

EA 870 Laboratório de Computação

Laboratório 7 – 1º. semestre de 2003 Programa monitor simplificado do HC11

1. Introdução

O objetivo deste laboratório é tratar o conceito do programa monitor do HC11. Hoje no laboratório utilizamos o BUFFALO. A idéia é que neste laboratório sejam programados e implementados quatro programas que executem funções similares às do BUFFALO.

Este roteiro está organizado nas seguintes seções:

- Seção 2 - Roteiro de Estudos: conjunto de perguntas que alinham os principais pontos que devem ser compreendidos para a execução das tarefas do laboratório;
- Seção 3 - Exercícios: conjunto de exercícios que devem ser respondidos e entregue em forma de relatório;
- Seção 4 - Referências: lista de referências que devem ser consultadas para a realização do estudo e tarefas associadas a este roteiro.
- Anexo A - Programas para teste e auxílio.

2. Roteiro de estudos

1. Em que parte do mapa de memória se localiza o programa monitor (BUFFALO)? E a memória RAM de usuário?
2. Como proceder para que o programa monitor feito neste laboratório não seja modificado indevidamente pelo programa do usuário?
3. Qual é o formato de um registro do tipo S-record (arquivo .s19)? O que significa cada par de caracteres em uma linha? Especificamente, e os dois últimos caracteres de cada linha?
4. O que significa a mensagem "rom-" que o comando MM do BUFFALO pode retornar? Em que condições ela ocorre. Como isto é implementado?
5. Quando o BUFFALO faz a chamada de outro programa através do comando CALL ou GO, como é tratada a pilha?
6. Se o comando CALL fosse simples, sem a possibilidade de breakpoints e não precisasse imprimir o conteúdo dos registradores, seriam necessárias duas pilhas?

3. Exercícios

Pede-se para fazer um programa monitor que seja um substituto do BUFFALO na execução dos comandos de display, carregamento e modificação de memória e do comando de execução de programas do usuário. O aluno está livre para implementar o seu

monitor desde que atenda os requisitos mínimos sugeridos neste roteiro. Critica-se muito que o BUFFALO é muito primitivo. Cabe a cada um programar um monitor mais amigável, porém simples.

O relatório deve conter o fluxograma dos comandos LOADT e CALL implementados, assim como a listagem do programa monitor.

O programa deste laboratório deverá ser compilado com o arquivo **crt.s** devidamente *modificado*. O início do arquivo deverá ser substituído pelo seguinte código (por quê?):

```
* define starting addresses
sect 0          * code
org $6500
sect 1          * data
org $7800
stackbase equ $7ffb
```

Para rodar os programas deste laboratório o aluno deve utilizar a linha de comando "call 6500 <enter>".

O programa de usuário a ser carregado para testes será o programa lab7teste.s, listado no Anexo A e disponível para download na página da disciplina. Perceba que este arquivo já possui o cabeçalho do crt.s e pode ser compilado com o seguinte comando:

```
ias11 -l -v -o lab7teste.s19 lab7teste.s
```

1. Implementar as seguintes subrotinas:

- void memorydisplay(int endereco), semelhante ao comando MD do BUFFALO, que mostra na tela o conteúdo de qualquer região da memória mapeada pelo HC11. O parâmetro passado é o endereço inicial. A cada linha o programa deve mostrar, na sequência: o endereço inicial; o conteúdo de 16 endereços a partir do endereço inicial; os 16 caracteres cujos códigos ASCII são os conteúdos previamente mostrados na linha. Caso o caractere seja não imprimível a subrotina deverá imprimir um espaço ou ponto. Cada chamada da subrotina deve mostrar 10 linhas. O aluno pode propor variações que ache melhores, desde que atenda o objetivo de mostrar o conteúdo da memória do HC11.
- void loadt(), semelhante ao comando Load T do BUFFALO, que lê um S-record (arquivo .s19) enviado serialmente pelo terminal através da porta SCI e armazena os dados que chegam nos endereços descritos pelo S-record. A cada linha do S-record processada a subrotina deve retornar ao terminal um ponto ('.'). Ao final da transmissão a subrotina deverá imprimir a string "Transmissão concluída com sucesso". Caso haja qualquer erro no S-record (comprimento de linha errado, checksum incoerente ou outros) a subrotina deverá imprimir a string "S-record inválido", e retornar. A recepção deverá ser baseada na rotina int getchar() do arquivo sci.c, que espera por um caractere ser enviado e então trata-o adequadamente. O programa deverá ainda ter o cuidado de não permitir que dados que chegam sejam escritos sobre o seu próprio código (!).
- void memorymodify(int endereco, unsigned char dado), semelhante ao comando MM do BUFFALO, onde dois parâmetros são passados: o endereço a ser modificado (2 bytes) e o dado que será escrito (1 byte). A subrotina deve escrever o dado no endereço especificado. Se o endereço for somente-leitura, deverá imprimir no terminal a string "rom-", semelhante ao procedimento do comando do BUFFALO.

