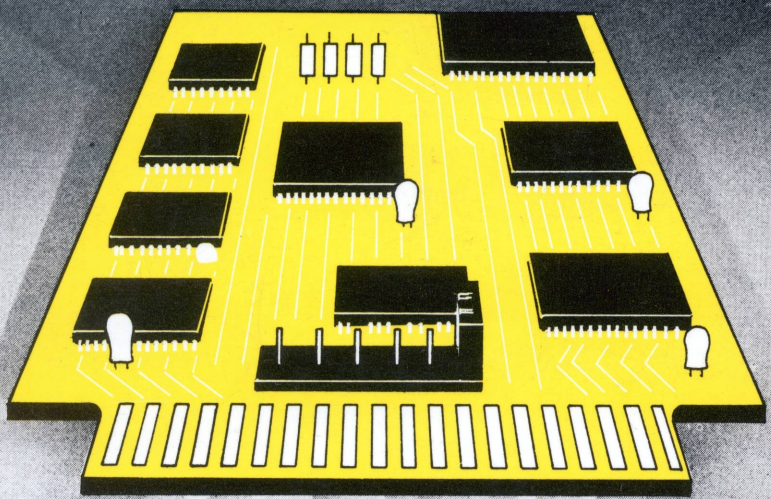
Wagner Ideali

68000

Família de Microprocessadores - 32 bits



Vol. 2

SOFTWARE

ÉRICA

68.000

FAMÍLIA DE
MICROPROCESSADORES — 32 Bits

SOFTWARE
Vol. 02

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

Ideali, Wagner, 1957-
I22s 68000 : família de microprocessador 32 bits /
v.1-2 Wagner Ideali. -- São Paulo : Érica, 1987-

Bibliografia.
Conteúdo: v. 1. Hardware -- v. 2. Software.

1. Motorola 68000 (Microprocessador) I. Título.

88-0921

CDD-001.6404
-001.642

Índices para catálago sistemático:

1. Microprocessadores 68000 : Programação : Processamento de dados 001.642
2. Motorola 68000 : Microprocessadores : Processamento de dados 001.6404

WAGNER IDEALI

68.000

FAMÍLIA DE MICROPROCESSADORES — 32 Bits

SOFTWARE
Vol. 02

ANO: 1993 92 91 90 89 88
EDIÇÃO: 10 9 8 7 6 5 4 3 2 1

LIVROS ÉRICA EDITORA LTDA.

LIVROS ÉRICA EDITORA LTDA.
Rua Jarinu, 594 - Tatuapé - São Paulo
Fone: 294-8686 - C.G.C. 50.268.838/0001-39
Caixa Postal 15.617

DEDICATÓRIA

Dedico esta obra aos meus sogros Antonio e Aparecida, à minha esposa Walkíria e aos meus filhos Thiago e Thayná, pelo apoio e compreensão constantes na elaboração da mesma.

PREFÁCIO

Dando sequência ao trabalho sobre a família do microprocessador 68000, iniciado no volume 1, vamos neste segundo volume, analisar o software, ou seja, a programação da pastilha 68000, bem como os outros microprocessadores pertencentes à família, tais como, 68008, 68010 e 68012.

Neste segundo volume, iniciaremos com uma análise rápida da constituição interna básica do 68000, modos de endereçamento, formato dos dados e etc.

Em seguida, analisaremos cada instrução em separado para que permita que essas possam ser acessadas e estudadas, quando forem necessárias durante a elaboração de um projeto.

Será abordada toda a parte de software que já envolve o processamento de excersão, pois este processamento especial é um dos pontos que forma os microprocessadores da família 68000 poderosos.

No que tange aos processadores 68010 e 68012, além das instruções será também abordada uma análise nas operações de Loop.

Por fim, serão feitos alguns exemplos de programas para ilustrar a programação em assembly da família 68000.

Espero com esta obra, transmitir ao estudante ou profissional da área de digital, mais especificamente na área dos microprocessadores, uma pequena parcela de conhecimento.

SUMÁRIO

INTRODUÇÃO

FAMÍLIA DOS PROCESSADORES 68000

O mundo da informática é dividido em duas partes, já bem conhecidas:

- Hardware
- Software

No Hardware, nós temos toda a estrutura eletrônica de um sistema de computação. O microprocessador, as memórias, periféricos, circuitos digitais para adequar sinais e porque não dizer a própria fonte de alimentação.

Toda esta estrutura eletrônica sofisticada, complexa e muitas vezes de difícil entendimento, de nada valeria sem a sua complementação que é o **Software.**

O software de um computador é toda parte "programável" , isto é, tudo aquilo que se pode alterar no sistema, sem que seja necessário alterar o circuito.

O sistema operacional que controla o computador, os programas de interfaceamento, os programas aplicativos, as linguagens utilizadas, etc. pertencem à parte denominada Software.

Antes de entrarmos nas instruções propriamente ditas da família 68000, vamos analisar alguns aspectos, não menos importantes, que são a estrutura do microprocessador 68000 e sua sofisticada família.

Quem desenvolve software em Assembly, sabe quanto é trabalhoso o desenvolvimento de um programa nesta linguagem, principalmente nos antigos microprocessadores de 8 bits.

Na tecnologia dos microprocessadores de 16 e 32 bits, tem-se incorporado sensíveis e profundas alterações para permitir, no desenvolvimento de projetos, um trabalho mais rápido, fácil e realizável.

A arte de programar os microprocessadores tem evoluído rapidamente nos últimos anos. A família dos processadores 68000 contém algumas instruções comparadas às existentes em linguagens de alto nível tais como, TRACE, STOP, DIVISÃO e etc.

Poderemos sentir durante o desenrolar desta obra, os inú meros recursos que contém as instruções e a arquitetura do 68000 e que serão de grande utilidade no desenvolvimento de um software.

I.1 Estrutura Interna

A família do 68000 pode operar com diferentes comprimen tos de dados, para a mesma instrução. Quando for usar uma ins trução, o programador define a instrução que deseja, o tipo de endereçamento e o comprimento dos dados que pode ser de 8 bits (1 byte), 16 bits (1 word) ou 32 bits (1 long word).

Os registradores internos para dados são chamados DØ até D7. Isto facilita a memorização dos mesmos, pois temos 8 regis tradores de dados de 32 bits, definidos como registrador DØ até registrador D7.

Há um grupo de sete registradores para os endereçamentos definidos como registradores AØ até A6.

Como todo microprocessador, este também contém Stack Pointer, programa counter e os flags denominados pelo fabricante do 68000 de "Condition code register".

Na figura I.1, podemos observar o conjunto de registrado res, Stack Pointer e Contador de Programa.

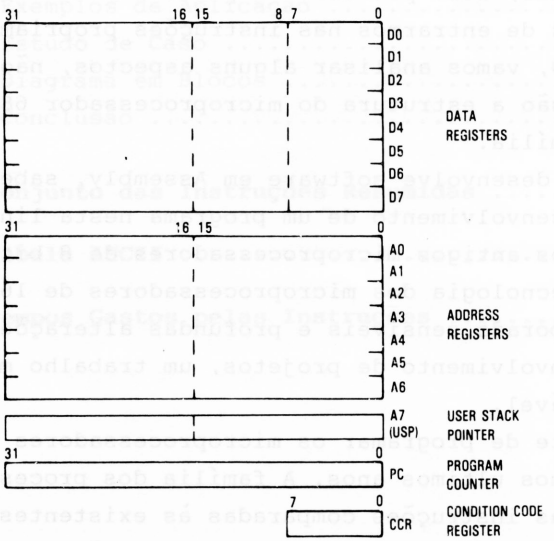


Figura I.1 - Conjunto de registradores do 68000.

A família 68000 tem dois modos de operação:

1 - **Modo usuário**: chamado assim para definir uma forma inferior de processamentos, onde certas instruções não são permitidas, bem como determinados tipos de acessos à memória, interrupções, etc. O quadro para uma operação no modo usuário é o da figura I.1.

2 - **Modo Supervisor:** Considerado um sistema superior de processamento. Utilizado quando se tem um sistema operacional, ou um programa gerenciador sobre outros (modo usuário). Neste modo de programação, todos os processamentos disponíveis no processador são permitidos, bem como, todas as instruções e interrupções. Quando estamos operando no modo supervisor, o sistema tem o seu próprio Stack Pointer, chamado Stack Pointer do supervisor. Há também um registrador de status, conforme a figura I.2.

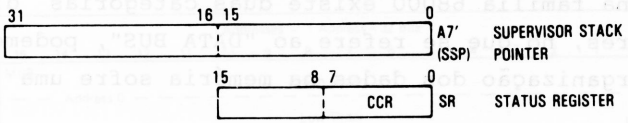


Figura I.2 - Complemento para o modo supervisor.

Para melhor entender o registrador de status, podemos ver na figura I.3 que o mesmo é composto de cinco flags, uma máscara de interrupção, um flip-flop para troca de modo de processamento (usuário ou supervisor) e o flip-flop para operar no modo do step-by-step (TRACE MODE). Este processo de step-by-step é uma facilidade que o 68000 apresenta para o desenvolvimento de software.

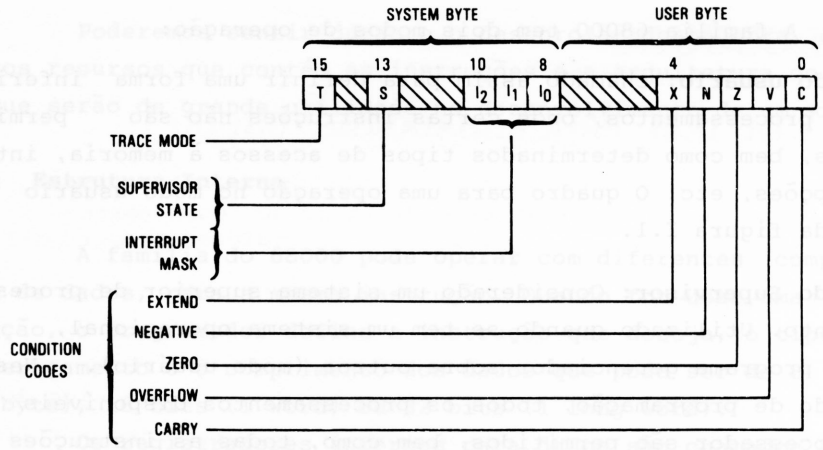


Figura I.3 - Registrador de Status.

I.2 Organização dos Dados e Capacidade de Endereçamento

Como na família 68000 existe duas categorias diferentes de processadores, no que se refere ao "DATA BUS", podemos verificar que a organização dos dados na memória sofre uma pequena diferença.

Nos microprocessadores 68000, 68010 e 68012, os dados são organizados conforme a figura I.4, enquanto no 68008, a organização dos dados está montada conforme a figura I.5.

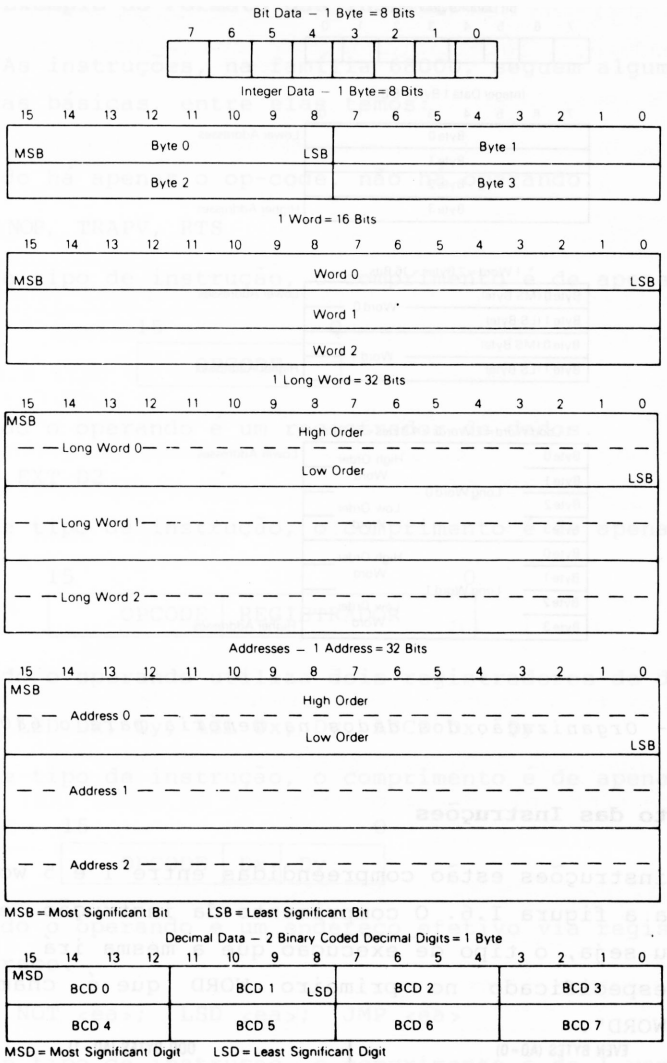


Figura I.4 - Organização dos dados na memória para o 68000,68010 e 68012.

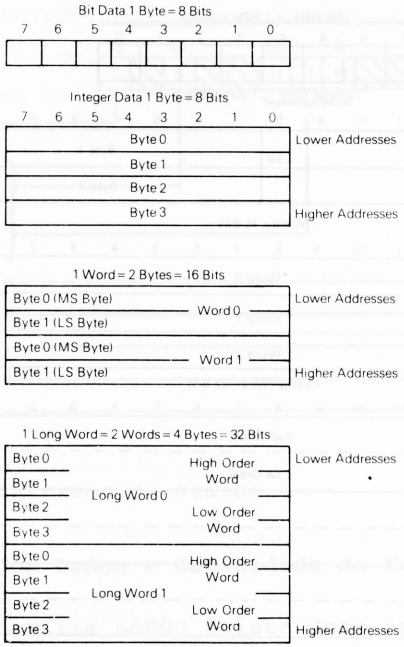


Figura I.5 - Organização dos dados na memória para o 68008.

I.2.1 Formato das Instruções

As instruções estão compreendidas entre 1 e 5 words, conforme mostra a figura I.6. O comprimento da instrução e a sua operação, ou seja, o tipo de execução que a mesma irá desempenhar, é especificado no primeiro WORD que é chamado de "OPERATION WORD".

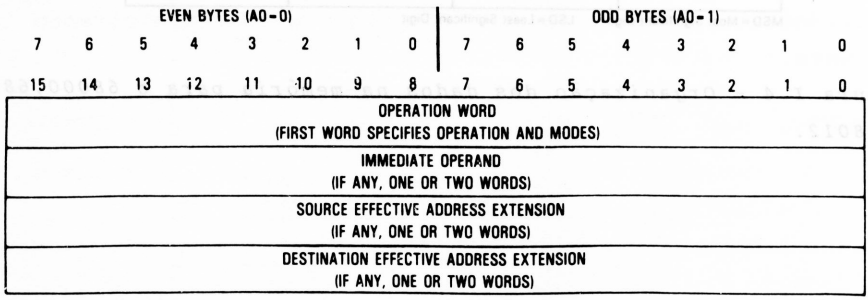


Figura I.6 - Formato de uma instrução.

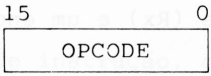
I.2.1.1 Exemplo do Formato das Instruções

As instruções, na família 68000, seguem algumas características básicas, entre elas temos:

1 - Quando há apenas o op-code, não há operando.

Ex.: NOP, TRAPV, RTS

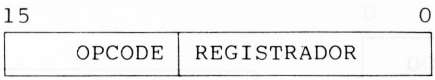
Neste tipo de instrução, o comprimento é de apenas 1 word.



2 - Quando o operando é um registrador de dados.

Ex.: EXT D2

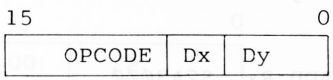
Neste tipo de instrução, o comprimento é de apenas 1 word.



3 - Quando o operando utiliza dois registradores de dados.

Ex.: LSD Dx, Dy, ASL Dx, Dy, ABCD Dx, Dy.

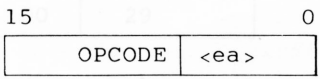
Neste tipo de instrução, o comprimento é de apenas 1 word.



4 - Quando o operando é um endereço efetivo via registrador de endereço.

Ex.: NOT <ea>; LSD <ea>; JMP <ea>

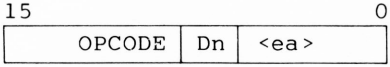
Neste tipo de instrução, o comprimento é de 1 word.



5 - Quando o operando é um endereço efetivo via registrador de endereços e um registrador de dados.

Ex.: MULU <ea>, Dn; LSD <ea>, Dy; ADD D5, <ea>

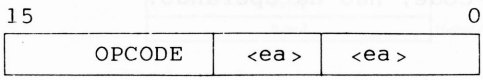
Neste tipo de instrução, o comprimento é de 1 word.



6 - Quando o operando são dois endereços efetivos via registrador de endereço. Permitido somente para a instrução "MOVE".

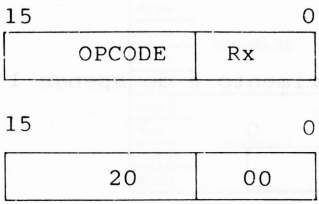
Ex.: MOVE <ea>, <ea>

Neste tipo de instrução, o comprimento é de 1 word.



