EA870 - Laboratório de Computação

Laboratório 2
Processamento de Exceção - Revisão
Firmware TUTOR - Funções Utilitárias
Upload e Download
Desenvolvimento de Programas

1. Introdução

Como ferramenta de auxílio ao desenvolvimento de programas será utilizado, ao longo deste curso, as facilidades oferecidas pelo firmware. O objetivo deste laboratório é a familiarização com o jogo de instruções do MC68000 e com o ambiente de desenvolvimento disponível no SBC68K.

Este roteiro está organizado como se segue:

- Seção 2: Processamento de Exceção do MC68000
- Seção 3: O Firmware TUTOR: ferramenta de montagem e conjunto de funções utilitárias
- Seção 4: Upload e Download de código do PC para o SBC68K
- Seção 5: Roteiro de implementação de programas
- Seção 6: Exercícios.

2. Processamento de Exceção

Processamento de exceção refere-se às ações executadas pelo MC68000 que fogem da dinâmica normal associada à execução de instruções. Um processamento de exceção está associado ao tratamento de: interrupções, reset, instruções ilegais, instruções TRAP, erro de endereçamento (acesso a word e long word em endereço ímpar), entre outros eventos excepcionais.

O MC68000 encontra-se sempre em um dos seguintes três estados de processamento: **normal**, **exceção** e **parado** (halted).

O **estado normal** de processamento refere-se ao ciclo normal de execução de instruções.

O **estado de exceção** está associado à ocorrência de condições excepcionais que fogem ou modificam o processamento normal de instruções. O processamento de exceção está associado a eventos internos ou externos ao processador. Exemplos de eventos externos são: **reset**, **erro no barramento** (bus error), e **interrupção**. Internamente um processamento de exceção pode estar associado: a **erro de endereçamento**, à **execução passo-a-passo** (tracing), à execução de uma **instrução ilegal** ou não existente, a uma **violação de privilégio**, à execução de uma instrução **TRAP** ou **TRAPV**, à execução da instrução **CHK**, e à execução de uma **divisão por zero**.

O **estado parado** é um caso especial do estado normal de processamento, onde o processador não efetua processamento interno ou mesmo acesso externo (memória ou E/S). Neste estado, o processador aguarda a ocorrência de algum evento externo que o conduza novamente para o estado normal de processamento. A ocorrência de uma exceção do tipo **reset** ou **interrupção** é um evento externo que permite a mudança do estado parado para o normal, passando pelo de exceção. Observar que uma forma do processador entrar no estado parado é pela execução da instrução **STOP**.

A figura 2.1 apresenta todos os tipos de exceções. As **exceções do grupo 0** abortam a instrução que esteja em execução. As **exceções do grupo 1** permitem que a instrução em execução seja executada até o fim. As **exceções do grupo 2** ocorre como parte do processamento de uma instrução. As exceções do grupo 0 possuem maior prioridade, seguidas das do grupo 1, e finalmente pelas do grupo 2 que possuem menor prioridade. Dentro de cada grupo a figura 2.1 apresenta também as prioridades relativas em ordem decrescente. Caso duas exceções ocorram simultaneamente a de maior prioridade será atendida primeiro.

GRUPO	EXCEÇÃO	PROCESSAMENTO
0	RESET ERRO NO BARRAMENTO ERRO DE ENDEREÇAMENTO	Processamento de exceção começa assim que o evento for reconhecido
1	TRACE INTERRUPÇÃO INSTRUÇÃO ILEGAL VIOLAÇÃO DE PRIVILÉGIO	Processamento de exceção começa antes do início da próxima instrução
2	TRAP, TRAPV CHK DIVISÃO POR ZERO	Processamento de exceção é disparado pela execução da instrução

Fig.2.1: Agrupamento e prioridade das exceções

Todo processamento de exceção ocorre no **modo supervisor**. Uma vez que o processador se encontre no modo usuário, a única maneira de mudar para o **modo supervisor** é através de uma exceção. Durante o processamento de exceção o **bit S** do **registrador de estado** é feito igual a "1", de forma a colocar o processador no modo supervisor. A transição do modo supervisor para o modo usuário, entretanto, pode ser feita com as seguintes instruções (todas privilegiadas - só podendo ser executadas no modo supervisor):

1) RTE
2) MOVE <ea>,SR
3) ANDI #XX,SR
4) EORI #XX,SR.

