr	3
BPL *rel*	Branch if Plus	PC ← (PC) + 2 + *rel* ? (N) = 0	–	–	–	–	–	–	REL	2A	rr	3
BRA *rel*	Branch Always	PC ← (PC) + 2 + *rel*	–	–	–	–	–	–	REL	20	rr	3
BRCLR *n*,*opr*,*rel*	Branch if Bit *n* in M Clear	PC ← (PC) + 3 + *rel* ? (Mn) = 0	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN *rel*	Branch Never	PC ← (PC) + 2	–	–	–	–	–	–	REL	21	rr	3

Tabela 3. Conjunto de Instruções (Folha 3 de 7)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BRSET *n,opr,rel*	Branch if Bit *n* in M Set	PC ← (PC) + 3 + *rel* ? (Mn) = 1	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET *n,opr*	Set Bit *n* in M	Mn ← 1	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BSR *rel*	Branch to Subroutine	PC ← (PC) + 2; push (PCL) SP ← (SP) – 1; push (PCH) SP ← (SP) – 1 PC ← (PC) + *rel*	–	–	–	–	–	–	REL	AD	rr	4
CBEQ *opr,rel* CBEQA #*opr,rel* CBEQX #*opr,rel* CBEQ *opr*,X+,*rel* CBEQ X+,*rel* CBEQ *opr*,SP,*rel*	Compare and Branch if Equal	PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 3 + rel ? (X) – (M) = $00 PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 2 + rel ? (A) – (M) = $00 PC ← (PC) + 4 + rel ? (A) – (M) = $00	–	–	–	–	–	–	DIR IMM IMM IX1+ IX+ SP1	31 41 51 61 71 9E61	dd rr ii rr ii rr ff rr rr ff rr	5 4 4 5 4 6
CLC	Clear Carry Bit	C ← 0	–	–	–	–	–	0	INH	98		1
CLI	Clear Interrupt Mask	I ← 0	–	–	0	–	–	–	INH	9A		2
CLR *opr* CLRA CLRX CLRH CLR *opr*,X CLR ,X CLR *opr*,SP	Clear	M ← $00 A ← $00 X ← $00 H ← $00 M ← $00 M ← $00 M ← $00	0	–	–	0	1	–	DIR INH INH INH IX1 IX SP1	3F 4F 5F 8C 6F 7F 9E6F	dd ff ff	3 1 1 1 3 2 4
CMP #*opr* CMP *opr* CMP *opr* CMP *opr*,X CMP *opr*,X CMP ,X CMP *opr*,SP CMP *opr*,SP	Compare A with M	(A) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A1 B1 C1 D1 E1 F1 9EE1 9ED1	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
COM *opr* COMA COMX COM *opr*,X COM ,X COM *opr*,SP	Complement (One's Complement)	M ← (M) = $FF – (M) A ← (A) = $FF – (M) X ← (X) = $FF – (M) M ← (M) = $FF – (M) M ← (M) = $FF – (M) M ← (M) = $FF – (M)	0	–	–	↕	↕	1	DIR INH INH IX1 IX SP1	33 43 53 63 73 9E63	dd ff ff	4 1 1 4 3 5
CPHX #*opr* CPHX *opr*	Compare H:X with M	(H:X) – (M:M + 1)	↕	–	–	↕	↕	↕	IMM DIR	65 75	ii ii+1 dd	3 4

Tabela 3. Conjunto de Instruções (Folha 4 de 7)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
CPX #*opr* CPX *opr* CPX *opr* CPX ,X CPX *opr*,X CPX *opr*,X CPX *opr*,SP CPX *opr*,SP	Compare X with M	(X) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A3 B3 C3 D3 E3 F3 9EE3 9ED3	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
DAA	Decimal Adjust A	$(A)_{10}$	U	–	–	↕	↕	↕	INH	72		2
DBNZ *opr,rel* DBNZA *rel* DBNZX *rel* DBNZ *opr*,X,*rel* DBNZ X,*rel* DBNZ *opr*,SP,*rel*	Decrement and Branch if Not Zero	A ← (A) – 1 or M ← (M) – 1 or X ← (X) – 1 PC ← (PC) + 3 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 3 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 4 + *rel* ? (result) ≠ 0	–	–	–	–	–	–	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC *opr* DECA DECX DEC *opr*,X DEC ,X DEC *opr*,SP	Decrement	M ← (M) – 1 A ← (A) – 1 X ← (X) – 1 M ← (M) – 1 M ← (M) – 1 M ← (M) – 1	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5
DIV	Divide	A ← (H:A)/(X) H ← Remainder	–	–	–	–	↕	↕	INH	52		7
EOR #*opr* EOR *opr* EOR *opr* EOR *opr*,X EOR *opr*,X EOR ,X EOR *opr*,SP EOR *opr*,SP	Exclusive OR M with A	A ← (A ⊕ M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A8 B8 C8 D8 E8 F8 9EE8 9ED8	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
INC *opr* INCA INCX INC *opr*,X INC ,X INC *opr*,SP	Increment	M ← (M) + 1 A ← (A) + 1 X ← (X) + 1 M ← (M) + 1 M ← (M) + 1 M ← (M) + 1	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3C 4C 5C 6C 7C 9E6C	dd ff ff	4 1 1 4 3 5
JMP *opr* JMP *opr* JMP *opr*,X JMP *opr*,X JMP ,X	Jump	PC ← Jump Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff 	2 3 4 3 2
JSR *opr* JSR *opr* JSR *opr*,X JSR *opr*,X JSR ,X	Jump to Subroutine	PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) – 1 Push (PCH); SP ← (SP) – 1 PC ← Unconditional Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff 	4 5 6 5 4
LDA #*opr* LDA *opr* LDA *opr* LDA *opr*,X LDA *opr*,X LDA ,X LDA *opr*,SP LDA *opr*,SP	Load A from M	A ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A6 B6 C6 D6 E6 F6 9EE6 9ED6	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Tabela 3. Conjunto de Instruções (Folha 5 de 7)