7 - Quando o operando é um registrador de endereços (endereço efetivo), ou registrador de dados (Rx) e um endereço absoluto curto.

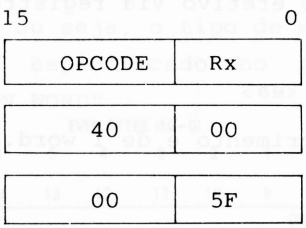
Ex.: MOVE Rx, $21ØØ; ADD Rx, $3ØØØ



Neste tipo de instrução, o comprimento é de dois words.

8 - Quando o operando é formado por um registrador de endereço (endereço efetivo), ou um registrador de dados (Rx) e um endereço absoluto longo.

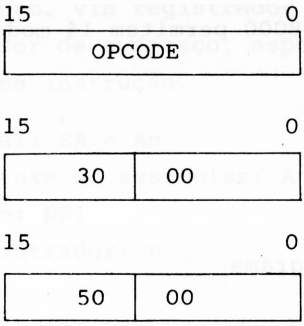
Ex.: MOVE Rx, $5F4ØØØ ; ADD Rx, $2A3ØØØ



Neste tipo de instrução, o comprimento é de 3 words.

9 - Quando o operando é formado por dois endereços absoluto curto. Permitido somente para a instrução "MOVE".

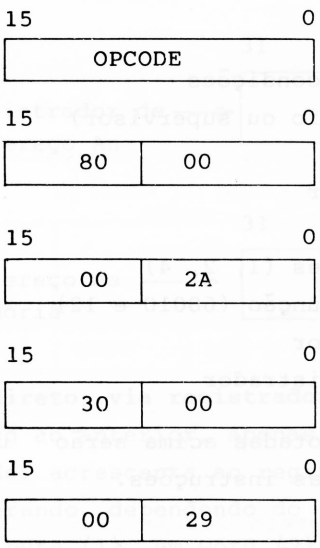
Ex.: MOVE $3ØØØ, $5ØØØ



Neste tipo de instrução, o comprimento é de 3 words.

10 - Quando o operando é formado por dois endereços absoluto longo. Permitido somente para a instrução "MOVE".

Ex.: MOVE $2A8ØØØ, $293ØØØ



Neste tipo de instrução, o comprimento é de 5 words.

I.2.2 Tipos de Endereçamento

A seguir, vamos fazer uma descrição dos tipos de endereçamentos possíveis para a família do 68000.

Os processadores da família 68000 permitem 14 modos de endereçamento, baseados em 6 tipos:

- Direto, via registradores
- Indireto, via registradores
- Absoluto
- Imediato
- Relativo ao contador de programa
- Implícito

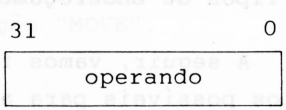
Para ficar claro o esquema montado a seguir, vamos seguir alguns mnemônicos adotados, a saber:

AN = Registrador de endereço
DN = Registrador de dados
Rn = Qualquer registrador (endereço ou dados)
XN = Qualquer registrador indexado (endereço ou dado)
PC = Contador de programa
SR = Registrador de Status
CCR = Registrador de códigos de condições
SP = Stack Pointer ativo (usuário ou supervisor)
USP = Stack Pointer do usuário
SSP = Stack Pointer do supervisor
d = Valor qualquer
N = Tamanho do operando em Bytes (1, 2, 4)
SFC,DFC = Registrador de código de função (68010 e 12)
VBR = Registrador da base de vetor
n = Especifica o número do registrador

Os símbolos e abreviações adotadas acima serão utilizadas no capítulo 1 para a descrição das instruções.

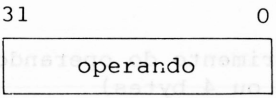
1 - Direto, via registrador de dados: O operando é o registrador de dados especificado no campo de endereço efetivo na instrução.

Geral: EA = Dn
Sintaxe do assembler: = Dn
Modo: = ØØØ
Registrador: = n



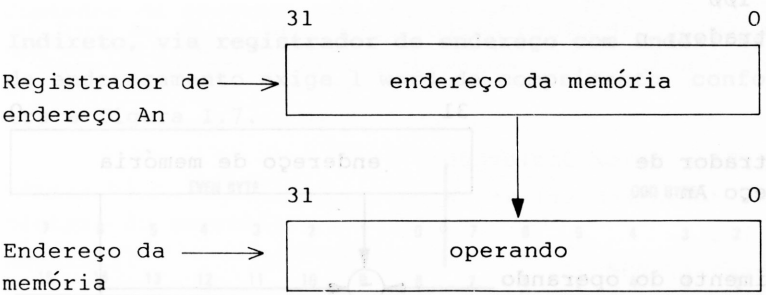
2 - **Direto, via registrador de endereço:** O operando é o registrador de endereço, especificado no campo de endereço efetivo na instrução.

Geral: EA = An
Sintaxe do assembler: An
Modo: ØØ1
Registrador: n



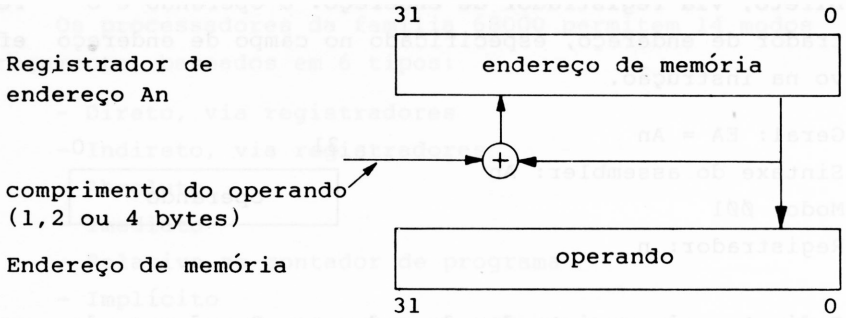
3 - **Indireto, via registrador de endereço:** O endereço do operando está no endereço apontado pelo registrador de endereço, especificado no campo de endereço da instrução.

Geral: EA = (An)
Sintaxe do assembler: (An)
Modo: Ø1Ø
Registrador: n



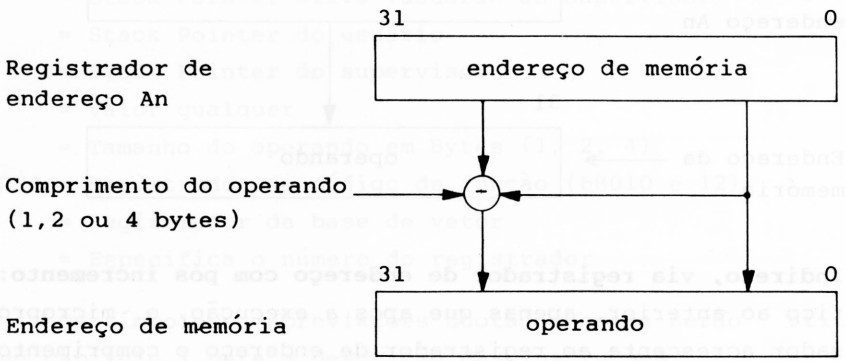
4 - **Indireto, via registrador de endereço com pós incremento:** Idêntico ao anterior, apenas que após a execução, o microprocessador acrescenta ao registrador de endereço o comprimento do operando, dependendo do seu comprimento. O mesmo pode ser um byte (1), um word (2), ou um long word (4).

Geral: An = An + N
Sintaxe do assembler: (An)+
Modo: Ø11
Registrador: n



5 - **Indireto, via registrador de endereço com pré-decremento:** Realiza operação idêntica do anterior, apenas que ao invés de somar 1,2 ou 4 bytes ao operando, este tipo de endereçamento subtrai o valor do operando antes de utilizá-lo.

Geral: An = An - N
Sintaxe do assembler: - (An)
Modo: 1ØØ
Registrador: n



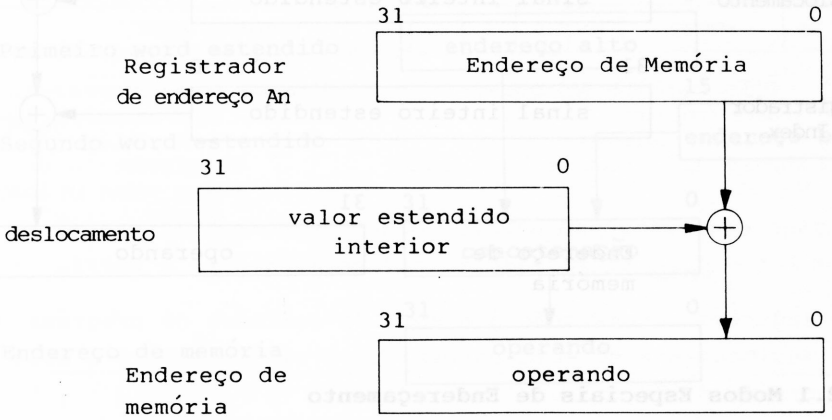
6 - **Indireto, via registrador de endereço com deslocamento:** Este modo de endereçamento requer um word de extensão. O endereço do operando é a soma do endereço que está no registrador de endereço com um deslocamento de 16 bits. Este valor de deslocamento é estendido em sinal para 32 bits, permitindo deslocamentos negativos.

Geral: EA = (An) + d16

Sintaxe do assembler: d16 (AN) ou (d16, An)

Modo: 1Ø1

Registrador: n



7 - Indireto, via registrador de endereço com index: Este modo de endereçamento exige 1 word de comprimento, conforme mostra a figura I.7.

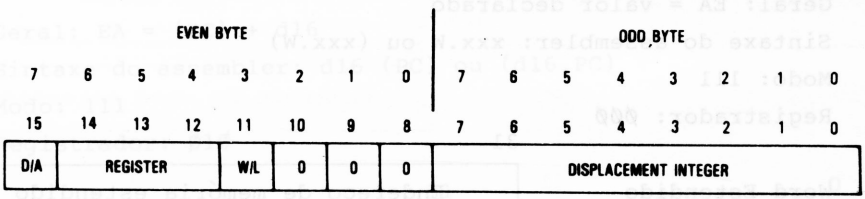


Figura I.7 - Formato do modo de endereçamento com Index.

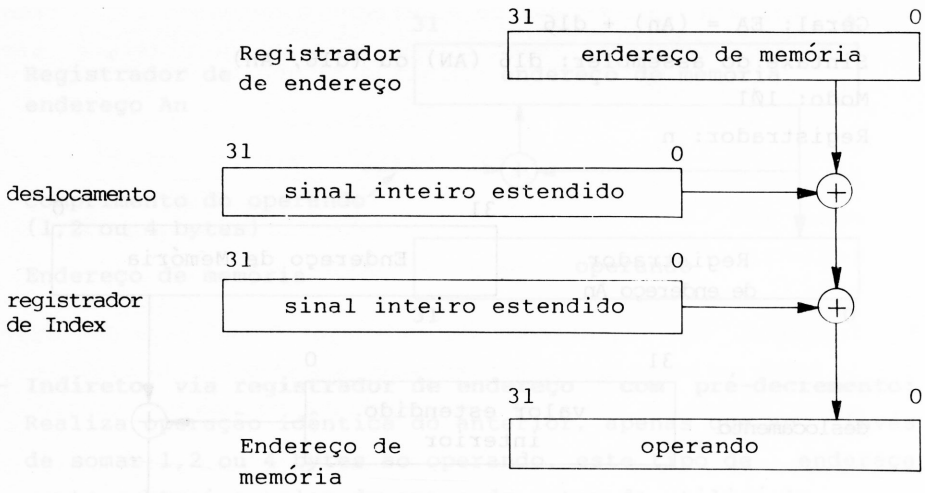
O endereço do operando é a soma do endereço dado pelo registrador de endereço, mais o deslocamento estendido em sinal, inteiro de 8 bits e mais o conteúdo de um registrador indexado.

Geral: EA = (An) + Xn + d8

Sintaxe do assembler: d8(An,Xn.W) ou (d8,An,Xn.W) ou d8(An,Xn.L) ou (d8,An,Xn,L)

Modo: 11Ø

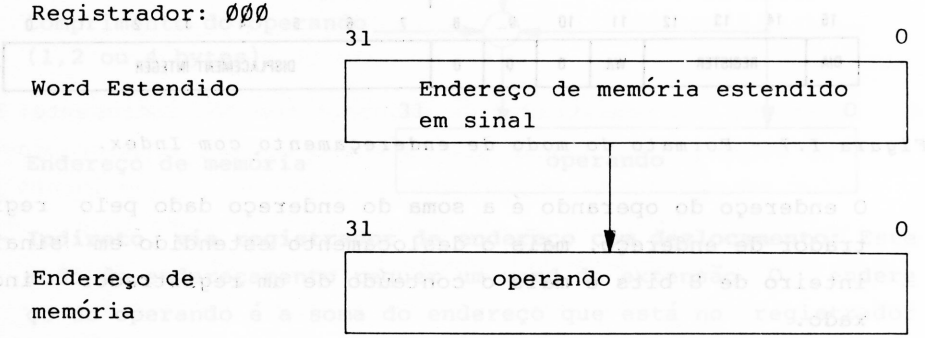
Registrador: n



I.2.2.1 Modos Especiais de Endereçamento

1 - **Endereçamento absoluto curto:** Este modo exige um word de extensão. O endereço do operando é a extensão de um word também.

Geral: EA = valor declarado
Sintaxe do assembler: xxx.W ou (xxx.W)
Modo: lll
Registrador: ØØØ



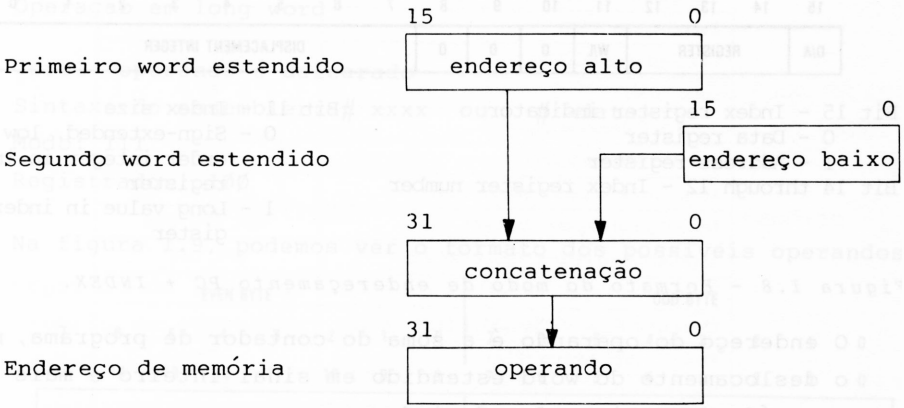
2 - **Endereçamento absoluto longo:** O modo de endereçamento requer 2 word de extensão. O endereço do operando é a concatenação de words estendidos. O word mais significativo é o primeiro word, sendo o segundo o menos significativo.

Geral: EA = valor declarado

Sintaxe do assembler: xxx.L ou (xxx.L)

Modo: 111

Registrador: ØØ1



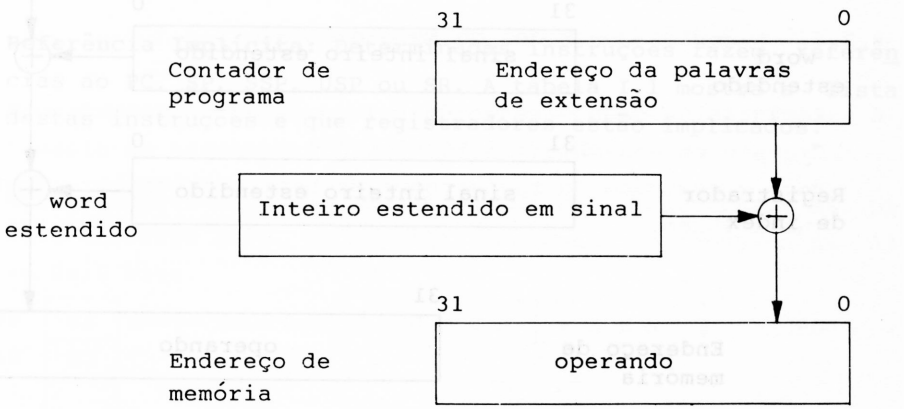
3 - **Contador de programa com deslocamento:** Este modo requer apenas um word de extensão. O valor do endereço do operando é a soma do endereço especificado no contador de programa, mais o deslocamento de 16 bits estendido em sinal.

Geral: EA = (PC) + d16

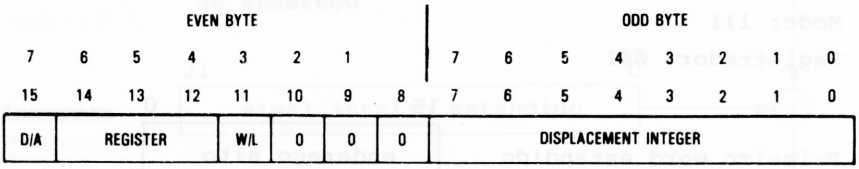
Sintaxe do assembler: d16 (PC) ou (d16,PC)

Modo: 111

Registrador: Ø1Ø



4 - Contador de programa com index: Este modo requer um word de comprimento, conforme mostra a figura I.8.