O processamento de exceção ocorre em 4 etapas distintas:

1) Uma cópia interna do registrador de estado (SR) é feita. O bit S do SR é feito igual a "1", colocando o processador no modo supervisor. Simultaneamente tracing (execução passo-a-passo) é inibido, zerando-se o bit T do SR. Para reset e interrupção, a máscara de interrupção é atualizada.

2) O **número do vetor de exceção** é determinado. Para uma interrupção, o número do vetor pode ser lido pelo barramento de dados. Nos outros casos, o número do vetor é calculado internamente ao processador. O número do vetor de exceção é utilizado para gerar o endereço do **vetor de exceção**.
Os **vetores de exceção** são as posições de memória onde se encontram os endereços das rotina que tratam as exceções. Todos vetores de exceção ocupam 2 palavras - exceto o vetor do reset que ocupa 4 palavras (64 bits).
O número do vetor é um número de 8 bits, que multiplicado por 4 resulta no endereço do vetor de exceção.

Como apresentado na figura 2.2, os vetores de exceção ocupam as primeiras 512 palavras (1024 bytes) da memória.

3) O estado do processador é salvo (para a exceção reset, este estado não é salvo). O PC e o SR são empilhados na pilha do supervisor (SSP). O PC, via de regra, aponta para a próxima instrução não executada, entretanto para as exceções **erro no barramento** e **erro de endereçamento** o valor do PC é imprevisível. Para estas exceções informações adicionais são ainda salvas na SSP.

4) O PC recebe o conteúdo do vetor de exceção, e o processador retorna ao estado normal de processamento, reiniciando o seu interminável ciclo de busca, decodificação e execução de instruções.

No texto que se segue, discute-se os vários tipos de exceção, detalhando-se como cada exceção ocorre e o processamento associado a elas.

No.	Endereço	Atribuição
0	$000	Reset: SSP inicial
1	$004	Reset: PC inicial
2	$008	Erro no Barramento
3	$00C	Erro de Endereçamento
4	$010	Instrução Ilegal
5	$014	Divisão por Zero
6	$018	Instrução CHK
7	$01C	Instrução TRAPV
8	$020	Violação de Privilégio
9	$024	Trace
10	$028	Emulação 1010
11	$02C	Emulação 1111
12	$030	Reservado
13	$034	Reservado
14	$38	Reservado
15	$03C	Interrupção Não-Inicializada
16-23	$040-$05F	Reservado
24	$060	Interrupção Espúria
25	$064	Interrupção 1: Auto-vetor
26	$068	Interrupção 2: Auto-vetor
27	$06C	Interrupção 3: Auto-vetor
28	$070	Interrupção 4: Auto-vetor
29	$074	Interrupção 5: Auto-vetor
30	$078	Interrupção 6: Auto-vetor
31	$07C	Interrupção 7: Auto-vetor
32-47	$080-$0BF	Instruções TRAP
48-63	$0C0-$0FF	Reservado
64-255	$100-$3FF	Vetores de Interrupção do Usuário

Fig. 2.2: Vetores de Exceção

2.1 Reset

O reset é ocasionado fisicamente forçando-se as entradas reset (pino 20) e halt (pino 19) do MC68000 simultaneamente para "0". O processamento associado ao reset é, em geral, reservado para a inicialização do sistema e recuperação de falhas graves no sistema. O processamento interrompido por um reset é abortado, não existindo forma de retomá-lo. O processador é forçado para o modo supervisor com: tracing inibido e máscara de interrupção com valor 7. O número do vetor de exceção é gerado internamente, tendo o valor 0. O endereço contido nas duas primeiras palavras (32 bits) é carregado no SSP. O endereço contido nas duas palavras seguintes define o novo valor inicial do PC. A execução normal de instruções começa então a partir do endereço apontado pelo PC. Observar que programa de inicialização deve residir a partir deste endereço.

2.2 Erro no barramento

Um erro no barramento (e.g., acesso a posição de memória inexistente) pode ser sinalizado por circuito externo ao processador através de seu pino 24. Na ocorrência de erro no barramento, o ciclo de barramento é abortado, dando-se início ao processamento de exceção (vetor de exceção número 2, veja Fig. 2.2).