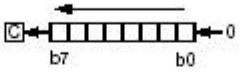
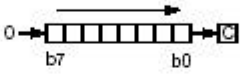
Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
LDHX #*opr* LDHX *opr*	Load H:X from M	H:X ← (M:M + 1)	0	–	–	↕	↕	–	IMM DIR	45 55	ii jj dd	3 4
LDX #*opr* LDX *opr* LDX *opr* LDX *opr*,X LDX *opr*,X LDX ,X LDX *opr*,SP LDX *opr*,SP	Load X from M	X ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AE BE CE DE EE FE 9EEE 9EDE	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
LSL *opr* LSLA LSLX LSL *opr*,X LSL ,X LSL *opr*,SP	Logical Shift Left (Same as ASL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
LSR *opr* LSRA LSR*X* LSR *opr*,X LSR ,X LSR *opr*,SP	Logical Shift Right		↕	–	–	0	↕	↕	DIR INH INH IX1 IX SP1	34 44 54 64 74 9E64	dd ff ff	4 1 1 4 3 5
MOV *opr*,*opr* MOV *opr*,X+ MOV #*opr*,*opr* MOV X+,*opr*	Move	$(M)_{Destination} \leftarrow (M)_{Source}$ H:X ← (H:X) + 1 (IX+D, DIX+)	0	–	–	↕	↕	–	DD DIX+ IMD IX+D	4E 5E 6E 7E	dd dd dd ii dd dd	5 4 4 4
MUL	Unsigned multiply	X:A ← (X) × (A)	–	0	–	–	–	0	INH	42		5
NEG *opr* NEGA NEGX NEG *opr*,X NEG ,X NEG *opr*,SP	Negate (Two's Complement)	M ← –(M) = $00 – (M) A ← –(A) = $00 – (A) X ← –(X) = $00 – (X) M ← –(M) = $00 – (M) M ← –(M) = $00 – (M)	↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	30 40 50 60 70 9E60	dd ff ff	4 1 1 4 3 5
NOP	No Operation	None	–	–	–	–	–	–	INH	9D		1
NSA	Nibble Swap A	A ← (A[3:0]:A[7:4])	–	–	–	–	–	–	INH	62		3
ORA #*opr* ORA *opr* ORA *opr* ORA *opr*,X ORA *opr*,X ORA ,X ORA *opr*,SP ORA *opr*,SP	Inclusive OR A and M	A ← (A) | (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AA BA CA DA EA FA 9EEA 9EDA	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
PSHA	Push A onto Stack	Push (A); SP ← (SP) – 1	–	–	–	–	–	–	INH	87		2
PSHH	Push H onto Stack	Push (H); SP ← (SP) – 1	–	–	–	–	–	–	INH	8B		2
PSHX	Push X onto Stack	Push (X); SP ← (SP) – 1	–	–	–	–	–	–	INH	89		2
PULA	Pull A from Stack	SP ← (SP + 1); Pull (A)	–	–	–	–	–	–	INH	86		2
PULH	Pull H from Stack	SP ← (SP + 1); Pull (H)	–	–	–	–	–	–	INH	8A		2
PULX	Pull X from Stack	SP ← (SP + 1); Pull (X)	–	–	–	–	–	–	INH	88		2

Tabela 3. Conjunto de Instruções (Folha 6 de 7)