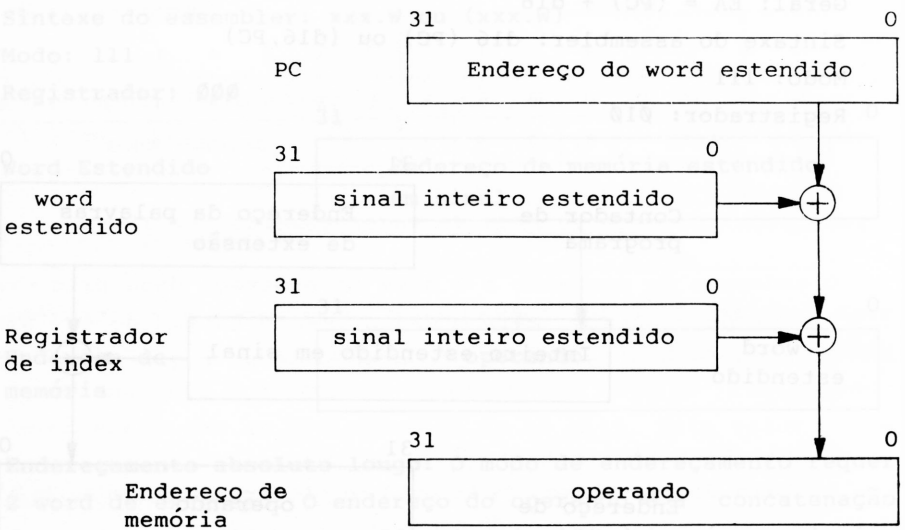
Bit 15 - Index register indicator
0 - Data register
1 - Address register
Bit 14 through 12 - Index register number

Bit 11 - Index size
0 - Sign-extended, low order integer in index register
1 - Long value in index register

Figura I.8 - Formato do modo de endereçamento PC + INDEX.

O endereço do operando é a soma do contador de programa, mais o deslocamento do word estendido em sinal inteiro e mais o conteúdo do registrador de index.

Geral: EA = (PC) +
Sintaxe do assembler: d8(PC,Xn.W) ou (d8,PC,Xn.W)
Modo: 111
Registrador: Ø11



5 - **Dado Imediato:** Este modo de endereçamento exige um ou dois words de extensão, dependendo do tamanho do operando.

Operação em byte
Operação em word
Operação em long word

Geral: operando é delcarado
Sintaxe do assembler: # xxxx ou # <data>
Modo: 111
Registrador: 1ØØ

Na figura I.9, podemos ver o formato dos possíveis operandos

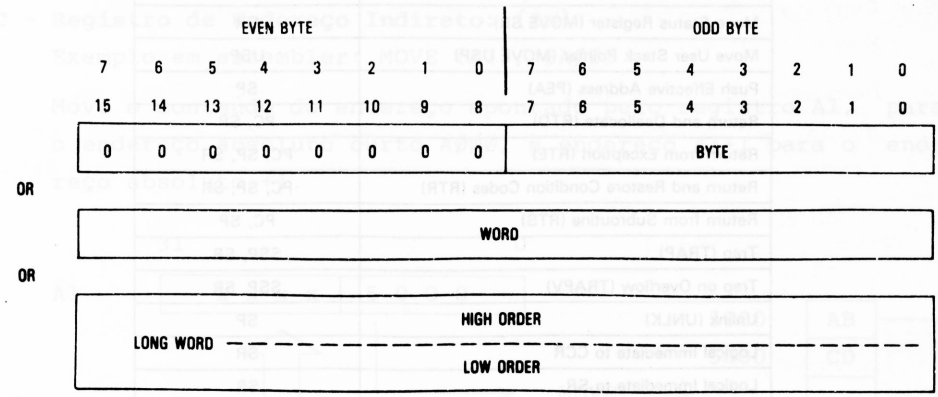


Figura I.9 - Formato dos operandos para o endereçamento de dado imediato.

6 - **Referência Implícita:** Determinadas instruções fazem referências ao PC, SP, SSP, USP ou SR. A tabela I.1 mostra a lista destas instruções e que registradores estão implicados.

Instruction	Implied Registers(s)
Branch Conditional (Bcc), Branch Always (BRA)	PC
Branch to Subroutine (BSR)	PC, SP
Check Register Against Bounds (CHK)	SSP, SR
Test Condition, Decrement and Branch (DBcc)	PC
Signed Divide (DIVS)	SSP, SR
Unsigned Divide (DIVU)	SSP, SR
Jump (JMP)	PC
Jump to Subroutine (JSR)	PC, SP
Link and Allocate (LINK)	PC, SP
Move Condition Codes (MOVE CCR)	SR
Move Control Register (MOVEC)	VBR, SFC, DFC
Move Alternate Address Space (MOVES)	SFC, DFC
Move Status Register (MOVE SR)	SR
Move User Stack Pointer (MOVE USP)	USP
Push Effective Address (PEA)	SP
Return and Deallocate (RTD)	PC, SP
Return from Exception (RTE)	PC, SP, SR
Return and Restore Condition Codes (RTR)	PC, SP, SR
Return from Subroutine (RTS)	PC, SP
Trap (TRAP)	SSP, SR
Trap on Overflow (TRAPV)	SSP, SR
Unlink (UNLK)	SP
Logical Immediate to CCR	SR
Logical Immediate to SR	SR

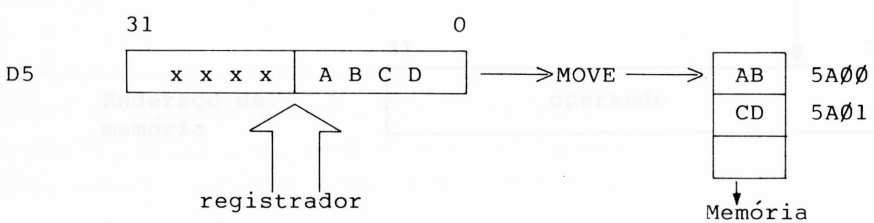
Tabela I.1 - Sumário das intruções implicadas no modo de endereçamento referência implícitas.

I.2.3 Exemplos dos Modos de Endereçamento

A - Registro de dado direto: Dn

Exemplo em assembler: MOVE D5,$5AØØ

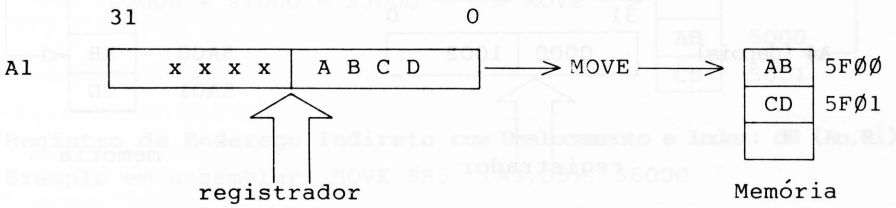
Move o conteúdo do registrador D5 para o endereço absoluto curto $5AØØ.



B - Registro de Endereço Direto: An

Exemplo em assembler: MOVE A1, $5FØØ

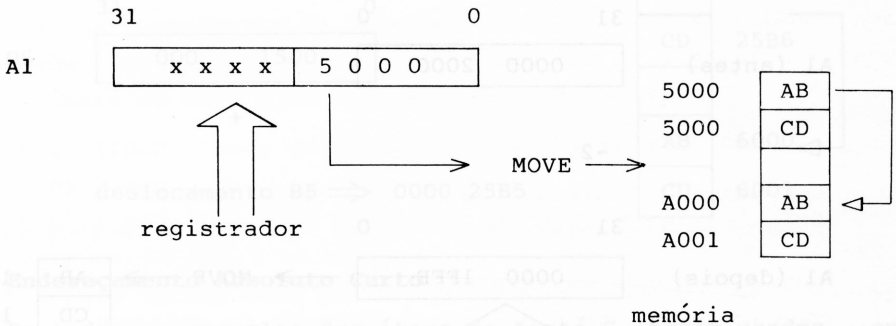
Move o conteúdo do registrador A1 para o endereço absoluto curto $5FØØ.



C - Registro de Endereço Indireto: (An)

Exemplo em assembler: MOVE (A1), $AØØØ

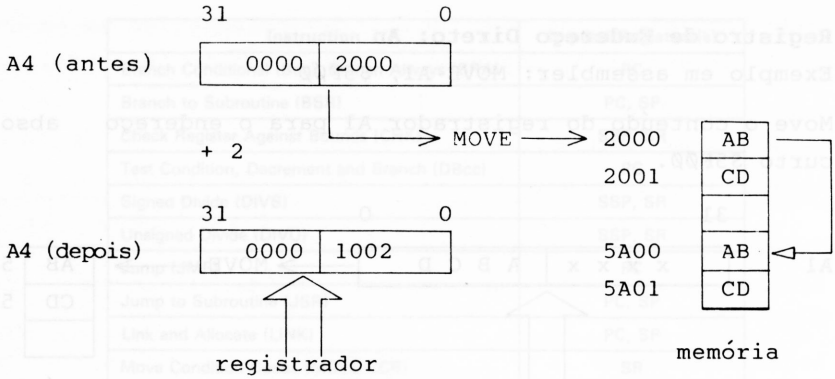
Move o conteúdo do endereço apontado pelo registro A1, para o endereço absoluto curto AØØØ, e endereço A1+1 para o endereço absoluto +1.



D - Registrador de Endereço Indireto Pós-incrementado: (An)+

Exemplo em assembler: MOVE (A4)+, $5AØØ

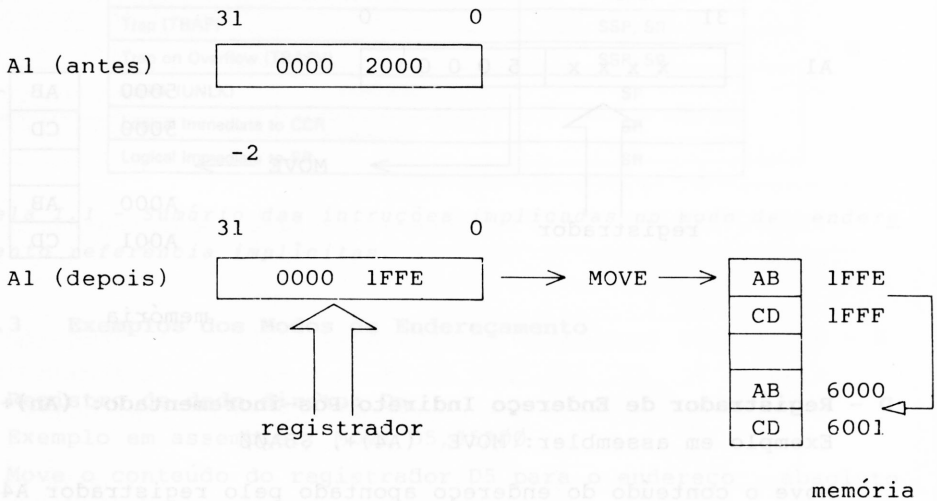
Move o conteúdo do endereço apontado pelo registrador A4, para o endereço 5AØØ. Em seguida, incrementa o conteúdo de A4 em dois bits.



E - Registro de Endereço Indireto Pré-decrementado: -(An)

Exemplo em assembler: MOVE - (A1), $6000

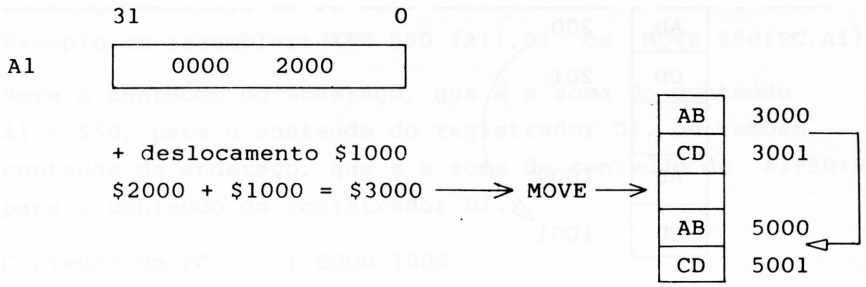
Decrementa o conteúdo do registrador A1 em 2 bits e em seguida, movimenta o conteúdo do endereço apontado pelo registrador A1, para o conteúdo do endereço absoluto curto declarado, ou seja, 6000.



F - Registro de Endereço Indireto com Deslocamento: d16 (An)

Exemplo em assembler: MOVE $1000 (A1), $5000

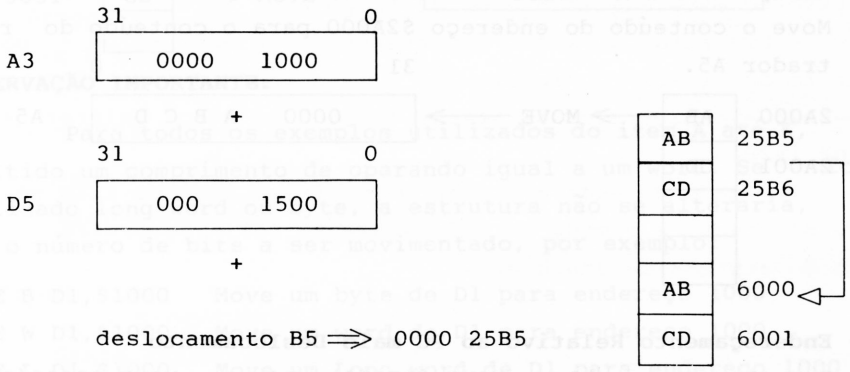
Move o conteúdo do endereço formado pelo conteúdo do registrador A1 + $1000, para o endereço absoluto curto $5000.



G - Registro de Endereço Indireto com Deslocamento e Index: d8 (An,Ri)

Exemplo em assembler: MOVE $B5 (A3,D5), $6000

Move o conteúdo do endereço formado pelo conteúdo do registrador A3 + registrador D5 + o deslocamento de 8 bits $B5, para o endereço absoluto curto $6000.



H - Endereçamento Absoluto Curto

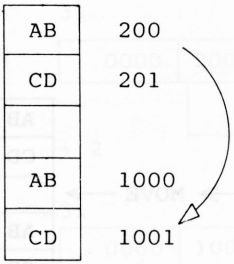
Em todos os exemplos dos ítens de A até G, foram usados endereços absoluto curto.

Este tipo de endereçamento, quando declarado de 0000 até 7FFF, refere-se aos endereços entre 0000 0000 até 0000 7FFF, e quando for declarado de 8000 a FFFF, refere-se aos endereços FFF 8000 até FFFF FFFF.

Isto foi construído para permitir ao usuário ter acesso no começo e fim da memória.

Exemplo em assembler: MOVE $200, $1000

Move o conteúdo do endereço $200 para o endereço $1000.



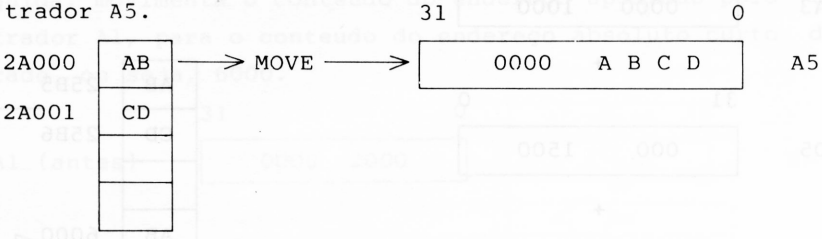
I - Endereçamento Absoluto Longo

Em todos os endereçamentos do item Ⓐ até Ⓖ, poderia ser utilizado o endereçamento longo.

Este endereçamento contém um endereço completo, ou seja, de 32 bits.

Exemplo em assembler: MOVE $2A000, A5

Move o conteúdo do endereço $2A000 para o conteúdo do registrador A5.

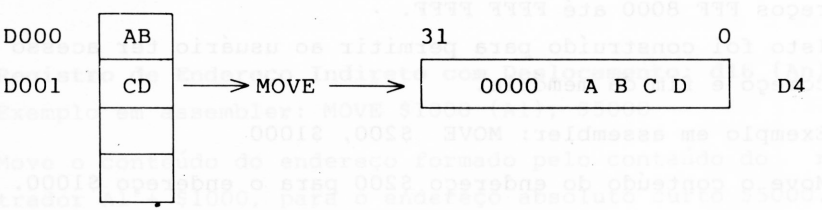


J - Endereçamento Relativo ao PC mais Deslocamento

Exemplo em assembler: MOVE ($5000), D4 ou MOVE $5000(PC),D4

Move o conteúdo do endereço $5000 para o registrador D4, ou como no outro exemplo, move o conteúdo do endereço $5000 mais o conteúdo do PC para o registrador D4.

Conteúdo do PC : 0000 8000
Deslocamento : 0000 5000
Endereço Final : 0000 D000



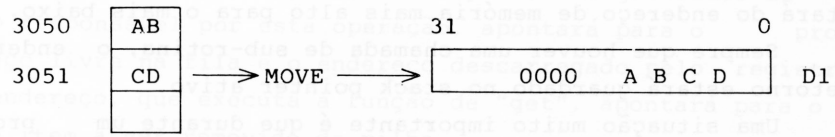
K - Endereço Relativo ao PC mais Deslocamento e mais o Index

Exemplo em assembler: MOVE $50 (A1),D1 ou MOVE $50(PC,A1),D1

Move o conteúdo do endereço, que é a soma do conteúdo de A1 + $50, para o conteúdo do registrador D1, ou também o conteúdo do endereço, que é a soma do conteúdo de A1+50+PC, para o conteúdo do registrador D1.