2.3 Erro de endereçamento

Uma exceção devido a erro de endereçamento ocorre quando da tentativa de se ter acesso a uma word ou long word residente em endereço ímpar.

2.4 Tracing: executando passo-a-passo

Para facilitar o desenvolvimento de programas, o MC68000 suporta a execução passo-a-passo (trace). No modo trace (Bit T do SR igual a "1"), após a execução de cada instrução uma exceção é gerada (vetor de exceção número 9, veja Fig. 2.2), facilitando a monitoração/depuração do programa em desenvolvimento.

2.5 Interrupção

Existem 7 níveis distintos de interrupções, cada um associado a uma prioridade. Os níveis de interrupção são numerados de 1 a 7, refletindo a sua prioridade (nível 7 prioridade máxima - nível 1 prioridade mínima). A máscara de interrupção (3 bits) contida no SR indica qual a prioridade de interrupção em vigor. Caso ocorra um pedido de interrupção com prioridade maior que a máscara de interrupção, o processamento de exceção associado a esta interrupção começa imediatamente após o término da instrução que eventualmente esteja em execução. Caso o pedido de interrupção tenha nível igual ou menor ao indicado pela máscara de interrupção, o seu atendimento é postergado até que a máscara contenha valor que permita o seu atendimento. O mecanismo de interrupção será ainda tratado detalhadamente neste curso.

2.6 Instruções ilegais e não-implementadas

O termo instrução ilegal é utilizado para designar qualquer palavra que não represente uma palavra válida de operação (operation word). Durante o processamento normal, caso uma destas palavras seja detectada, o processador inicia um processamento de exceção (vetor de exceção número 4; veja Fig. 2.2). A Motorola se reserva o direito de definir novas instruções utilizando palavras de operações ilegais. Entretanto, três palavras são reservadas para serem ilegais permanentemente: $4AFA, $4AFB, e $4AFC. As duas primeiras são reservadas para produtos motorola. A última está reservada para o usuário.

Palavras de operação com os bits 15 a 12 iguais a 1010 ($A) ou 1111 ($F) são designadas como instruções não implementadas. Vetores de exceção estão associados a estas palavras para permitir a emulação, por programa, de novas instruções (vetores de exceção número 10 e 11; veja Fig. 2.2).

2.7 Violação de privilégio

Para garantir a segurança do sistema, algumas instruções são privilegiadas, só podendo ser executadas no modo supervisor. As instruções privilegiadas são:

1) STOP,
2) RESET,
3) RTE,
4) MOVE <ea>,SR,
5) ANDI #XX,SR,
6) EORI #XX,SR,
7) ORI #XX,SR,
8) MOVE An,USP, e
9) MOVE USP,An.

A tentativa de execução de uma destas instruções no modo usuário ocasiona uma exceção de violação de privilégio (vetor de exceção número 8; veja Fig. 2.2)

2.8 Instruções que geram processamento de exceção

Algumas instruções estão associadas ao processamento de exceção. Tais instruções são: TRAP, TRAPV, DIVS, e DIVU.

A execução de uma instrução TRAP sempre força o processamento de exceção. A instrução TRAP #N gera um número de vetor de instrução igual a 32 + #N (o intervalo de números de vetor de exceção reservado para TRAP é 32-47, veja Fig. 2.2). As instruções TRAP fornecem um mecanismo seguro de permitir que os usuários tenham acesso a funções (críticas) do sistema.

As instruções TRAPV e CHK disparam o processamento de exceção se erro for detectado durante o processamento. A instrução TRAPV está associada à ocorrência de overflow. A instrução CHK está associada ao fato do conteúdo de registrador estar fora de faixa permitida (p. ex., esta instrução pode ser usada para evitar que usuário tenha acesso a elementos fora da dimensão pré-definida de um array).

As instruções de divisão DIVS e DIVU acarretam uma exceção, quando da divisão por zero.