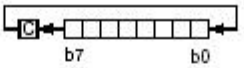
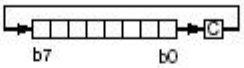
Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
ROL *opr* ROLA ROLX ROL *opr*,X ROL ,X ROL *opr*,SP	Rotate Left through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	39 49 59 69 79 9E69	dd ff ff	4 1 1 4 3 5
ROR *opr* RORA RORX ROR *opr*,X ROR ,X ROR *opr*,SP	Rotate Right through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	36 46 56 66 76 9E66	dd ff ff	4 1 1 4 3 5
RSP	Reset Stack Pointer	SP ← $FF	–	–	–	–	–	–	INH	9C		1
RTI	Return from Interrupt	SP ← (SP) + 1; Pull (CCR) SP ← (SP) + 1; Pull (A) SP ← (SP) + 1; Pull (X) SP ← (SP) + 1; Pull (PCH) SP ← (SP) + 1; Pull (PCL)	↕	↕	↕	↕	↕	↕	INH	80		7
RTS	Return from Subroutine	SP ← SP + 1; Pull (PCH) SP ← SP + 1; Pull (PCL)	–	–	–	–	–	–	INH	81		4
SBC #*opr* SBC *opr* SBC *opr* SBC *opr*,X SBC *opr*,X SBC ,X SBC *opr*,SP SBC *opr*,SP	Subtract with Carry	A ← (A) – (M) – (C)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A2 B2 C2 D2 E2 F2 9EE2 9ED2	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
SEC	Set Carry Bit	C ← 1	–	–	–	–	–	1	INH	99		1
SEI	Set Interrupt Mask	I ← 1	–	–	1	–	–	–	INH	9B		2
STA *opr* STA *opr* STA *opr*,X STA *opr*,X STA ,X STA *opr*,SP STA *opr*,SP	Store A in M	M ← (A)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP1 SP2	B7 C7 D7 E7 F7 9EE7 9ED7	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
STHX *opr*	Store H:X in M	(M:M + 1) ← (H:X)	0	–	–	↕	↕	–	DIR	35	dd	4
STOP	Enable $\overline{IRQ}$ Pin; Stop Oscillator	I ← 0; Stop Oscillator	–	–	0	–	–	–	INH	8E		1
STX *opr* STX *opr* STX *opr*,X STX *opr*,X STX ,X STX *opr*,SP STX *opr*,SP	Store X in M	M ← (X)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP1 SP2	BF CF DF EF FF 9EEF 9EDF	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
SUB #*opr* SUB *opr* SUB *opr* SUB *opr*,X SUB *opr*,X SUB ,X SUB *opr*,SP SUB *opr*,SP	Subtract	A ← (A) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A0 B0 C0 D0 E0 F0 9EE0 9ED0	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Tabela 3. Conjunto de Instruções (Folha 7 de 7)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
SWI	Software Interrupt	PC ← (PC) + 1; Push (PCL) SP ← (SP) – 1; Push (PCH) SP ← (SP) – 1; Push (X) SP ← (SP) – 1; Push (A) SP ← (SP) – 1; Push (CCR) SP ← (SP) – 1; I ← 1 PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte	–	–	1	–	–	–	INH	83		9
TAP	Transfer A to CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	INH	84		2
TAX	Transfer A to X	X ← (A)	–	–	–	–	–	–	INH	97		1
TPA	Transfer CCR to A	A ← (CCR)	–	–	–	–	–	–	INH	85		1
TST *opr* TSTA TSTX TST *opr*,X TST ,X TST *opr*,SP	Test for Negative or Zero	(A) – $00 or (X) – $00 or (M) – $00	0	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4
TSX	Transfer SP to H:X	H:X ← (SP) + 1	–	–	–	–	–	–	INH	95		2
TXA	Transfer X to A	A ← (X)	–	–	–	–	–	–	INH	9F		1
TXS	Transfer H:X to SP	(SP) ← (H:X) – 1	–	–	–	–	–	–	INH	94		2

A	Accumulator
C	Carry/borrow bit
CCR	Condition code register
dd	Direct address of operand
dd rr	Direct address of operand and relative offset of branch instruction
DD	Direct to direct addressing mode
DIR	Direct addressing mode
DIX+	Direct to indexed with post increment addressing mode
ee ff	High and low bytes of offset in indexed, 16-bit offset addressing
EXT	Extended addressing mode
ff	Offset byte in indexed, 8-bit offset addressing
H	Half-carry bit
H	Index register high byte
hh ll	High and low bytes of operand address in extended addressing
I	Interrupt mask
ii	Immediate operand byte
IMD	Immediate source to direct destination addressing mode
IMM	Immediate addressing mode
INH	Inherent addressing mode
IX	Indexed, no offset addressing mode
IX+	Indexed, no offset, post increment addressing mode
IX+D	Indexed with post increment to direct addressing mode
IX1	Indexed, 8-bit offset addressing mode
IX1+	Indexed, 8-bit offset, post increment addressing mode
IX2	Indexed, 16-bit offset addressing mode
M	Memory location
N	Negative bit
n	Any bit
opr	Operand (one or two bytes)
PC	Program counter
PCH	Program counter high byte
PCL	Program counter low byte
REL	Relative addressing mode
rel	Relative program counter offset byte
rr	Relative program counter offset byte
SP1	Stack pointer, 8-bit offset addressing mode
SP2	Stack pointer 16-bit offset addressing mode
SP	Stack pointer
U	Undefined
V	Overflow bit
X	Index register low byte
Z	Zero bit
&	Logical AND
|	Logical OR
⊕	Logical EXCLUSIVE OR
( )	Contents of
–( )	Negation (two's complement)
#	Immediate value
«	Sign extend
←	Loaded with
?	If
:	Concatenated with
↕	Set or cleared
—	Not affected

Fonte: Data Sheet M68HC08 – Microcontrollers (2004)