Conteúdo do PC : 0000 1000
Registrador A1 : 0000 2000
Deslocamento 8 bits: 0000 0050

Endereço Efetivo : 0000 3050



OBSERVAÇÃO IMPORTANTE:

Para todos os exemplos utilizados do item A até K, foi admitido um comprimento de operando igual a um word. Se fosse utilizado long word ou byte, a estrutura não se alteraria, apenas o número de bits a ser movimentado, por exemplo:

```
MOVE B D1,$1000   Move um byte de D1 para endereço 1000
MOVE W D1,$1000   Move um word de D1 para endereço 1000
MOVE L D1,$1000   Move um Long word de D1 para endereço 1000
```

Para fechar esta parte sobre endereçamento, temos na tabela I.2, o sumário dos tipos de endereçamentos possíveis.

Addressing Mode	Mode	Register
Data Register Direct	000	Register Number
Address Register Direct	001	Register Number
Address Register Indirect	010	Register Number
Address Register Indirect with Postincrement	011	Register Number
Address Register Indirect with Predecrement	100	Register Number
Address Register Indirect with Displacement	101	Register Number
Address Register Indirect with Index	110	Register Number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate	111	100

Tabela I.2 - Sumário dos tipos de endereçamento.

I.3 Sistema de Stack Pointer

O registrador de endereço de número 8 (A7) é o Stack Pointer do sistema. O stack pointer do sistema pode ser tanto o SSP (Stack Pointer no modo supervisor) como o USP (Stack Pointer no modo usuário), depende apenas do estado lógico em que se encontra o bit S do registrador de status. Quando os bits do registrador de status indicar o modo supervisor, o SSP está ativo como stack pointer do sistema em caso contrário, ou seja, quando o bit do registrador de status indicar o modo usuário, então será a vez do USP estar ativo como stack pointer do sistema.

Como na maioria dos microprocessadores, o stack pointer apontará do endereço de memória mais alto para o mais baixo.

Sempre que houver uma chamada de sub-rotina, o endereço de retorno estará guardado no stack pointer ativo.

Uma situação muito importante é que durante um processamento de "traps" e "interrupts", o conteúdo do contador de programa e registrador de status ficarão no stack pointer supervisor (SSP).

O modo de endereçamento -(SP) criará um novo item no sistema de stack, e o modo de endereçamento (SP)+ destruirá um item do sistema de stack ativo.

I.4 Queues

O programador pode implementar as filas (Queues) com o modo de endereçamento indireto, via registrador, usando o sistema de "postincrement" ou "predecrement", ou seja, com pós-incremento ou pré-decremento. O usuário pode implementar usando dois registradores de endereço (A0 até A6), que serão preenchidos com memória de endereço mais alto até ao de mais baixo endereço, ou vice-versa. O conceito de filas é que os dados entrarão por um lado e sairão pelo outro, com isso dois registradores serão usados: um para apagar as informações e o outro para enviar as mesmas.

Na figura I.10, temos uma idéia de como formar as filas.

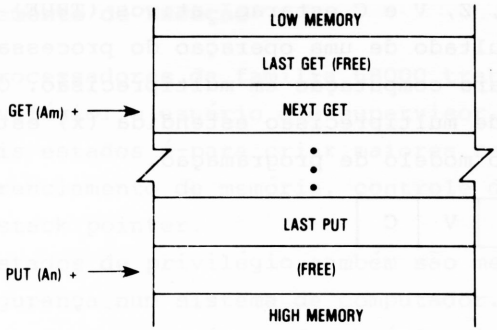


Figura I.10 - Desenho básico das filas (Queues).

Depois de uma operação de "put", o registrador de endereço responsável por esta operação, apontará para o próximo espaço livre na fila e o endereço descarregado pelo registrador de endereço, que executa a função de "get", apontará para o próximo item a ser removido da pilha.

Pilhas que se desenvolverem de um endereço mais alto para outros mais baixo na memória, serão implementados com:

(An) - colocar dados na fila

(Am) - recolher dados da fila.

I.5 Flags

Nesta seção, vamos discutir como os códigos de condições (flags) são utilizados, o significado de cada bit e como são computados.

I.5.1 Registrador de Código de Condições

O registrador de código de condições é uma parte do registrador de status e contém cinco bits:

N - negativo

Z - zero

V - Over Flow

C - carry

X - Extend

Os bits N, Z, V e C estarão ativos (TRUE) quando eles refletirem o resultado de uma operação do processador. O bit x é um operando para computação em multiprecisão. O carry bit (C) e o operando de multiprecisão estendida (x) estão separados para simplificar o modelo de programação.

X	N	Z	V	C

Onde:

N(Negative) : O bit (N) será "setado" se o bit mais significativo do resultado for = 1. Em caso contrário será "resetado".

Z(Zero) : O bit (Z) será setado se o resultado da operação for igual a zero. Em caso contrário será resetado.

V(Overflow) : O bit (V) será setado se no resultado de uma operação aritmética houver um overflow (estouro). Em caso contrário será resetado.

C(Carry) : O bit (C) será setado em duas situações diferentes. Uma quando houver um carry (vai um) fora do bit mais significativo, e a outra se houver um "empresta um" de uma operação de subtração anterior. Em caso contrário será resetado.

X(Extend) : Transparente para movimento de dados. Quando for afetado por uma operação aritmética, ele será setado da mesma forma como o bit carry (C).

No próximo capítulo, será usada uma notação para representar os resultados no registrador de código de condição. A notação será a seguinte:

*	afetado pelo resultado da operação
-	não é afetado pelo resultado da operação
0	resetado
1	setado
U	indefinido depois da operação

I.6 Processamento de Exceção

Os processadores da família 68000 trabalham com dois estados de privilégio: o usuário e o supervisor. O motivo da existência de tais estados é para criar maiores recursos no que se refere ao gerenciamento de memória, controle de acesso e também controle de stack pointer.

Os estados de privilégio também são mecanismos para gerar maior segurança num sistema de computador.

O acesso a certas áreas da memória, instruções privilegiadas, etc são fáceis de controlar, usando esta técnica de estados de previlégio.

I.6.1 Estado Supervisor

Este estado é o de maior privilégio. Ele é determinado pelo bit "S" do registrador de status. Se o bit S = 1, o processador está no modo supervisor, caso contrário, S = Ø, estará no modo usuário.

Todos os tipos de instruções endereçamentos, acessos e processamentos são permitidos no modo supervisor.

Durante qualquer processamento de exceção, o processador irá para o modo supervisor e será utilizado o stack pointer do supervisor.

I.6.2 Estado Usuário

Este estado tem um nível inferior de privilégio em relação ao estado supervisor (ou modo supervisor como chamam alguns escritores).

A maioria das instruções são executadas tanto no modo supervisor como usuário, mas há algumas que têm uma grande importância a nível de sistema, foram atribuídas uma posição de alto privilégio, podendo ser executadas apenas no modo supervisor.

Por exemplo, quando estamos operando no modo usuário, não é permitida a execução de instruções como reset, stop e outras. Para assegurar que um programa no modo usuário não entre no modo supervisor, e a partir daí vá alterar informações não permitidas, a instrução que determina a mudança de estado é uma

instrução privilegiada, ou seja, só e possível ser executada no modo supervisor.

I.6.3 Troca de Estados de Privilégio

A partir do instante em que o programa está rodando no modo usuário, somente um processamento de exceção permitirá a troca de estado. No instante em que o processador entra no processamento de exceção, os registradores, bem como o registrador de status, são salvos na memória. Neste instante, o bit "S" é colocado no modo supervisor. Quando o processador retornar do processamento de exceção, o mesmo estará no modo supervisor.

A mudança do modo supervisor para o usuário é bem mais simples. Esta operação pode ser realizada por quatro instruções: Retorno de Exceção (RTE), mover para o registrador de status (Move Word to SR), AND imediatamente com o registrador de status (ANDI TO SR) e por fim uma lógica "OR EXCLUSIVE" imediatamente com o registrador de status (EORI TO SR). A instrução RTE busca um "novo" registro de status, por assim dizer, e o contador de programa no stack do supervisor, sendo assim pode-se determinar qual nível lógico que terá no bit "S" (previamente estipulado).

I.6.4 Processamentos de Exceção

O processamento de exceção é caracterizado por uma interrupção ocasionada por diferentes situações, a saber:

1 - Reset
2 - Traps
3 - Instruções ilegais ou não implementadas
4 - Violação de privilégio
5 - Erro de comunicação
6 - Erro de endereçamento

No processamento de exceção, duas coisas são muito importantes e devem ser bem analisadas:

- Tabela de vetor de interrupção
- Quadro do stack pointer formado.

I.6.4.1 Tabela de Vetor de Interrupção

A tabela dos vetores de interrupção é a de localização de memória onde o processador busca o endereço da rotina, a qual pertence o processamento de exceção em curso. O vetor é formado por dois words de comprimento, exceção deve ser feita ao reset, que contém quatro words como pode ser visto na tabela I.3.

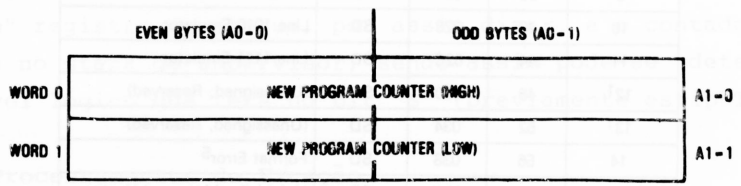
Vector Number(s)	Address		Space[6]	Assignment
	Dec	Hex		
0	0	000	SP	Reset: Initial SSP[2]
1	4	004	SP	Reset: Initial PC[2]
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12[1]	48	030	SD	(Unassigned, Reserved)
13[1]	52	034	SD	(Unassigned, Reserved)
14	56	038	SD	Format Error[5]
15	60	03C	SD	Uninitialized Interrupt Vector
16-23[1]	64	040	SD	(Unassigned, Reserved)
	92	05C		—
24	96	060	SD	Spurious Interrupt[3]
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors[4]
	188	0BC		—
48-63[1]	192	0C0	SD	(Unassigned, Reserved)
	255	0FF		—
64-255	256	100	SD	User Interrupt Vectors
	1020	3FC		—

Tabela I.3 - Tabela de Vetores.

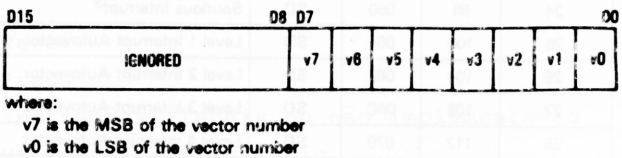
O número dos vetores é formado internamente ou externamente, dependendo da causa do processamento de exceção. No caso, por exemplo de interrupção, durante o ciclo de reconhecimento deste, o periférico que gerou fornece um vetor com 8 bits através da via de dados (DATA BUS).

O processador forma o offset do vetor através do deslocamento à esquerda do número do vetor em dois bits, e preenche de zeros os bits restantes mais significativos. Sendo assim, teremos os 32 bits para formar o endereço. No caso do 68000 e 68008, este offset é usado como endereço absoluto para o processador obter o vetor. No caso do 68010 e 68012 a este offset é somado aos 32 bits do VBK, registrador de base de vetor, para então obter o endereço real do vetor. Isto se deve ao fato de que a tabela de vetores do 68010 e 68012 é flutuante, ou seja, o programador pode colocá-la em qualquer posição da memória.

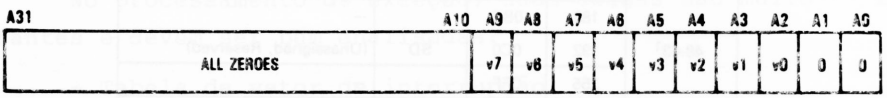
A figura I.11 ilustra melhor os tipos de vetores de exceção.



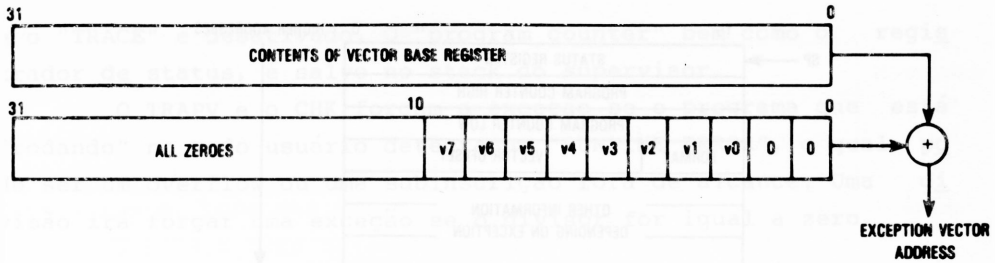
I.11a - Exception Vector Format.



I.11b - Peripheral Vector Number Format.



I.11c - Address Translated from 8-Bit Vector Number (MC68000/MC68008).



I.11d - Exception Vector Addres Calculation (MC68010/MC68012).

Figura I.11 - Formato dos Vetores.

I.6.4.2 Quadro do Stack Pointer

Dependendo do tipo de processamento de exceção que seja requisitado e também dependendo do processador (68000, 68008, 68010 e 68012), será formado um quadro especial de stack. Este quadro é formado para preservar algumas informações importantes tais como, Registrador de status, contador de programa, registradores e etc.

Os processadores 68000 e 68008 têm um quadro para stack de exceção chamado de Grupo 1 e Grupo 2, como mostra a figura I.12.

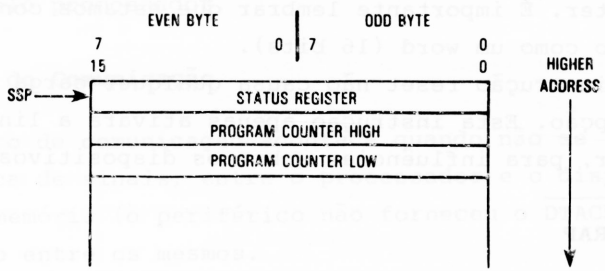
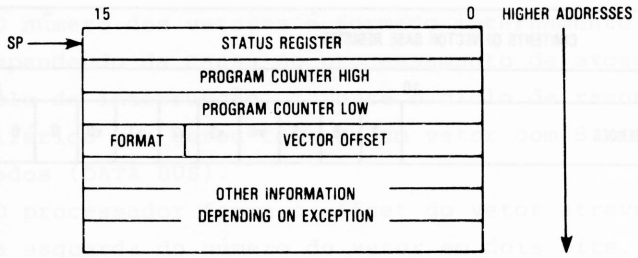


Figura I.12 - Quadro do Stack Grupo 1 e 2 (68000 e 68008).

Os processadores 68010 e 68012 têm um quadro onde o número de informações depende do tipo de exceção. A figura I.13 mostra o quadro geral. Nos próximos parágrafos, vamos observar separadamente outras formações de stack, dependendo do processamento de exceção.



Format Code	Stacked Information
0000	Short Format (4 Words)
1000	Long Format (29 Words)
All Others	Unassigned, Reserved

Figura I.13 - Quadro Geral de Stack Para o 68010 e 68012.

I.6.4.3 Reset

A operação de reset é considerada o mais alto nível de interrupção.

Quando é forçado um reset por hardware, (ver as considerações no volume I), o processador lê o primeiro e segundo endereços da memória e o considera como endereço do stack pointer do supervisor. Em seguida, recolhe o terceiro e quarto endereços, transformando o conteúdo dos mesmos em endereço inicial do programa counter. É importante lembrar que estamos considerando cada endereço como um word (16 bits).

A instrução reset não causa qualquer carga do vetor reset de interrupção. Esta instrução apenas ativará a linha RESET do processador, para influências sobre os dispositivos periféricos.

I.6.4.4 TRAP

Os "traps" são processamentos de exceção causados por instruções. Algumas instruções são usadas especificamente para gerar os "traps". Uma instrução trap irá forçar uma exceção. É utilizada para implementar chamadas no sistema, quando em programas do usuário.

Quando o processador encontrar uma instrução TRAP, iniciará então o processamento de exceção a partir do endereço do vetor da tabela I.1, vetores 5 ou 6, dependendo do tipo do "trap". O registrador de status é copiado, o estado supervisor é ativado

e o "TRACE" é desativado. O "program counter" bem como o registrador de status, é salvo no stack do supervisor.

O TRAPV e o CHK forçam a exceção se o programa que está "rodando" no modo usuário detecta um "RUNTIME ERROR", o qual pode ser um overflow ou uma subinscrição fora de alcance. Uma divisão irá forçar uma exceção se o divisor for igual a zero.

I.6.4.5 Instruções Ilegais

São consideradas instruções ilegais aqueles words em códigos Hexadecimais que não estão definidos no set de instruções da família 68000. Se durante a execução de instruções foi encontrada uma instrução ilegal, ocorrerá um processamento de exceção. Algumas instruções ilegais são reservadas para o próprio fabricante (para futuras ampliações, talvez) e outras para o próprio usuário. Por exemplo os números $4AFA e $4AFB são reservados para a Motorola e o $4AFC para o próprio usuário construir sua "própria" instrução.

Neste como, em todos os processamentos de exceção são preservados registrador de status, contador de programa, etc. O stack formado é o do GRUPO 2 que será salvo no stack do modo supervisor, e por fim, o vetor de exceção para instrução ilegal é acessado pelo processador.