3. O Firmware TUTOR

O TUTOR, como discutido no laboratório anterior, é um interpretador de comandos que permite ao usuário inserir, executar, testar e depurar programas no SBC68K. Em particular, no que se refere à inserção de programas, o TUTOR oferece facilidades de montagem (assembler) que permitem a programação no formato mnemônico. É importante frisar que as facilidades de montagem suportadas pelo TUTOR não são as de um montador, mas sim, a de um interpretador. Cada instrução no formato mnemônico, ao ser inserida, é imediatamente convertida (interpretada) para o seu código binário correspondente, sem que uma tabela de símbolos seja elaborada. Assim, não é possível utilizar rótulos ou símbolos para representar endereços e/ou valores. Endereços e valores devem ser sempre representados pelos seus valores numéricos. As facilidades de montagem suportadas pelo TUTOR estão descritas com todo rigor no capítulo 3 da referência [1]. A leitura deste capítulo é necessária para a correta e eficaz programação no formato mnemônico no ambiente fornecido pelo TUTOR. Observar atentamente as convenções adotadas no ambiente do TUTOR e, em particular, o tratamento dispensado às instruções tipo branch (veja página 3-18 de [1]) e aos modos de endereçamento relativos ao PC (página 3-18 de [1]).

Adicionalmente, o Firmware TUTOR oferece ainda ao usuário um conjunto de funções utilitárias para:

a) Ler ou escrever caracter ou conjunto de caracteres (string) nas interfaces de entrada e saída com o terminal de vídeo.
b) Operar a unidade de disco flexível.
c) Converter: hexadecimal para decimal, hexadecimal para ASCII, e ASCII para hexadecimal.
d) Alocar e controlar áreas de memória (buffers)
e) Transferir o controle de volta para o TUTOR, com ou sem inicialização, e
f) Inserir novas funções utilitárias.

Estas funções formam um conjunto de subrotinas acessíveis através da exceção TRAP #14, conforme a sequência de chamada (ver página 4-2 da referência [1])

```
MOVE.B    #<número da função>,D7
TRAP      #14
```

A tabela de funções suportadas, com os seus respectivos números, é apresentada a seguir:

Número	Nome	Descrição
255	----	reservado
254	----	reservado
253	LINKIT	apendicular a tabela do usuário a tabela TRAP 14
252	FIXDADD	apendicular um string endereçado por A5 ao buffer reservado e utilizado pelo TUTOR
251	FIXBUF	inicializar A5 e A6 com o endereço do buffer reservado
250	FIXDATA	inicializar A6 com o endereço do buffer reservado antes de apendicular o string endereçado por A5 a ele
249	FIXDCRLF	inicializar A6 com o endereço do buffer reservado antes de mover 'CR', 'LF'e string endereçado por A5
248	OUTCH	enviar um caracter simples à porta 1
247	INCHE	receber um caracter simples da porta 1
246	-----	reservado
245	-----	reservado
244	CHRPRINT	enviar um caracter simples à porta 3
243	OUTPUT	enviar um string à porta 1
242	OUTPUT21	enviar um string à porta 2
241	PORTIN1	receber um string da porta 1
240	PORTIN20	receber um string da porta 2
239	-----	reservado
238	-----	reservado
237	PRCRLF	enviar um string à porta 3
236	HEX2DEC	converter valor hexadecimal ao valor decimal representado em ASCII
235	GETHEX	converter caracter ASCII em hexadecimal
234	PUTHEX	converter um dígito hexadecimal no D0 para ASCII
233	PNT2HX	converter dois dígitos hexadecimais no D0 para ASCII
232	PNT4HX	converter três dígitos hexadecimais no D0 para ASCII
231	PNT6HX	converter seis dígitos hexadecimais no D0 para ASCII
230	PNT8HX	converter oito dígitos hexadecimais no D0 para ASCII
229	START	reinicializar TUTOR
228	TUTOR	desviar ao interpretador de comandos TUTOR
227	OUT1CR	enviar um string, 'CR', 'LF' à porta 1
226	GETNUMA	converter número representado em ASCII, na base hexadecimal, para representação hexadecimal
225	GETNUMD	converter número representado em ASCII, na base decimal, para

		representação hexadecimal
224	PORTIN1N	receber string da porta 1
223	DSELECT	selecionar (ativar) um dos drives
222	DRESTORE	reposicionar a cabeça na trilha 0 do drive selecionado
221	DREADS	ler um setor do drive selecionado
220	DREADT	ler uma trilha do drive selecionado
219	DSEEK	mover a cabeça para a trilha especificada no drive selecionado
218	DWRITS	escrever um setor no drive selecionado
217	DSTEPIN	avançar a cabeça de uma trilha no drive selecionado
216	DWRITT	escerver uma trilha no drive selecionado
215	UNSELECT	desativar ("deselecionar") o drive selecionado
214-128	-----	reservado
127-0	-----	funções definidas pelos usuários

Observar que algumas funções requerem a inicialização de registradores e outras destroem o conteúdo deles. Para detalhes adicionais, consultar o capítulo 4 de [1].