- `void call(int endereco)`, semelhante ao comando CALL do Buffalo, que faz a execução de um programa e retorna para o monitor com um RTS. Para simplificar, não há necessidade de imprimir o conteúdo dos registradores da CPU nem de implementar o tratamento de breakpoints.
2. Fazer um programa principal que é um prompt de usuário. Se o usuário digitar os caracteres 'd', 'm', 'l', 'c', o programa deverá buscar parâmetros adicionais do usuário e executar `memorydisplay`, `memorymodify`, `loadt`, `call`, respectivamente. O caractere **CTRLC** finaliza o programa. Qualquer outro caractere digitado deverá imprimir na tela uma mensagem de erro do tipo “Comando inválido”.

Para diminuir a carga de trabalho deste laboratório, o arquivo **lib.c** possui algumas funções de auxílio que são listadas no Anexo A. que estará disponível na página da disciplina. Tente fazer um código bem simples para caber na memória disponível (6KB). Caso seja realmente difícil, faça diferentes programas, cada um em um arquivo, para serem compilados e testados separadamente. O programa então será compilado da seguinte maneira:

```
icc11 -v -o lab7.s19 lab7.c lib.c sci.c printf.c
```

O conteúdo de `lib.c` é o seguinte:

- `unsigned char ascii2hexa(unsigned char c)`, que transforma o caractere ASCII `c` em hexadecimal.
- `unsigned char ascii2hexachar(unsigned char h, unsigned char l)`, que converte dois caracteres ASCII em um único hexadecimal. Útil para MM e LOAD T.
- `unsigned int ascii2hexaint(unsigned char hh, unsigned char hl, unsigned char lh, unsigned char ll)`, que converte quatro caracteres ASCII para um inteiro de 16 bits em hexadecimal.
- `unsigned int getAddress()`, que espera quatro caracteres serem digitados e usa as subrotinas acima para convertê-los para inteiro de 16 bits em hexadecimal. Útil para receber os endereços usados em MD, MM e CALL. Esta subrotina dependa da subrotina `getchar()` do `sci.c`.

4. Referências

- /1/ *MCHC11 Reference Manual*; Motorola Inc.
- /2/ *MC68HC11A8 HCMOS Single-Chip Microcontroller Technical Data*; Motorola Inc.
- /3/ *M68HC11EVB Evaluation Board User's Manual*; Motorola Inc.
- /4/ *Motorola Freeware Cross Assembler, User's Manual*
- /5/ *PD HC11-FEEC - Manual Preliminar Interno FEEC*.
- /6/ *ICC11 DOS based C Cross Compiler for MC68HC11, User's Manual*
- /7/ *C A Linguagem de Programação, Padrão ANSI - Kernighan, Richie - Editora Campus*
- /8/ *Tutorial de programação C para HC11 - Murillo Fernandes Bernardes*

Anexo A

```
lab7teste.s:
```

```

* define starting addresses
sect 0          * code
org $6500
sect 1          * data
org $7800
stackbase equ $7ffb

```

lib.c:

```
#define CTRLC 3
```

```
/*ascii2hexa: transforma o caractere ASCII c em hexadecimal sem sinal.*/
```

```
unsigned char ascii2hexa(unsigned char c) {
    unsigned char ret = 0;
    if (c > 0x2F && c < 0x3A) ret = c - 0x30;
    else if (c > 0x40 && c < 0x47) ret = c - 0x37;
    else if (c > 0x60 && c < 0x67) ret = c - 0x57;
    else ret = 0xFF;
    return ret;
}
```

```
/*ascii2hexachar: converte dois caracteres ASCII em um único hexadecimal sem sinal.*/
```

```
unsigned char ascii2hexachar(unsigned char h, unsigned char l) {
    return (ascii2hexachar(h) * 16 + ascii2hexachar(l));
}
```

```
/*ascii2hexaint: converte quatro caracteres ASCII para um inteiro de 16 bits sem sinal em hexadecimal.
Nao trata caracteres ascii incorretos. Cuidado ao usar.*/
```

```
unsigned int ascii2hexaint(unsigned char hh, unsigned char hl, unsigned char lh, unsigned char ll) {
    unsigned int h, l;
    h = (unsigned int)ascii2hexachar(hh, hl);
    l = (unsigned int)ascii2hexachar(lh, ll);
    return(h * 0x100 + l);
}
```

```
/*getAddress: espera quatro caracteres serem digitados e usa as subrotinas acima para
converte-los para inteiro de 16 bits sem sinal em hexadecimal. Retorna
0 se for cancelada sua execucao. Esta subrotina dependa da subrotina
getchar() do sci.c*/
```

```
unsigned int getAddress() {
    unsigned int address = 0;
    int i, c, ok = 0, cancel = 0;
    unsigned char buff[4];

    while (!ok && !cancel) {
        ok = 1;
        printf("\nEndereço: ");
        for (i = 0; i < 4; i++) {
            c = getchar();
            if (CTRLC == c) {
                ok = 0;
                cancel = 1;
                address = 0;
                break;
            } else {
                if (ascii2hexachar(c) > 0x0F) {
                    printf("\nCaractere invalido!");
                    ok = 0;
                    break;
                } else {
                    buff[i] = c;
                }
            }
        }
    }
    if (ok) {
        address = ascii2hexaint(buff[0], buff[1], buff[2], buff[3]);
    }
    return(address);
}
```