I.6.4.6 Erro de Comunicação

O erro de comunicação acontece quando não se processa a devida troca de sinais, entre o processador e o dispositivo periférico ou memória (o periférico não forneceu o $\overline{\text{DTACK}}$), durante a comunicação entre os mesmos.

O procedimento do processamento de exceção é o mesmo como para os casos anteriores, sendo que algumas informações são acrescidas.

No caso do 68000 e 68008, o stack formado é o da figura I.14, enquanto o stack do 68010 e 68012 será o da figura I.15.

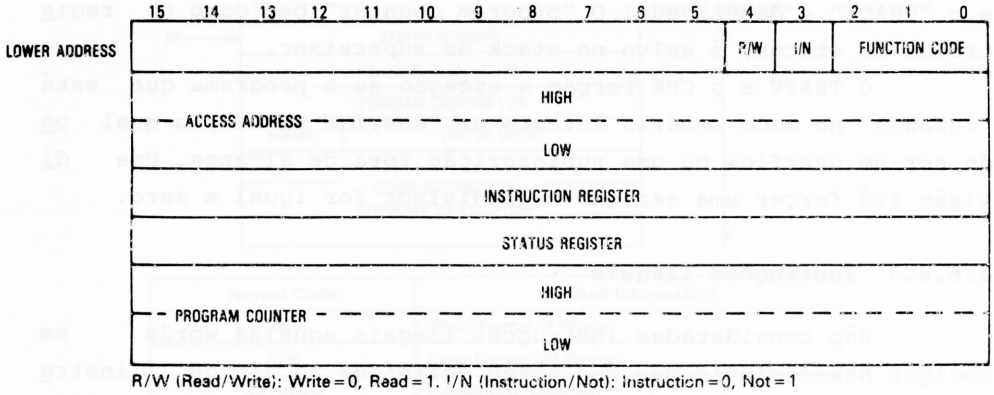


Figura I.14 - Stack para os erros de comunicação e endereçamento dos processadores 68000/68008.

Podemos notar que o stack do 68010 e 68012 é mais completo, justamente devido à implementação da memória e máquina virtual.

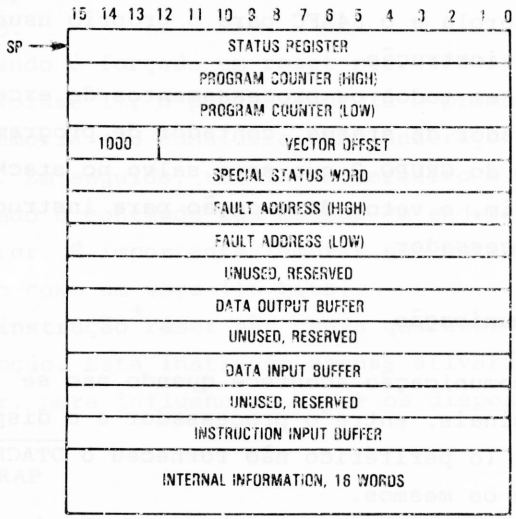


Figura I.15 - Stack para os erros de comunicação e endereçamento dos processadores 68010 e 68012.

Se um erro de comunicação acontecer dentro de um processamento de exceção devido a um erro de comunicação, ficará caracterizado um erro fatal. Sendo assim o processador irá automaticamente para um estado de "Halt", cessando assim todo e qualquer processamento. Somente um "Reset" por hardware o fará retornar ao processamento normal.

I.6.4.7 Erro de Endereçamento

O processamento de exceção devido a um erro de endereçamento, ocorre quando o processador tentar ler ou escrever num endereço ímpar. O efeito é idêntico a um erro de comunicação. Sendo assim, tudo que foi discutido sobre o erro de comunicação é válido para o erro de endereçamento.

I.7 Pipelining e Loop

Nos processadores da família 68000, existe uma característica que se chama "Pipelining". O pipelining envolve a busca e execução de forma concorrente. Este procedimento dá ao processador maior velocidade na execução das instruções. Os processadores tradicionais fazem a busca das instruções na memória de forma serial, ou seja, uma a uma.

Nos processadores da família 68000, são buscadas duas words por vez da memória, assim que a decodificação é iniciada outra palavra (word) é procurada na memória. Deste modo, ao começar a execução da instrução, duas palavras foram trazidas da memória. Se houver uma instrução de desvio, a palavra que segue é ignorada.

Nos processadores 68010 e 68012, a idéia de pipelining foi melhorada. Como é feito uma busca antecipada de duas palavras (words), os processadores 68010 e 68012 verificam se uma das instruções é uma primitiva DBcc. Em determinados casos, o processador pode executar a operação "loop", sem a necessidade de buscar esta instrução novamente na memória. A instrução DBcc é uma típica instrução de loop, pois a mesma é composta de três partes distintas. A primeira é um contador, a segunda um teste para ver se a condicional desejada foi satisfeita e a terceira parte é um deslocador (desvio), para o processador saltar para aquele endereço determinado pelo desvio, se a condicional não foi ainda satisfeita.

O programa da figura I.16 mostra um típico programa de loop executado pelo 68010 e 68012.

```
        LEA ORIGEM, A0          carrega o apontador A0
        LEA DESTIN, A1          carrega o apontador A1
        MOVE.W #COMPRIM,D0      carrega o contador D0
LOOP    MOVE.W (A0)+,(A1)+      Loop para remover o bloco dos
                                dados
        DBEQ D0, Loop           Parar se o dado no Reg D0  é
                                igual a zero.
```

Figura I.16 - Exemplo de programa no modo Loop.

Há um grupo de instruções que permitem executar a função "loop" no 68010 e 68012 e as mesmas estão listadas na tabela I.4. Estas instruções podem ser usadas em 3 modos de endereçamento indireto, para formar instruções de um word em loop; (An), (An)+ e -(An).

Opcodes	Applicable Addressing Modes	
MOVE [BWL]	(Ay) to (Ax) (Ay) to (Ax)+ (Ay) to -(Ax) (Ay)+ to (Ax) (Ay)+ to (Ax)+ (Ay)+ to -(Ax)	-(Ay) to (Ax) -(Ay) to (Ax)+ -(Ay) to -(Ax) Ry to (Ax) Ry to (Ax)+
ADD [BWL] AND [BWL] CMP [BWL] OR [BWL] SUB [BWL]	(Ay) to Dx (Ay)+ to Dx -(Ay) to Dx	
ADDA [WL] CMPA [WL] SUBA [WL]	(Ay) to Ax -(Ay) to Ax (Ay)+ to Ax	
ADD [BWL] AND [BWL] EOR [BWL] OR [BWL] SUB [BWL]	Dx to (Ay) Dx to (Ay)+ Dx to -(Ay)	

Opcodes	Applicable Addressing Modes
ABCD [B] ADDX [BWL] SBCD [B] SUBX [BWL]	-(Ay) to -(Ax)
CMP [BWL]	(Ay)+ to (Ax)+
CLR [BWL] NEG [BWL] NEGX [BWL] NOT [BWL] TST [BWL] NBCD [B]	(Ay) (Ay)+ -(Ay)
ASL [W] ASR [W] LSL [W] LSR [W] ROL [W] ROR [W] ROXL [W] ROXR [W]	(Ay) by #1 (Ay)+ by #1 -(Ay) by #1

NOTE
[B, W, or L] indicate an operand size of byte, word, or long word.

Tabela I.4 - Tabela das instruções que permitem loop no 68010 e 68012.

CAP. 1

ESTUDO DAS INSTRUÇÕES

Neste capítulo, faremos um estudo profundo nas instruções dos processadores 68000, 68008, 68010 e 68012.

Para um melhor entendimento deste material, vamos inicialmente fazer uma análise das instruções subdivididas em grupos básicos e em seguida analisá-las separadamente.

Como o estudo exige muitas explicações, torna-se necessário adotarmos alguns símbolos e definições para facilitar a análise.

1.1 Notações Utilizadas

OPERANDO		SIGNIFICADO
An	=	Registrador de endereço
Dn	=	Registrador de dados
Rn	=	Registrador Genérico
Pc	=	Program counter
SR	=	Status Register
CCR	=	Registrador de códigos de condições (flags)
SSP	=	Stack supervisor
USP	=	Stack usuário
SP	=	Stack pointer
X	=	Operando de extensão (flag)
N	=	Flag (negativo)
Z	=	Flag de zero
V	=	Flag de overflow
C	=	Flag de carry
Immediate Data	=	Operando imediato
d	=	Deslocamento
SOURCE	=	Operando origem
Destination	=	Operando destino
Vector	=	Vetor de exceção
< ea >	=	Endereço efetivo.

SUBCAMPOS		SIGNIFICADO
<bit> of <operand>	=	bit de um operando
(<operand>)	=	operando de uma instrução
<operand> 10	=	operando de uma instrução em decimal
(<address register>)	=	operando indireto que indica uma posição de memória
-(<address register>)	=	operando indireto que indica uma posição de memória
(<address register>)+	=	operando indireto que indica uma posição de memória
#xxx or # <DATA>	=	dado imediato

SÍMBOLOS		SIGNIFICADO
$\longrightarrow$	=	operando vai da esquerda para direita
$\longleftrightarrow$	=	operandos são trocados
+	=	soma
-	=	subtração
*	=	multiplicação
/	=	divisão
$\wedge$	=	lógica "E"
$\vee$	=	lógica "OU"
$\oplus$	=	lógica "OU EXCLUSIVO"
<	=	teste relacional (menor do que)
>	=	teste relacional (maior do que)
shifted by	=	deslocamento de operando
rotated by	=	rotação de um operando
~	=	complemento do operando

1.2 Conjunto das Instruções em Grupos

O conjunto de instruções da família 68000 forma uma ferramenta para desempenhar as seguintes operações básicas:

1 - Movimento de dados
2 - Aritmética com operações com números inteiros
3 - Operações lógicas
4 - Deslocamento e rotação
5 - Manipulaçãc de bits
6 - Manipulação de campos

7 - Aritmética BCD
8 - Controle de programa
9 - Controle de sistema
10 - Comunicação para multiprocessamento

1.2.1 Operação com Movimento de Dados

Este conjunto de instruções é o mais simples e está baseado na instrução "MOVE".

A instrução de movimento de dados permite transferência entre registradores, registradores para memória, memória para registrador e memória para memória. É possível transferir byte, word e long word.

Além das instruções MOVE, há algumas instruções mais, tais como: EXG (trocar conteúdo de registradores), LEA (carregar um endereço efetivo), etc. Na tabela 1.1, temos o resumo das instruções de movimento.

Instruction	Operand Syntax	Operand Size	Operation
EXG	Rn, Rn	32	Rn ↔ Rn
LEA	<ea>, An	32	<ea> → An
LINK	An, #<d>	16, 32	SP − 4 → SP; An → (SP); SP → An; SP + d → SP
MOVE MOVEA	<ea>, <ea> <ea>, An	8, 16, 32 16, 32 → 32	Source → Destination
MOVEM	list, <ea> <ea>, list	16, 32 16, 32 → 32	Listed Registers → Destination Source → Listed Registers
MOVEP	Dn, (d_{16},An) (d_{16},An), Dn	16, 32	Dn[31:24] → (An + d), Dn[23:16] → (An + d + 2); Dn[15:8] → (An + d + 4); Dn[7:0] → (An + d + 6) (An + d) → Dn[31:24]; (An + d + 2) → Dn[23:16]; (An + d + 4) → Dn[15:8]; (An + d + 3) → Dn[7:0]
MOVEQ	#<data>, Dn	8 → 32	Immediate Data → Destination
PEA	<ea>	32	SP − 4 → (SP); <ea> → (SP)
UNLK	An	32	An → SP; (SP) → An; SP + 4 → SP

Tabela 1.1 - Conjunto de Instrução de Movimento.

1.2.2 Operação Aritmética Interna

Na operação aritmética, estão inclusos as quatro operações básicas: soma, subtração, divisão e multiplicação. Também faz parte a função "comparação", clear e negação.

As instruções de soma, comparação e subtração são permitidas tanto em operações com endereços ou com dados, sendo es

tes de qualquer tamanho. As operações com endereços são limitadas aos comprimentos chamados "LEGAIS" tais como: word, long word (16, 32 bits). As instruções de clear e negação podem ser de qualquer tamanho de dados.

Nas operações de multiplicação e divisão, são possíveis dados com ou sem sinal.

Aritmética de multiprecisão pode ser possível utilizando as instruções estendidas tais como: ADDX (soma estendida), SUBX (subtração estendida), etc.

A tabela 1.2 mostra o resumo das instruções aritméticas.

Instruction	Operand Syntax	Operand Size	Operation
ADD ADDA	Dn, <ea> <ea>, Dn <ea>, An	8, 16, 32 8, 16, 32 16, 32	Source + Destination → Destination
ADDI ADDQ	#<data>, <ea> #<data>, <ea>	8, 16, 32 8, 16, 32	Immediate Data + Destination → Destination
ADDX	Dn, Dn –(An), –(An)	8, 16, 32 8, 16, 32	Source + Destination + X → Destination
CLR	<ea>	8, 16, 32	0 → Destination
CMP CMPA	<ea>, Dn <ea>, An	8, 16, 32 16, 32	Destination – Source
CMPI	#<data>, <ea>	8, 16, 32	Destination – Immediate Data
CMPM	(An)+, (An)+	8, 16, 32	Destination – Source
DIVS/DIVU	<ea>, Dn	32/16 → 16:16	Destination/Source → Destination (Signed or Unsigned)
EXT	Dn Dn	8 → 16 16 → 32	Sign Extended Destination → Destination
MULS/MULU	<ea>, Dn	16 × 16 → 32	Source*Destination → Destination (Signed or Unsigned)
NEG	<ea>	8, 16, 32	0 – Destination → Destination
NEGX	<ea>	8, 16, 32	0 – Destination – X → Destination
SUB SUBA	<ea>, Dn Dn, <ea> <ea>, An	8, 16, 32 8, 16, 32 16, 32	Destination – Source → Destination
SUBI SUBQ	#<data>, <ea> #<data>, <ea>	8, 16, 32 8, 16, 32	Destination – Immediate Data → Destination
SUBX	Dn, Dn –(An), –(An)	8, 16, 32 8, 16, 32	Destination – Source – X → Destination

Tabela 1.2 - Conjunto de instruções aritméticas.

1.2.3 Operação Lógica

As operações lógicas AND, OR, EOR ou NOT são possíveis em todos os tamanhos de dados (8, 16 e 32 bits). A instrução

"TST" é uma comparação aritmética de operando com valor em zero, o qual será refletido nos códigos de condição (flags).

A tabela 1.3 mostra o resumo destas instruções.

Instruction	Operand Syntax	Operand Size	Operation
AND	<ea>, Dn Dn, <ea>	8, 16, 32 8, 16, 32	Source Λ Destination → Destination
ANDI	#<data>, <ea>	8, 16, 32	Immediate Data Λ Destination → Destination
EOR	Dn, <ea>	8, 16, 32	Source ⊕ Destination → Destination
EORI	#<data>, <ea>	8, 16, 32	Immediate Data ⊕ Destination → Destination
NOT	<ea>	8, 16, 32	~ Destination → Destination
OR	<ea>, Dn Dn, <ea>	8, 16, 32 8, 16, 32	Source V Destination → Destination
ORI	#<data>, <ea>	8, 16, 32	Immediate Data V Destination → Destination
TST	<ea>	8, 16, 32	Source – 0 to Set Condition Codes

Tabela 1.3 - Conjunto de instruções lógicas.

1.2.4 Operação de Deslocamento e Rotação

A operação de deslocamento pode ser feita em qualquer direção (direita ou esquerda).

As instruções de deslocamento e rotação podem ser realizadas tanto nos registradores como na memória.

A tabela 1.4 mostra o resumo destas instruções.

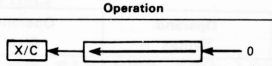
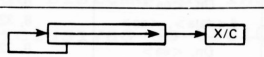
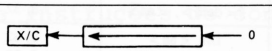
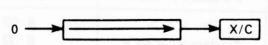
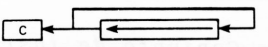
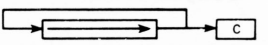
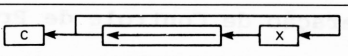
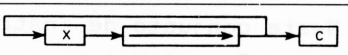
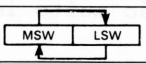
Instruction	Operand Syntax	Operand Size	Operation
ASL	Dn, Dn #<data>, Dn <ea>	8, 16, 32 8, 16, 32 16	
ASR	Dn, Dn #<data>, Dn <ea>	8, 16, 32 8, 16, 32 16	
LSL	Dn, Dn #<data>, Dn <ea>	8, 16, 32 8, 16, 32 16	
LSR	Dn, Dn #<data>, Dn <ea>	8, 16, 32 8, 16, 32 16	
ROL	Dn, Dn #<data>, Dn <ea>	8, 16, 32 8, 16, 32 16	
ROR	Dn, Dn #<data>, Dn <ea>	8, 16, 32 8, 16, 32 16	
ROXL	Dn, Dn #<data>, Dn <ea>	8, 16, 32 8, 16, 32 16	
ROXR	Dn, Dn #<data>, Dn <ea>	8, 16, 32 8, 16, 32 16	
SWAP	Dn	32	

Tabela 1.4 - Conjunto de instruções de deslocamento e rotação.