4. Upload e Download de código do PC para o SBC68K

Durante o desenvolvimento de programas no ambiente deste laboratório far-se-á necessário salvar o conteúdo de posições de memória do SBC68K em disco flexível, assim como, restaurar o conteúdo destas posições de memória previamente armazenadas em disco flexível.

Para salvar o conteúdo de um bloco de memória do SBC68K em disco digitar, no prompt do DOS, o comando:

upload <endereço_inicial> <endereço_final> <nome_do_arquivo>

O programa Upload transfere o conteúdo do bloco de memória que se inicia no endereço_inicial e terminal no endereço_final para o arquivo especificado. O programa Upload transfere o conteúdo do bloco de memória do SBC68K para o PC através linha serial que interliga estas máquinas pelos conectores TERMINAL (SBC68K) e Canal A (COM1 - PC). O conteúdo do bloco de memória é transferido, e armazenado no arquivo, no formato S-Record.

O formato S-record é um formato legível (formado apenas por caracteres ASCII) utilizado para transferir programas e dados entre computadores. Neste formato, as informações a serem transferidas são codificadas em um conjunto de seqüências de caracteres (strings) denominados S-records. Um S-record é composto dos seguintes campos:

tipo	tamanho	endereço	dados	Checksum

O campo tipo é formado por 2 caracteres (por exemplo: S0, S1, etc). O campo tamanho, dado por 2 caracteres, especifica o número de bytes (número de pares de caracteres) que compõem o S-record (excluídos os campos tipo e tamanho). O campo de endereço pode ter 4, 6 ou 8 caracteres, e especifica endereços de 16, 24 ou 32 bits respectivamente. O campo dados tem tamanho variável, podendo ter de 0 a 2n caracteres (representar 0 a n bytes). O campo checksum, formado por 2 caracteres, é o byte menos signficativo do complemento de 1 da soma dos valores (bytes) representados pelos pares de caracteres que formam os campos de tamanho, endereço e dados.

O TUTOR e o programa Upload suportam os seguites tipos de S-records:

S0 – S-record de início de um conjunto de S-records. O campo de dados pode conter informação descrevendo o bloco de S-record que se segue (por exemplo, identificação do programa que gerou os S-records). O campo de endereço contém o valor 0000.

S1 – S-record contendo informação (campo de dados) que residem a partir do endereço especificado em 16 bits (campo de endereço).

S2 – S-record contendo informação (campo de dados) que residem a partir do endereço especificado em 24 bits (campo de endereço).

S9 – S-record que finaliza um bloco de S-records.

O conjunto de S-records que se segue codifica o conteúdo das posições de memória \$1000 até \$101F:

S0030000FC
S1131000000102030405060708090A0B0C0D0E0F64
S1131010101112131415161718191A1B1C1D1E1F54
S9030000FC

Neste exemplo tem-se um S-record do tipo S0 (início), dois S-record do tipo S1 (especificando o conteúdo das posições de memória), e um S-record do tipo 9, finalizando o conjunto. A decomposição destes S-records em seus respectivos campos é:

tipo	tamanho	endereço	Dados	checksum
S0	03	0000		FC
S1	13	1000	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	64
S1	13	1010	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F	54
S9	03	0000		FC

O S-record S0 indica o início do conjunto sem conter informação adicional de identificação.

O primeiro S-Record do tipo S1 é composto das seguintes informações (pares de caracteres):

S1	identifica o tipo de S-record.
13	Hexadecimal 13 (decimal 19) indica o número de pares de caracteres que se seguem.
10 00	4 caracteres especificando o endereço \$1000 de 16 bits
00	Hexadecimal 00 (decimal 00) conteúdo da posição de memória \$1000
01	Hexadecimal 01 (decimal 00) conteúdo da posição de memória \$1001
02	Hexadecimal 02 (decimal 02) conteúdo da posição de memória \$1002
…	…
0F	Hexadecimal 0F (decimal 0F) conteúdo da posição de memória \$100F
64	Checksum = (\$13 + \$10 + \$00 + \$00 + \$01 + \$02 + … + \$0E + 0F) negado

O S-record S9 finaliza o conjunto de S-records.

No exemplo, o conjunto de S-records especifica os valores \$00 \$01 \$02 \$03 até \$1F para o conteúdo das posições de memória de \$1000 até \$101F.

O conteúdo de um bloco de memória salvo com o programa Upload pode restaurado com o programa Download. Para tanto digitar no prompt do DOS

download <nome_arquivo>

Os programas upload e download foram desenvolvidos pelo Prof. J. Mario De Martino.

5. Roteiro de implementação de programas

Como metodologia para a implementação dos programas deste curso sugere-se o seguinte roteiro:

1. Ler o enunciado e entender o problema;
2. Definir a estrutura de dados e elaborar um fluxograma do programa a ser implementado;
3. Codificar o fluxograma em linguagem de máquina, utilizando o conjunto de instruções do MC68000 no formato mnemônico;
4. Inserir o programa em linguagem de máquina na memória do SBC68K;
5. Executar, testar e depurar o programa;
6. Imprimir uma listagem do programa; e
7. Comentar a listagem.

A seguir, discute-se cada um dos itens deste roteiro.

5.1 Ler o enunciado e entender o problema

A partir deste laboratório, a maioria dos exercícios propostos estão associados à implementação de um ou mais programas. Os enunciados dos exercícios devem ser lidos e entendidos para uma clara compreensão do problema a ser resolvido. A partir da compreensão do problema é recomendável que uma descrição funcional do programa a ser implementado seja elaborada. Esta descrição tem como objetivo detalhar e delimitar o escopo e funcionalidade do programa a ser implementado, servindo de referência para os passos seguintes.

5.2 Definir a estrutura de dados e elaborar um fluxograma do programa a ser implementado

Antes de qualquer trabalho de codificação, é importante descrever a solução através da descrição das estruturas de dados, assim como, dos algoritmos a serem utilizados.

As estruturas de dados definem como os dados serão/estão organizados (variável isolada, vetor, lista ligada, pilha, etc.), para que os mesmos possam ser devidamente processados pelo algoritmo a ser utilizado.

Uma forma interessante de se descrever o processamento a ser realizado é através de um fluxograma. Um fluxograma descreve não só as ações a serem executadas, como também, a ordem (fluxo) de execução desta ações. Em geral, a elaboração de um fluxograma pode passar por vários refinamentos, passando de um descrição global da solução até um grau de detalhamento que permita uma codificação quase imediata.

5.3 Codificar o fluxograma em linguagem de máquina, utilizando o conjunto de instruções do MC68000 no formato mnemônico

Uma vez elaborado o fluxograma, inicia-se o processo de codificação propriamente dito. Um primeiro passo é definir como as estruturas de dado serão implementadas: se em memória (quais posições ?), ou em registradores (quais registradores ?). Em seguida, faz-se necessário converter as ações do fluxograma em instruções em linguagem de máquina. A experiência (domínio do jogo de instruções) e o bom senso (não complicar se é possível simplificar) definem as diretrizes básicas da boa codificação.

5.4 Inserir o programa em linguagem de máquina na memória do SBC68K

A inserção do programa na memória do SBC68K deve ser efetuada preferencialmente no formato mnemônico, através do comando MM com a opção DI (ver pág. 2-24 e capítulo 3 de [1]). Já a inserção de dados pode ser efetuada com o auxílio dos comandos MM, MS, BF, .A0 - .A7, .D0 - .D7, entre outros.

Para salvar o programa em disquete utilizar o programa Upload (seção 4). Para carregá-lo do disquete para a memória utilizar o programa Download (seção 4).

5.5 Executar, testar e depurar o programa

Para executar um programa deve-se inicializar eventuais registradores, posições de memória e dispositivos periféricos que serão utilizados pelo programa. Essa inicialização pode ser feita pelos comandos do TUTOR, antes da transferência de controle de execução do interpretador de comandos TUTOR para o programa do usuário, ou pelo próprio programa. Observar que

- O programa precisa estar residente em memória principal para ser executado.
- O conteúdo do apontador de pilha ativo (A7) deve ser válido. Como regra prática é sugerido que a primeira instrução dos programas seja

  LEA $1000,A7

  e que os mesmos sejam carregados a partir da posição $1000

A transferência de controle de execução do programa TUTOR para o programa do usuário faz-se através dos comandos: GD, GO, G, GT, TR e TT.