1.2.5 Operação em B.C.D.

Operação aritmética de multiprecisão em BCD é possível, utilizando as instruções: ABCD (soma decimal com extensão), SBCD (subtração decimal com extensão) e NBCD (negação decimal com extensão). A tabela 1.5 mostra estas instruções.

Instruction	Operand Syntax	Operand Size	Operation
ABCD	Dn, Dn -(An), -(An)	8 8	$Source_{10} + Destination_{10} + X \rightarrow Destination$
NBCD	<ea>	8	$0 - Destination_{10} - X \rightarrow Destination$
SBCD	Dn, Dn -(An), -(An)	8 8	$Destination_{10} - Source_{10} - X \rightarrow Destination$

Tabela 1.5 - Conjunto de instruções em BCD.

1.2.6 Operação de Manipulação de Bits

A operação de manipulação de bits é possível, utilizando as instruções BTST (testa bit), BSET (seta e testa o bit), BCLR (testa o bit e limpa), BCHG (testa e troca o bit). Todas estas operações são possíveis em registradores ou memória. Nos registradores, o operando será sempre 32 bits, enquanto que na memória será 8 bits. A tabela 1.6 é um resumo destas instruções.

Instruction	Operand Syntax	Operand Size	Operation
BCHG	Dn, <ea> #<data>, <ea>	8, 32 8, 32	~(<Bit Number> of Destination) → Z → Bit of Destination
BCLR	Dn, <ea> #<data>, <ea>	8, 32 8, 32	~(<Bit Number> of Destination) → Z; 0 → Bit of Destination
BSET	Dn, <ea> #<data>, <ea>	8, 32 8, 32	~(<Bit Number> of Destination) → Z; 1 → Bit of Destination
BTST	Dn, <ea> #<data>, <ea>	8, 32 8, 32	~(<Bit Number> of Destination) → Z

Tabela 1.6 - Conjunto de instruções de manipulação de bit.

1.2.7 Operação de Controle de Programa

Neste conjunto de instruções, estão inclusos os "Branch" e "Jumps" que são muito importantes para a elaboração das sub-rotinas.

Para os leitores menos familiarizados com certos termos, os "Branchs" são os que conhecemos normalmente por "CALL" nos processadores de 8 bits. A tabela 1.7 mostra o resumo destas instruções.

Instruction	Operand Syntax	Operand Size	Operation
Conditional			
Bcc	<label>	8, 16	If Condition True, Then PC + d → PC
DBcc	Dn, <label>	16	If Condition False, Then Dn − 1 → Dn If Dn ≠ −1, Then PC + d → PC
Scc	<ea>	8	If Condition True, Then 1's → Destination; Else 0's → Destination
Unconditional			
BRA	<label>	8, 16	PC + d → PC
BSR	<label>	8, 16	SP − 4 → SP; PC → (SP); PC + d → PC
JMP	<ea>	none	Destination → PC
JSR	<ea>	none	SP − 4 → SP; PC → (SP); Destination → PC
NOP	none	none	PC + 2 → PC
Returns			
RTD	#<d>	16	(SP) → PC; SP + 4 + d → SP
RTR	none	none	(SP) → CCR; SP + 2 → SP; (SP) → PC; SP + 4 → SP
RTS	none	none	(SP) → PC; SP + 4 → SP

Tabela 1.7 - Conjunto de instruções Branch e Jump.

1.2.8 Operação de Controle de Sistema

As instruções de controle de sistema são instruções privilegiadas, sendo que algumas são exclusivas do 68010 e 68012. Na tabela 1.8, está o resumo destas instruções de controle.

Instruction	Operand Syntax	Operand Size	Operation
		Privileged	
ANDI	#<data>, SR	16	Immediate Data ∧ SR → SR
EORI	#<data>, SR	16	Immediate Data ⊕ SR → SR
MOVE	<ea>, SR SR, <ea>	16 16	Source → SR SR → Destination
MOVE	USP, An An, USP	32 32	USP → An An → USP
MOVEC	Rc, Rn Rn, Rc	32 32	Rc → Rn Rn → Rc
MOVES	Rn, <ea> <ea>, Rn	8, 16, 32	Rn → Destination Using DFC Source Using SFC → Rn
ORI	#<data>, SR	16	Immediate Data V SR → SR
RESET	none	none	Assert $\overline{\text{RESET}}$ line
RTE	none	none	(SP) → SR; SP + 2 → SP; (SP) → PC; SP + 4 → SP; Restore Stack According to Format
STOP	#<data>	16	Immediate Data → SR; STOP
		Trap Generating	
BKPT	#<data>	none	Execute Breakpoint Acknowledge Bus Cycle; . . . Trap as Illegal Instruction
CHK	<ea>, Dn	16	If Dn < 0 or Dn > (ea), Then CHK Exception
ILLEGAL	none	none	SSP − 2 → SSP; Vector Offset → (SSP); . . . SSP − 4 → SSP; PC → (SSP); SSP − 2 → SSP; SR → (SSP); Illegal Instruction Vector Address → PC
TRAP	#<data>	none	SSP − 2 → SSP; Format and Vector Offset → (SSP); . . . SSP − 4 → SSP; PC → (SSP); SSP − 2 → SSP; SR → (SSP); Vector Address → PC
TRAPV	none	none	If V Then Take Overflow TRAP Exception
		Condition Code Register	
ANDI	#<data>, CCR	8	Immediate Data ∧ CCR → CCR
EORI	#<data>, CCR	8	Immediate Data ⊕ CCR → CCR
MOVE	<ea>, CCR CCR, <ea>	16 16	Source → CCR CCR → Destination
ORI	#<data>, CCR	8	Immediate Data V CCR → CCR

Tabela 1.8 - Conjunto de instruções de controle de sistema.

1.2.9 Operação de Multiprocessamento

É possível executar a comunicação entre os processadores da família 68000, através da instrução TAS que executa um ciclo de READ - MODIFY - WRITE (leitura e escrita modificada). Esta instrução não é divisível, ou seja, durante a sua execução o processador não pode ser interrompido.

A tabela 1.9 mostra o resumo da mesma.

Instruction	Operand Syntax	Operand Size	Operation
TAS	<ea>	8	Destination – 0; Set Condition Codes; 1 → Destination [7]

Figura 1.9 - Operação de Multiprocessamento.

Como podemos sentir até o presente momento, a família dos processadores 68000 têm, além de outras instruções, as mesmas que a grande maioria de processadores possuem com a vantagem de que são mais completas, funcionais e rápidas.

Em seguida, serão analisadas todas as instruções uma a uma para um perfeito entendimento das mesmas.

1.3 Análise Individual das Instruções

Para a descrição individual serão utilizados os símbolos descritos no item 1.1 deste capítulo. A forma em que as instruções serão apresentadas são as seguintes:

Nome da instrução : Neste campo, o nome da instrução.

Resumo de operação : Análise rápida da função que a instrução executa.

Sintaxe do assembler : Formato da instrução quando for construir um programa no editor para posterior compilação com o assembler.

Tamanho do operando : Possibilidades de comprimento do operando, ou seja, byte, word, long word ou nenhum.

Descrição de operação : Análise detalhada da função que a instrução executa.

Formato da instrução : Mostra como fica a instrução depois de transformada em código objeto.

Flags : Neste campo, mostra os flags (ou códigos de condição) que são afetados pela instrução.
Quando o flag é afetado, aparecerá o sinal "*" e quando não for, aparecerá o sinal "I" de ignorar.

A B C D — Soma decimal ou tensão

Resumo : $Fonte_{10} + Destino_{10} + x \Rightarrow Destino_{10}$

Sintaxe em assembler : ABCD Dy,Dx ou ABCD -(Ay), -(Ax)

Tamanho : Byte

Descrição : Soma o conteúdo do operando "Fonte", mais o operando "Destino", mais o flag x, em aritmética BCD, colocando o resultado no operando "Destino".

Formato :

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	Registro Dx ou Ax	1	0	0	0	0	R/M	Registrador Ry ou Ay

Dx, Ax = especifica o registrador origem
Dy, Ay = especifica o registrador destino
Se R/M=Ø indica a transferência de registrador para registrador
Se R/M=1 indica a transferência de memória para memória, no modo de endereçamento pré-decrementado

Flags :

X	N	Z	V	C
*	I	*	I	*

ADD — Soma binária

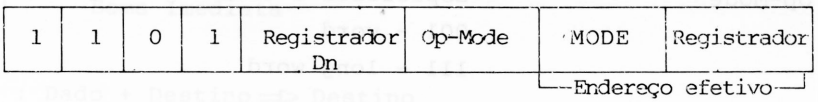
Resumo : Origem + destino → destino

Sintaxe em assembler : ADD <ea>, Dn
ADD Dn, <ea>

Tamanho : Byte, word ou long word

Descrição : Soma o operando "origem" mais o "destino" em aritmética binária, colocando o resultado no operando "destino".

Formato : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



Registrador Dn = pode ser qualquer um dos oito registradores de dados (000 = D0 e 111 = D7).

Op-Mode = dependendo do formato, teremos:

Byte	word	long	Operação
000	001	010	<ea> + Dn → Dn
100	101	110	Dn + <ea> → <ea>

Mode Registrador = Determina geral o tipo de endereçamento.

Flags :

X	N	Z	V	C
*	*	*	*	*

A D D A

Soma endereços

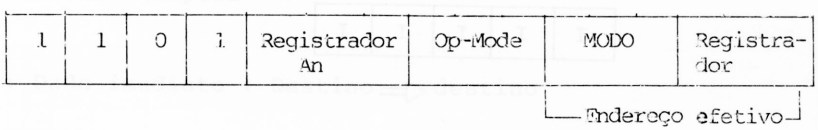
Resumo : Origem + Destino ⇒ Destino

Sintaxe do Assembler: ADDA <ea>, An

Tamanho : word ou long word

Descrição : Soma o operando "Origem" mais o operando "destino" com resultado no "destino", sendo que o operando "origem" é um endereço de memória e o destino é um registrador de endereço.

Formato : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



Registrador An = Pode ser qualquer um dos oito registradores de endereço. Este será sempre o destino.

Op-Mode = Especifica o tamanho da operação
001 - word
111 - long word

Endereço Efetivo = Neste campo (bit0 até bit5), onde deve ser definido o modo e o registrador. Na tabela 1.10, temos a esquematização dos modos e registradores.

Itens	MODO DE ENDEREÇAMENTO	MODO	REGISTRADOR
1)	Dn	000	Dn
2)	An	001	An
3)	(An)	010	An
4)	(An)+	011	An
5)	-(An)	100	An
6)	(d16,An)	101	An
7)	(d8,An,Xn)	110	An
8)	(xxx).W	111	000
9)	(xxx).L	111	001
10)	#<DATA>	111	100
11)	(d16,PC)	111	010
12)	(d8,PC,Xn)	111	011

Tabela 1.10 - Modos de endereçamento com seus respectivos códigos e registradores.

Flags : nenhum é afetado

X	N	Z	V	C
I	I	I	I	I

A D D I — Soma Imediata

Resumo : Dado + Destino ⇒ Destino

Sintaxe do Assembler : ADDI # <DATA>, <ea>

Tamanho : Byte, word e long word

Descrição : Soma um dado com o operando destino e guarda o resultado no destino.

Formato :

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	0	0	0	0	1	1	0	Tamanho	Endereço efetivo Modo	Registrador
word										
long word										

Tamanho ⇒ 00 byte
01 word
10 long word

Endereço efetivo : Válida a tabela 1.10, excluindo os itens 2, 10, 11 e 12.

Flags : Todos são afetados

X	N	Z	V	C
*	*	*	*	*

A D D Q — Soma rápida

Resumo : Dado imediato + Destino ⇒ destino

Sintaxe do Assembler : ADDQ #<data>, <ea>

Tamanho : Byte, word e long word

Descrição : Soma um dado de 8 bits (1 até 8 bits) ao operando destino, sendo o mesmo um endereço efetivo, colocando o resultado no destino. O operando destino pode operar com 8, 16 ou 32 bits.

Formato :

15	14	13	12	11 10 9	8	7 6	5 4 3 2 1 0
0	1	0	1	DADO	Ø	Tamanho	Endereço efetivo Modo Registrador

Dado: Pode assumir valores entre Ø a 7

Tamanho: 00 → byte
01 → word
10 → long word

Endereço efetivo: Válida a tabela 1.10, menos os itens 12, 11 e 10

Flags : Todos os Flags são afetados

X	N	Z	V	C
*	*	*	*	*

A D D X

Soma com extensão

Resumo : Origem + destino + x ⇒ destino

Sintaxe de Assembler: ADDX Dy,Dx ou
ADDX -(Ay), -(Ax)

Tamanho : Byte, word ou long word

Descrição : Soma o operando origem mais o destino e também a extensão, colocando o resultado no operando destino.

Formato :

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	0	1	Registrador Rx	1	Tamanho	0	0	R/M	Registrador Ry

Registrador Rx = Campo para especificar o registrador destino

Registrador Ry = Campo para especificar o registrador origem

Tamanho = 00 → byte; 01 → word; 10 → long word.

R/M = Ø ⟶ indica que os registradores são de dados
1 ⟶ indica que os registradores são de endereço, usando o modo pré-decrementado.

Flags : Todos são afetados

X	N	Z	V	C
*	*	*	*	*

A N D Lógica "e"

Resumo : Origem ^ Destino ⟹ Destino

Sintaxe do Assembler: AND <ea>, Dn
AND Dn, <ea>

Tamanho : Byte, word e long word

Descrição : Executa uma operação lógica "e" entre o operando origem e o destino, colocando o resultado no destino.

Formato :

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	1	0	0	Registrador DN	Op-Mode	Endereço Efetivo Modo Registrador

Registrador Dn = define gual o registrador de dados que será usado.

Op-Mode =

Byte	word	Long	Operação
000	001	010	(ea) ^ Dn ⟶ Dn
100	101	110	Dn ^ (ea) ⟶ (ea)

Endereço efetivo = Usar a tabela 1.11 se a localização especificada for a origem e usar a tabela 1.12, se a localização especificada for o destino.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011

Tabela 1.11

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

Tabela 1.12

Flags : X N Z V C

-	*	*	Ø	Ø

X = não é afetado

Z = depende do resultado da operação. Setado se o resultado for igual à zero.

N = setado se o bit mais significativo for igual a estado lógico 1.

V,C = sempre zerado.

A N D I — Lógica "e" imediato

Resumo : Dado imediato ^ Destino ⇒ Destino

Sintaxe do Assembler : ADDI #<data> , <ea>

Tamanho : Byte, word e long word

Descrição : Faz uma operação lógica "e" entre um dado (constante de 8, 16 ou 32 bits) e um endereço especificado. O resultado ficará no destino.

Formato :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	Tamanho		Endereço Efetivo Modo			Registrador		
word								Byte data							
long word															

Tamanho = 00 : Byte
01 : Word
10 : Long word

Endereço efetivo = Usar a tabela 1.11, menos os três últimos modos de endereçamento.

Flags : Resultado igual a instrução AND.

ANDI TO CCR Lógica e Imediato com os códigos de condições

Resumo : Origem ^ CCR ⟶ CCR

Sintaxe do Assembler : ADDI #<data>, CCR

Tamanho : byte

Descrição : Faz uma operação lógica "e" entre um dado de 8 bits e os sinais (flags) de código de condições e guarda o resultado na parte do registradcr de status, onde estão os códigos de condições.

Formato : 15 8 7 0

0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte de dados							

Flags :

X	N	Z	V	C
*	*	*	*	*

X = zerado se o bit 4 do dado imediato = Ø
N = zerado se o bit 3 do dado imediato = Ø
Z = zerado se o bit 2 do dado imediato = Ø
V = zerado se o bit 1 do dado imediato = Ø
C = zerado se o bit 0 do dado imediato = Ø

ANDI TO SR Lógica "e" imediato com o registrador de status

Resumo : Se estado supervisor, então Origem ^SR⇒SR senão TRAP.

Sintaxe do Assembler : ANDI #<DATA>,SR

Tamanho : Word

Descrição : Faz uma operação lógica "e" entre um dado de 16 bits e o conteúdo do registrador SR. Se o processador estiver no estado supervisor, a operação se dará normalmente. No modo usuário irá gerar um TRAP.

Formato : 15 0

0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0
Dado de 16 bits															

Flags : Idêntico à instrução ANDI to CCR.

ASL.ASR Deslocamento aritmético

Resumo : Destino deslocado por <n contagem> →Destino

Sintaxe do Assembler : Asd Dx,Dy
Asd# <dados>, Dy
Asd <ea>
Onde d pode ser esquerda (L) ou direita (R).

Tamanho : Byte, word ou long word

Descrição : Deslocamento aritmético para a instrução ASL significa preencher de zeros através dos bits menos significativos, durante o deslocamento, enquanto que para a instrução ASR significa preencher com o bit de sinal, através dos bits mais significativos.
A figura 1.1, mostra as duas situações.