O teste e a depuração de um programa, passa, em geral, pelo acompanhamento dos valores intermediários produzidos durante a sua execução. Este acompanhamento pode ser efetuado somente em pontos chaves (breakpoints), ou mesmo após a execução de cada instrução que compõe o programa (execução passo-a-passo). Em ambos os casos, o acompanhamento é efetuado através do retorno do controle ao TUTOR, permitindo, por exemplo, que o programador inspecione e altere conteúdos de memória e de registradores.

O TUTOR suporta a definição de breakpoints através dos comandos: BR, NOBR, GT e TT, enquanto que a execução passo-a-passo é suportada pelos comandos: TR e TT.

Informações complementares encontram-se na seção 3.4 da referência [1].

5.6 Imprimir uma listagem do programa

Para a impressão das instruções no formato mnemônico de um programa residente em memória, pode-se, entre outras formas, utilizar o comando MD3. Observar que a impressora deve estar devidamente conectada e ligada ao SBC68K.

5.7 Comentar a listagem

Identificar, para cada bloco funcional apresentado no fluxograma, o conjunto de instruções correspondente, anotando na listagem a funcionalidade associada. Indicar também a utilização dos registradores e das posições de memória.

Anexar à listagem do programa, uma descrição sucinta de todas as informações necessárias para garantir a execução correta do programa, incluindo o nome do arquivo que contém o programa, em que endereço de memória o programa deve ser carregado, quais posições de memória e/ou registradores devem ser inicializados, etc.

6. Exercícios

OBS.: PARA CADA PROGRAMA IMPLEMENTADO ENTREGAR:

1. **Descrição do procedimento para a execução do programa;**
2. **Fluxograma explicativo do algoritmo utilizado;**
3. **Listagem comentada do programa; com referências aos blocos funcionais apresentados no fluxograma; e**
4. **Código executável em disquete compatível com o programa Download.**

1. (1,0 pt) O programa abaixo executa a seguinte operação: a = b + c. As variáveis de 8 bits estão alocadas nas seguintes posições de memória respectivamente: $3000, $3100 e $3200.

```
LEA       $1000,A7     ; inicializa pilha
MOVE.B    $3100,D0     ; carrega b
ADD.B     $3200,D0     ; b + c
MOVE.B    D0,$3000     ; guarda resultado em a
MOVE.B    #228,D7      ; retorna para o TUTOR
TRAP      #14
```

Inserir o programa a partir do endereço $1000 e testá-lo com dados que, imediatamente após a execução da instrução ADD, resultem em (utilizar o recurso de BREAKPOINT do TUTOR para visualizar os resultados):

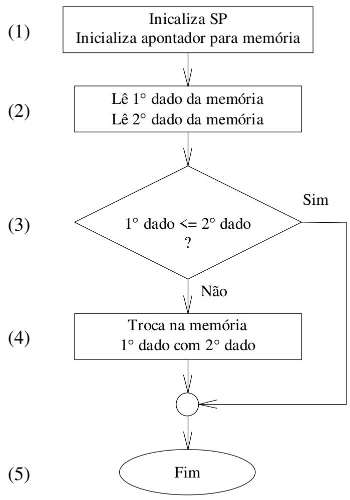
(a) C = 0 e Z = 0 (carry = 0 e zero = 0 do CCR),
(b) C = 0 e Z = 1,
(c) C = 1 e Z = 0, e
(d) C = 1 e Z = 1.

EXPLICITAR NO RELATÓRIO OS VALORES DE TESTE UTILIZADOS.

2. (1,0 pt) O programa abaixo ordena, na ordem crescente, os elementos de um vetor. O vetor é composto por 2 elementos de 16 bits com sinal, armazenados a partir da posição

$3000. (Observe que a instrução BLE - Branch if Less or Equal - é um desvio condicional Bcc, onde cc é LE)

```
        LEA       $1000,A7    (1)
        LEA       $3000,A0    (1)
        MOVE.W    (A0)+,D0    (2)
        MOVE.W    (A0)+,D1    (2)
        CMP.W     D1,D0       (3)
        BLE.L     FIM         (3)
        MOVE.W    D0,-(A0)    (4)
        MOVE.W    D1,-(A0)    (4)
FIM:    MOVE.B    #228,D7     (5)
        TRAP      #14         (5)
```

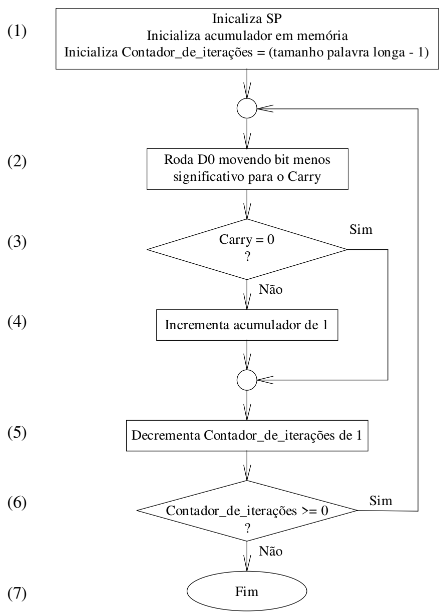


(a) O programa continuará a funcionar corretamente caso a instrução BLE seja substituída pela instrução BMI (Branch if Minus)? Justifique. (Dica: Testar com os seguintes valores em decimal: +20000 e -20000).

(b) Qual é a condição de desvio a ser utilizada para que o programa ordene corretamente 2 números SEM SINAL? Justifique (Dica: Testar com os seguintes valores em decimal: 60000 e 1)

3. (1,0 pt) Considere o programa abaixo que conta quantos bits iguais a 1 existem na palavra longa (long word) residente no registrador D0, e coloca o resultado na posição de memória $3000.

```
          LEA       $FFE,A7     (1)
          MOVE.B    #0,$3000    (1)
          MOVE.B    #31,D7      (1)
VOLTA:    ROR.L     #1,D0       (2)
          BCC.L     PULA        (3)
          ADD.B     #1,$3000    (4)
PULA:     SUBI.B    #1,D7       (5)
          BPL.L     VOLTA       (6)
          MOVE.B    #228,D7     (7)
          TRAP      #14         (7)
```



Modifique o programa de forma a utilizar a instrução DBF (Test Condition, Decrement, and Branch if False - veja instrução DBcc).

4. (1,0 pt) Fazer um programa para apresentar o conteúdo hexadecimal do registrador A0 na tela do terminal. Para a conversão hexadecimal-ASCII, utilize a função utilitária do TUTOR

PNT8HX. Para a saída na tela, utilize a função utilitária OUTPUT. (Exemplo: O registrador A0 contém o valor $1C35DE76. Na tela deve aparecer os símbolos 1C35DE76).

5. (1,5 pt) Fazer um programa para apresentar o conteúdo hexadecimal do registrador A0 na tela do terminal, sem o uso das funções utilitárias de conversão do TUTOR. Para a saída na tela do terminal, utilizar a função OUTCH. (Exemplo: vide exercício anterior).

6. (1,5 pt) Fazer um programa que ecoe no vídeo cada caracter digitado pelo usuário seguido do seu respectivo valor hexadecimal. O caracter CTRL-A ($01), ao ser digitado, deve terminar o programa. (Exemplo: O usuário tecla a letra A. Aparece no vídeo a letra A seguida de 41).

7. (1.5 pt) Escrever um programa que funcionalmente seja equivalente à função utilitária GETNUMA do TUTOR (veja seção 4.3.3 de [1]). GETNUMA converte seqüência de caracteres ASCII representando um número hexadecimal para o seu respectivo valor. Considerar apenas números representáveis em 16 bits. (Exemplo: A entrada é composta dos valores ASCII $41 $30 (´A´ ´0´). Após a execução da função, tem-se em D0 o valor $00A0.

8. (1,5 pt) Fazer um programa que funcionalmente seja equivalente à função utilitária GETNUMD do TUTOR (veja seção 4.3.3 de [1]. GETNUMD converte seqüência de caracteres ASCII representando um número decimal para o seu respectivo valor. Considerar apenas números representáveis em 16 bits. (Exemplo: A entrada é composta dos valores ASCII $31 $30 (´1´ ´0´). Após a execução da função, tem-se em D0 o valor $000A).

7. Referência

[1] Arnewsh Inc.: *SBC68K User's Manual Revision 1.1*, 2nd Edition, 1990

ANEXO A

TABELA ASCII