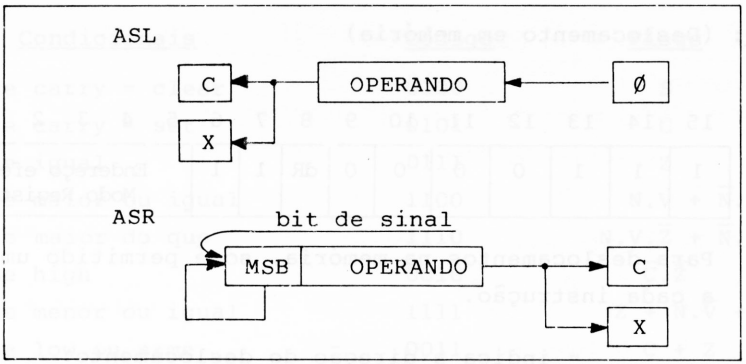


Figura 1.1 - Diagrama das instruções ASL e ASR.

Formato da: (Deslocamento em registradores)
Instrução

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Registrador de contagem	dR	Tamanho	I/R	Ø	Ø	Registrador

Registrador de Contagem : Neste campo, é especificado o número de deslocamento ou o registrador onde tem este número.

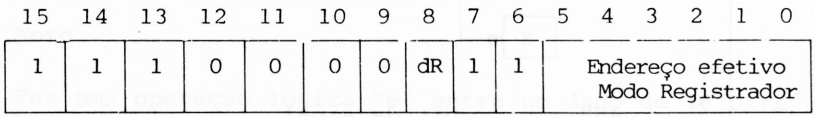
Se I/R = Ø : O número de deslocamento é especificado neste campo diretamente, ou seja, Ø a 7 deslocamentos.

Se I/R = 1 : O número de deslocamento estará no registrador de dados especificado neste campo.

dR : Especifica a direção do deslocamento:
Se dR=Ø deslocamento à direita
Se dR=1 deslocamento à esquerda

Registrador : Especifica qual registrador terá seu conteúdo deslocado.

Formato de instrução: (Deslocamento em memória)



Para deslocamentos na memória, só é permitido um bit a cada instrução.

dR = indica a direção do deslocamento:
dR = Ø ⟶ à direita
dR = 1 ⟶ à esquerda

Endereço Efetivo = Neste campo, deverá ser especificado o modo de endereçamento e qual o registrador. Para tal, usar a tabela 1.12.

Flags : Todos os flags são afetados.

B C C Desvio relativo condicional

Resumo : Se (condicional for verdadeira) então ⟶ PC + d = PC

Sintaxe do Assembler : BCC <Label>

Tamanho : Byte ou Word

Descrição : Se a condicional é satisfeita, então a execução do programa continuará na localização PC + deslocamento. O deslocamento é um complemento de dois, podendo o seu valor ser positivo ou negativo. Se o deslocando de oito bits contido na instrução for = ØØ, então será utilizado o word imediato à instrução. As condicionais possíveis são:

Condicionais	Código	Flags
BCC se carry = clear	0100	S
BCS se carry = set	0101	C
BEQ se igual	0111	Z
BGE se maior ou igual	1100	$N.V + \overline{N}.\overline{V}$
BGT se maior do que	1110	$N.V.\overline{Z} + \overline{N}.\overline{V}.\overline{Z}$
BHI se high	0010	$\overline{C}.\overline{Z}$
BLE se menor ou igual	1111	$Z + N.\overline{V} + \overline{N}.V$
BLS se low ou same	0011	$C + Z$
BLT se menor do que	1101	$N.\overline{V} + \overline{N}.V$
BMI se menos	1011	N
BNE se não igual	0110	$\overline{Z}$
BPL se plus	1010	$\overline{N}$
BVC se overflow clear	1000	$\overline{V}$
BVS se overflow set	1001	V

Formato da Instrução:

15	14	13	12	11 10 9 8	7 6 5 4 3 2 1 0
0	1	1	0	Condicional	8 bits de deslocamento
16 bits de deslocamento quando 8 bits = ØØ					

Condicional = Será uma das quatorze condicionais mostradas nesta instrução.

8 bits de deslocamento = Inteiro, complemento de dois, mostrando a relativa distância (em bytes) entre a instrução BCC e a próxima instrução a ser executada, se a condição for satisfeita.

16 bits de deslocamento = Operação idêntica à descrita para 8 bits, sendo que neste caso são 16 bits e só é possível se o campo de 8 bits for igual a ØØ.

Flags : Não são afetados.

B C H G Testa um bit e troca

Resumo : ~ (< número do bit> do destino)→ Z depois
~ (< número do bit> do destino)→ <número do bit> do destino.

Sintaxe do Assembler : BCHG Dn, <ea>
BCHG # <DATA>,<ea>

Tamanho : Byte e long word

Descrição : Um bit no operando destino é testado e o seu estado é colocado no flag Z. Depois de testado, o estado do bit especificado é trocado no operando destino.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Registrador Dn			1	0	1	Endereço Efetivo Modo			Registrador		

Registrador DN = Especifica o registrador de dados Dn, que contém um número, que por sua vez é o número do bit que deverá ser testado no operando destino tamanho = 32 bits.

Endereço Efetivo = Especifica o destino, usar a tabela 1.13 para obtê-lo.

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
–(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8PC,Xn)	—	—

*Long only; all others are byte only.

Tabela 1.13 - Modos de endereçamento.

Formato da Instrução: Quando utilizar um número imediato para especificar o bit.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	Endereço Efetivo Modo Registrador					
0	0	0	0	0	0	0	0	Número do bit							

Campo do número do bit = especificar o bit que se deseja testar, tamanho igual a 8 bits.

Endereço efetivo = Utilizar a tabela 1.13 para identificar o endereço desejado.

Flags : Somente o Z

B C L R Testa um bit e limpa

Resumo : ~ (<número do bit> do destino)—> Z e então
Ø → <número do bit do destino>

Sintaxe do Assembler : BCLR Dn <ea>
BCLR # <data>, <ea>

Tamanho Byte e long word

Descrição : Um bit do operando destino é lido e o seu estado é transferido para o flag Z, em seguida o conteúdo deste bit do operando é zerado.
Se for utilizado o conteúdo de um registrador de dados, poderão ser acessados 32 bits. Se for utilizado um número declarado após a instrução, só serão possíveis 8 bits.

Formato de Instrução: Quando for utilizado um registrador de dados

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Registrador Dn			1	1	0	Endereço Efetivo Modo Registrador					

Registrador Dn = Registrador de dados que contém o número do bit a ser acessado tamanho = 32 bits.

Endereço efetivo = Especifica o destino. Usar a tabela 1.13 para obtê-lo.

Formato da Instrução: Quando for utilizado um dado imediato.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	Endereço Efetivo Modo			Endereço Efetivo Registrador		
0	0	0	0	0	0	0	0	Número do bit							

Endereço efetivo = Especifica o destino, usar a tabela 1.13 para obtê-lo.

Número do bit = Especifica o bit que se deseja limpar. Tamanho igual a 8 bits.

Flags : Apenas o Z

B K P T Breakpoint

Resumo : No 68010 e 68012 → executa exceção de Breakpoint.
No 68000 e 68008 → executa exceção de instrução ilegal

Sintaxe do Assembler : BKPT #<data>

Tamanho : --

Descrição : Esta instrução é utilizada para ajudar no desenvolvimento do programa, pois através da mesma, pode se colocar "paradas" em alguns pontos do programa para então serem possíveis algumas análises. Nos processadores 68010 e 68012, esta instrução é reconhecida como breakpoint, enquanto no 68000 e 68008 não, por isso acontecerá um "TRAP", ou seja, um processamento de exceção de instrução ilegal.

Formato de Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	1	BKPT #		

BKPT# = dado imediato com valor de Ø a 7, especificando 8 pontos de breakpoint por software.

Flags : Não são afetados

B R A — Desvio incondicional

Resumo : PC + d ⟶ PC

Sintaxe do Assembler : BRA <label>

Tamanho : Byte ou word

Descrição : A execução continua na localização PC+ deslocamento, após a instrução BRA. O deslocamento é um número interno em complemento de dois, que conta a relativa distância em bytes. Se o deslocamento de 8 bits existente na instrução for igual a zero, então será utilizado o word imediato à instrução.

Formato de Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	Deslocamento de 8 bits							
Deslocamento de 16 bits deslocar de 8 bits = ØØ															

Flags : Não são afetados.

B S E T — Testa um bit e seta

Resumo : ~ (<número do bit> do destino) ⟶ Z
1 ← <número do bit do destino

Sintaxe do Assembler : BSET Dn, <ea>
BSET #<data>, <ea>

Tamanho : Byte e long word

Descrição : Um determinado bit do operando é testado, sendo o seu estado refletido no flag Z. Em seguida, é imposto estado lógico 1 a este bit.
Se for utilizado um registrador de dados, pode-se acessar os 32 bits do registrador. Se for utilizado um endereço de memória, só será possível acessar 8 bits.

O número do bit poderá ser especificado de duas formas:

1 - Imediato ⟶ constante de 8 bits

2 - Registrador ⟶ número em 32 bits

Formato da Instrução: Via Registrador

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Registrador Dn			1	1	1	Endereço Efetivo Modo			Registrador		

Registrador Dn = Especifica o registrador de dados que tem o número do bit que se deseja acessar.

Endereço efetivo = Especifica o endereço destino, onde se deseja alterar algum bit. O tipo de endereçamento pode-se escolher na tabela 1.13.

Formato da Instrução: Via Dado Imediato

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	Endereço Efetivo Modo			Registrador		
0	0	0	0	0	0	0	0	Número do bit							

Número do bit = dado de 8 bits

Endereço efetivo = formação idêntica ao processo via registrador

Flags : Somente o flag Z

B S R Desvio para sub-rotina

Resumo : SP-4 ⟹ SP; PC → (SP); PC + d → PC

Sintaxe do Assembler : BSR <label>

Tamanho : Byte, word

Descrição : O long word que vem após a instrução BSR será preservado no SP, em seguida, o programa continuará no endereço PC + deslocamento. Este deslocamento é um número inteiro complemento de dois, que contém a dis

tância relativa em bytes entre a instrução BSR e a sub-rotina que se deseja atingir.
Se os oito bits de deslocamento que estão contidos na instrução for zero, então prevalece os 16 bits do próximo word.

Formato da Instrução:

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
0	1	1	0	0	0	0	1	deslocamento de 8 bits
Se 8 bits = ØØ então teremos deslocamento de 16 bits								

Flags : Não são afetados

B T S T Testa um bit

Resumo : ~ (<número do bit> do destino)⇒ Z

Sintaxe do Assembler : BTST Dn, <ea>
BTST #<data>, <ea>

Tamanho : Byte e long word

Descrição : Um determinado bit do destino é testado, ou seja, coloca o conteúdo do seu estado lógico no flag Z. Se a operação usar um registrador de dados para definir o número do bit do destino, que será alterado, pode-se então acessar 32 bits, mas se for utilizado um dado imediato, só será possível acessar 8 bits do destino.

Formato da Instrução: (Via registrador)

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
0	0	0	0	Registrador Dn	1	0	0	Endereço Efetivo Modo Registrador

Registrador Dn = Especifica o registrador de dados, onde será definido o bit do "destino" a ser alterado.

Endereço efetivo = Especifica o "destino" que terá um de seus bits acessado. Para escolher o modo de endereçamento utiliza a tabela 1.13.

Flags : Só o flag Z será afetado.

Formato da Instrução: (Via dado imediato)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	Endereço efetivo Modo			Registrador		
0	0	0	0	0	0	0	0	Número do bit							

Endereço efetivo = Análise idêntica ao formato de instrução via registrador de dados.

Número do bit = Byte onde será especificado qual o bit a ser testado.

Flags : Somente o flag Z será alterado

C H K Testar limite superior e inferior do registrador de dados

Resumo : Se Dn <0 or Dn> Origem então TRAP

Sintaxe do Assembler : CHK <ea>, Dn

Tamanho : word

Descrição : O conteúdo do primeiro word (word menos significativo) do registrador de dados, especificado na instrução, é examinado e comparado com o limite superior, especificado no <ea>, (o valor superior é um número inteiro de complemento de dois) e com o limite inferior que é o zero. Se o valor contido no registrador de dados for menor que zero e maior que definido no <ea>, então o microprocessador inicia um processamento de exceção. É gerado o vetor de exceção para a instrução CHK.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Registrador Dn			1	1	0	Endereço Efetivo Modo			Registrador		

Registrador Dn = Contém o número de registrador de dados que terá o seu conteúdo verificado.

Endereço Efetivo = Especifica o operando que fornece o limite superior.
Os endereçamentos possíveis são os definidos pela tabela 1.11.

Flags : X = não se altera
N = altera
Z,V,C = indefinido

C L R Limpa em operando

Resumo : Ø ⟶ destino

Sintaxe do Assembler : CLR <ea>

Tamanho : Byte, word e long word

Descrição : O destino é zerado, ou seja, é colocado estado lógico Ø em todos os bits da operação.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Tamanho		Endereço efetivo Modo			Registrador		

Tamanho ⇒ 00 = byte
01 = word
10 = long word

Endereço efetivo ⇒ Especifica o destino que será alterado. Usar a tabela 1.13 para obter o modo de endereçamento e o registrador.

Flags : X N Z V C

-	0	1	0	0

X = não é afetado
N = sempre zerado
Z = sempre setado
V = sempre zerado
C = sempre zerado

C M P — Comparar

Resumo : Destino - origem

Sintaxe do Assembler : CMP <ea>, Dn

Tamanho : Byte, word e long word

Descrição : Subtrai o operando origem do operando destino, sendo que o operando origem é um endereço de memória e o destino é um registrador de dados. Os conteúdos dos operandos não se alteram. O resultado da operação altera apenas os flags.

Formato da Instrução:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	0	1	1	Registrador Dn	Op-Mode	Endereço Efetivo Modo Registrador

Registrador Dn ⇒ especifica o registrador destino

Op-Mode ⇒ Byte = 000
Word = 001
long = 010

Endereço Efetivo= especifica o modo de endereçamento para a origem. Utilizar a tabela 1.10 para determinar o modo e o registrador para endereçamento.

Flags : X N Z V C

-	*	*	*	*

X = não é afetado
N = set se o resultado for negativo. Caso contrário reset
Z = set se o resultado for igual. Caso contrário reset
V = Set se houver "overflow". Caso contrário reset
C = Set se houver "borrow". Caso contrário reset

Observação Importante

Existem 4 instruções de comparações: 1º CMP, 2º CMPA, 3º CMPA, 4º CMPM.

A instrução CMP é utilizada quando um dos operandos for um registrador de dados e o outro um registrador de endereços.

A instrução CMPA é utilizada quando um dos operandos for registrador de endereços de memória.

A instrução CMPI é utilizada quando um dos operandos for um dado imediato.

A instrução CMPM é utilizado quando os dois operandos forem endereços de memória.

C M P A — Compara Memória

Resumo : Endereço destino - Endereço origem

Sintaxe do Assembler : CMPA <ea>, An

Tamanho : Word e long word

Descrição : Subtrai o operando origem do operando destino, sem alterar os seus respectivos conteúdos, alterando os estados lógicos dos flags.

Formato da Instrução:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	Registrador An	Op-Mode	Endereço Efetivo Modo	Endereço Efetivo Registrador

Registrador An = especifica o registrador de endereço que servirá de destino.

Op-Mode = 011 - word
111 - long word

Endereço Efetivo = especifica o operando origem. Utilizar a tabela 1.10 para obter o endereçamento desejado.

Flags : X N Z V C

-	*	*	*	*

X = não é afetado
N = setado se resultado é negativo. Caso contrário resetado
Z = setado se resultado é zero(igual). Caso contrário resetado
V = setado se resultado houve overflow. Caso contrário resetado
C = setado se resultado houve borrow. Caso contrário resetado

C M P I — Comparação Imediata

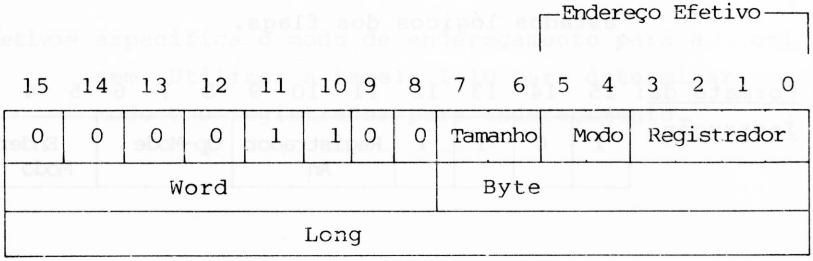
Resumo : Destino - Dado imediato

Sintaxe do Assembler : CMPI #<DATA>, <ea>

Tamanho : Byte, word e long word

Descrição : Subtrai um dado imediato do operando destino, sem alterar o conteúdo do operando destino. O resultado desta operação afetará os flags do registrador de código de condição.

Formato da Instrução:



Tamanho: 00 = byte
01 = word
10 = long word

Endereço efetivo = Especifica o operando destino. Utilizar a tabela 1.13 para obter o endereçamento.

Flags : X N Z V C

-	*	*	*	*

X = não é afetado
N = setado se o resultado é negativo. Caso contrário é resetado
Z = setado se o resultado é zero. Caso contrário é resetado
V = setado se o resultado houve Overflow. Caso contrário é resetado
C = setado se o resultado houve Borrow. Caso contrário é resetado

C M P M — Comparar Memória

Resumo : Destino - origem

Sintaxe do Assembler : CMPM (Ay)+, (Ax)+

Tamanho : Byte, word, long word

Descrição : Subtrai o conteúdo do operando origem do conteúdo do operando destino. O conteúdo do operando destino não sofre alteração. As alterações acontecerão apenas nos flags do registrador de códigos de condições.

Formato :

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	0	1	1	Registrador Ax	1	Tamanho	0	0	1	Registrador Ay

Registrador Ax (indica o destino)
Registrador Ay (indica o origem)

Tamanho: ØØ = Byte
Ø1 = word
1Ø = long word

Flags : X N Z V C

-	*	*	*	*

X = não é afetado
N = setado se resultado é negativo. Caso contrário é resetado
Z = setado se resultado é zero. Caso contrário é resetado
V = setado se resultado houve overflow. Caso contrário é resetado
C = setado se resultado houve borrow. Caso contrário é resetado

D B c c Testa condição, decrementa e desloca

Resumo : Se a condição é falsa então (Dn -1 ⇒ Dn; se Dn ≠ -1 então PC + d ⇒ PC).

Sintaxe do Assembler : DBcc Dn, <label>

Tamanho : word

Descrição : Esta instrução é formada por 3 partes distintas. A primeira parte é um teste de condição, a segundo um contador e a terceira um deslocamento. Esta instrução forma um loop, porque primeiro é testado se a condição desejada foi satisfeita, em caso afirmativo está terminada a execução da instrução, em caso negativo, o word menos significativo da instrução é decrementado em um bit. Quando este contador chegar em -1 também a instrução termina. Enquanto não for satisfeita a condição desejada, o programa continua na localização indicada pelo valor corrente do PCA mais um deslocamento de 16 bits, definido na própria instrução.

O cc na instrução pode ser:

CC	carry clear	0100	$\overline{C}$
CS	carry set	0101	C
EQ	equal	0111	Z
F	never true	0001	O
GE	greater or equal	1100	$N.V+\overline{N}.\overline{V}$
GT	greater than	1110	$N.V.\overline{Z}+\overline{N}.\overline{V}.\overline{Z}$
HI	high	0010	$\overline{C}.\overline{Z}$
LE	lers or equal	1111	$Z+N.\overline{V}+\overline{N}.V$

LS	low or same	0011	C + Z
LT	lers than	1101	$N.\overline{V}+\overline{N}.V$
MI	minus	1011	N
NE	not equal	0110	$\overline{Z}$
PL	plus	1010	$\overline{N}$
T	always true	0000	1
VC	overflow clear	1000	$\overline{V}$
VS	overflow set	1001	V

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	condição				1	1	0	0	1	Registrador		
Deslocamento															

Condição ⇒ um dos dezesseis códigos da tabela de condições definida nesta instrução.

Registrador ⇒ especifica qual registrador de dados que atuará como contador.

Deslocamento ⇒ indica a distância em bytes que será feito o salto (BRANCH).

Flags : Nenhum é afetado

D I V S Divisão com Sinal

Resumo : Destino/origem ⇒destino

Sintaxe do Assembler : DIVS <ea>, Dn 32/16 → 16R:16q.

Tamanho : word

Descrição : Divide o operando destino pelo origem e guarda o resultado no operando destino. Importante observar que o resultado é um dado de 32 bits, para isso temos o operando destino sendo de 32 bits e o operando origem em 16 bits. O quociente é o word menos significativo do operando destino e o resto é o word mais significativo. Neste tipo de divisão, os sinais dos operandos são levados em conta.

Observação Importante:

Divisão por zero causa uma interrupção especial (TRAP) e o transbordo do resultado é acusado pelo flags de overflow.

Formato de Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Registrador Dn			1	1	1	Endereço Efetivo Modo			Registrador		

Registrador Dn : especifica o operando destino, que pode ser qualquer um dos 8 registradores de dados.

Endereço Efetivo : especifica o operando origem. Os tipos de endereçamentos possíveis são so pertencentes à tabela 1.11..

Flags :

C	V	Z	N	X
0	*	*	*	-

C = sempre zerado

V = seta se a divisão gerou um overflow. Reseta em caso contrário

Z = seta se o quociente é zero. Reseta em caso contrário

N = seta se o quociente é negativo. Reseta em caso contrário
X = N é afetado.

D I V U Divisão sem Sinal

Resumo : Destino/Origem ⟹ Destino

Sintaxe do Assembler : DIVU < ea >, Dn 32/16 ⟶ 16r:16q.

Tamanho : word

Descrição : Divide o operando destino pelo origem e guarda o resultado no operando destino. O operando destino é um long word dividido entre o quociente (word menos significativo) e o resto (word mais significativo). Neste tipo de divisão, não é levado em conta os sinais dos operandos.

Observação Importante:

A divisão por zero causa uma interrupção especial (TRAP) é transbordo ou estouro de resultado é acusado pelo flag de overflow.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Registrador Dn			0	1	1	Endereço Efetivo Modo			Registrador		

Registrador Dn : Especifica o operando destino que pode ser qualquer um dos 8 registradores de dados.

Endereço efetivo : Especifica o operando origem. Os tipos de endereçamentos possíveis são os pertencentes aos da tabela 1.11.

Flags

C	V	Z	N	X
Ø	*	*	*	-

C = permanece zerado
V = setado se um overflow for deletado. Caso contrário será resetado
Z = setado se o quociente igual a zero. Caso contrario será resetado
N = setado se o quociente for negativo. Caso contrário será resetado
X = não é afetado

E O R Lógica ou Exclusivo

Resumo : Origem ⊕ Destino ⟹ Destino

Sintaxe do Assembler : EOR Dn, <ea>

Tamanho : Byte, word e long word

Descrição : Executa a operação lógica ou exclusivo entre o operando origem e o destino, sendo que o resultado ficará no operando destino. Como operando origem, só poderá ser utilizado registradores de dados, sendo que no destino é definido um campo de endereçamento.

Formato da Instrução:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	0	1	1	Registrador Dn	Op-Mode	Endereço Efetivo Modo Registrador

Registrador Dn ⇒ qualquer registrador de dados
Op-Mode ⇒ byte - 100
word - 101
long - 110
Endereço Efetivo ⇒ especifica o campo do endereçamento. Os endereçamentos possíveis são so definidos na tabela 1.14.

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	—	—
(An)+	011	reg. number:An			
-(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	—
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	—

Tabela 1.14 - Modos de Endereçamento.

Flags : C V Z N X

Ø	Ø	*	*	-

C = permanece zerado

V = permanece zerado

Z = será setado se o resultado for zero. Resetado em caso contrário

N = será setado se o bit mais significativo for setado. Resetado em caso contrário.

X = não é afetado

E O R I OR - exclusivo imediato

Resumo : Dado imediato ⊕ Destino ⟹ Destino

Sintaxe do Assembler : EORI #<DATA>, <ea>

Tamanho : Byte, word e long word

Descrição : Executa a operação lógica ou exclusiva do operando destino com um dado imediato, colocando o resultado no operando destino.

Formato da Instrução:

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	0	0	0	1	0	1	0	Tamanho	Endereço Efetivo Modo	Registrador
word (16 bits)								byte (8 bits)		
long word (32 bits)										

Tamanho: 00 = operação com byte
01 = operação com word
10 = operação com long word

Endereço efetivo : Especifica o operando destino. Usar a tabela 1.14 para obter os modos de endereçamentos possíveis.

Flags : C V Z N X

Ø	Ø	*	*	-

X = não é afetado
N = setado se o bit mais significativo do resultado for 1, caso contrário será resetado.
Z = setado se o resultado igual a zero, caso contrário será resetado.
V = permanece zerado
C = permanece zerado

EORI TO CCR Ou exclusivo imediato com os códigos de condições

Resumo : Origem ⊕ CCR ⟹ CCR

Sintaxe do Assembler : EORI #<data>, CCR

Tamanho : byte

Descrição : Executa a operação lógica ou-exclusivo entre um operando imediato de comprimento igual a um byte, com os 8 bits menos significativos do registrador de código de condições.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Dado de 8 bits							

Flags : C V Z N X

C	V	Z	N	X
*	*	*	*	*

C = troca se o bit Ø do operando igual a 1. Mantém se operando = Ø
V = troca se o bit 1 do operando igual a 1. Mantém se operando = Ø
Z = troca se o bit 2 do operando igual a 1. Mantém se operando = Ø
N = troca se o bit 3 do operando igual a 1. Mantém se operando = Ø
X = troca se o bit 4 do operando igual a 1. Mantém se operando = Ø

EORI TO SR — Ou exclusivo imediato com o "status register" (instrução privilegiada)

Resumo : Se o processador estiver no estado supervisor então: Origem ⊕ SR ⟹ SR, senão Trap.

Sintaxe do Assembler : EORI #<data>, SR

Tamanho : word

Descrição : Executa a operação lógica ou-exclusivo, entre um operando imediato de comprimento igual a um word e o registrador de status.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0
Word data (16 bits)															

Flags : C V Z N X

C	V	Z	N	X
*	*	*	*	*

C = Troca se bit Ø do operando igual a 1. Mantém se bit Ø do operando igual a Ø.

V = Troca se bit 1 do operando igual a 1. Mantém se bit 1 do operando igual a Ø.

Z = Troca se bit 2 do operando igual a 1. Mantém se bit 2 do operando igual a Ø.

N = Troca se bit 3 do operando igual a 1. Mantém se bit 3 do operando igual a Ø.

X = Troca se bit 4 do operando igual a 1. Mantém se bit 4 do operando igual a Ø.

E X G — Troca conteúdo dos registradores

Resumo : Rx ◄► Ry

Sintaxe do Assembler : EXG Dx,Dy
EXG Dx,Ax
EXG Ax,Ay

Tamanho : long

Descrição : Troca o conteúdo de dois registradores. Esta é feita no registrador completo, ou seja, 32 bits. Pode ser também entre registradores de dados, endereços ou entre dados e endereços.

Formato da Instrução:

15	14	13	12	11 10 9	8	7 6 5	4 3 2 1 0
1	1	0	0	Registrador Rx		Op-Mode	Registrador Ry

Op-Mode = 01000 ⇒ entre registradores de dados
01001 ⇒ entre registradores de endereços
10001 ⇒ entre registradores de dados e endereços

Registrador Rx e Ry ⇒ definir o número do registrador que será afetado.

Flags : não são alterados

E X T — Sinal estendido

Resumo : Destino com sinal estendido ⟶ destino

Sintaxe do Assembler : EXT Dn

Tamanho : Word, long word

Descrição : Estende o bit de sinal de um registrador de dados de um byte para um word, ou de um word para um long word, dependendo do comprimento selecionado. Se foi selecionada uma operação com word, o bit 7 do registrador de dados escolhido é copiado para o bit 15, se for selecionada uma operação com long word, então o bit 15 será copiado para o bit 31.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	Op-Mode			0	0	0	Registrador Dn		

Op-Mode = especifica o tamanho da operação
- 010 - estende o sinal do byte menos significativo do registrador de dados para word.
- 011 - estende o sinal do word menos significativo do regsitrador de dados para long word.

Registrador Dn = define o registrador de dados Dn.

Flags :

C	V	Z	N	X
Ø	Ø	*	*	-

C = sempre zerado
V = sempre zerado
Z = seta se o resultado igual a zero. Reseta se contrário
N = seta se o resultado é negativo. Reseta se contrário
X = não é afetado

ILLEGAL Instrução Ilegal

Resumo : SSP-2 ⇒ SSP; Vector offset → (SSP) (68008)
SSP-4 ⇒ SSP; PC ⇒ (SSP)
SSP-2 ⇒ SSP; SR ⇒ (SSP)
Endereço do Vetor de instrução ilegal → PC

Sintaxe do Assembler : ILLEGAL

Tamanho : Não há

Descrição : Esta instrução causa um processamento de exceção do tipo "instrução ilegal".

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

Flags : Não são afetados

J M P Instrução de Salto

Resumo : Endereço destino → PC

Sintaxe do Assembler : JMP <ea>

Tamanho : Não há

Descrição : O programa continua a partir do endereço especificado na instrução.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	Endereço efetivo Modo			Registrador		

Endereço efetivo = especifica o endereço da próxima instrução.
Os modos permitidos estão na tabela 1.15.

Flags : Não é afetado

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	—	—	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	—	—
(An)+	—	—			
-(An)	—	—			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	010
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	011

Tabela 1.15 - Modos de endereçamento.

J S R — Salto para sub-rotina

Resumo : SP-4 ⇒ SP; PC → (SP)
Endereço destino → PC

Sintaxe do Assembler : JSR <ea>

Tamanho : Não há

Descrição : Esta instrução é uma chamada de sub-rotina. O endereço da instrução, imediatamente seguinte ao JSR, é colocada no stack. O programa então continua pelo endereço especificado na instrução.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	Endereço Efetivo Modo			Registrador		

Endereço efetivo = Especifica o endereço da próxima instrução. Os modos permidos estão na tabela 1.15.

Flags : Não são afetados

L E A Carrega endereço efetivo

Resumo : <ea> ⇒Am

Sintaxe do Assembler : LEA <ea>, An

Tamanho : Long word

Descrição : Um endereço efetivo é carregado dentro do registrador Am.

Formato da Instrução:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	1	0	0	Registrador An	1	1	1	Endereço Efetivo Modo	Endereço Efetivo Registrador

Registrador An ⇒ Especifica o registrador An onde será carregado o endereço efetivo.

Endereço Efetivo ⇒ Especifica o endereço a ser carregado. Os modos permitidos estão na tabela 1.15.

Flags : Não são afetados

L I N K Ligar e Alocar

Resumo : SP-4 ⇒ SP; An → (SP)
SP → An; SP+d → SP

Sintaxe do Assembler : LINK An, #<deslocamento>

Tamanho : Não há

Descrição : O conteúdo do registrador (An) especificado é colocado no stack. Depois da função "PUSH" ser executada, o valor do apontador de stack (stack pointer) é colocado no registrador An. Por fim, o endereço do stack pointer é somado ao deslocamento de 16 bits estendido em sinal.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	Registrador An		
Deslocamento de 16 bits															

Registrador An = Qualquer registrador de endereço An.

Deslocamento = Um word. Especifica o complemento de dois que será adicionado ao apontador de stack.

Flags : Não são afetados

LSL, LSR Deslocamento lógico

Resumo : Destino deslocado por <COUNT> ⟹ destino

Sintaxe do Assembler : LSd Dx,Dy
LSd #<data>, Dy
LSd <ea>
O d define se é esquerda (L) ou direita (R)

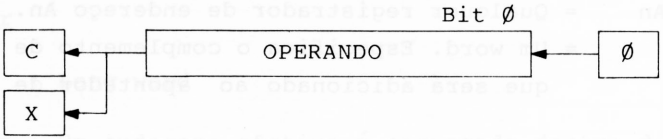
Tamanho : Byte, word ou long word

Descrição : Desloca os bits do operando na direção que for especificada. O carry bit receberá o último bit que foi deslocado para fora do operando. A contagem do número de deslocamento pode ser efetuada de duas formas:

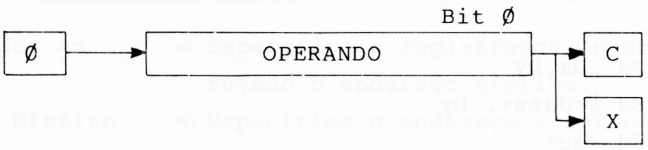
1. Imediata = O número de deslocamentos é especificado na própria instrução. A faixa de deslocamento vai de 1 até 8.
2. Registrador = O número de deslocamentos está definido no conteúdo de um registrador de dado Dn (a contagem tem um módulo = 64).

Se for utilizado um endereço de memória para ser deslocado, é importante saber que só é possível deslocar 1 bit por instrução.

Na instrução LSL, o operando é deslocado para a esquerda. Os bits deslocados para fora do registrador (através do bit mais significativo) vão para o carry bit e o extend bit. Zeros são deslocados para dentro do registrador, através do bit menos significativo.



Na instrução LSR, o operando é deslocado à direita. Os bits deslocados para fora do registrador (através do bit menos significativo) vão para o carry bit e extend bit. Zeros são deslocados para dentro do registrador através do bit mais significativo.



Formato da: (Deslocamento em registrador)

Instrução

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	CONT/REG	dR	SIZE	I/R	0	1	Registrador Dn

SIZE = Especifica o tamanho do operando

ØØ = byte

Ø1 = word

1Ø = long word

dR = Define a direção

Ø = direita

1 = esquerda

Registrador Dn = Define qual o registrador de dados que terá o seu conteúdo deslocado.

I/R = Define se é imediato ou feito através de registrador.

Se I/R = Ø, então o campo "CONT" define o registrador (módulo 64).

Se I/R = 1, então o campo "CONT" define o deslocamento. Os valores de Ø a 7 representam de 1 até 8 deslocamentos respectivamente.

Formato da Instrução: (Deslocamento na Memória)

15	14	13	12	11	10	9	8	7	6	5 4 3 2 1 0
1	1	1	0	0	0	1	dR	1	1	Endereço Efetivo Modo Registrador

dR = Define a direção DR = 1 → esquerda
DR = Ø → direita

Endereço efetivo = Especifica o operando que será deslocado. As formas possíveis estão na tabela 1.12

Flags : X N Z V C

X	N	Z	V	C
*	*	*	Ø	*

C = Será alterado (set ou reset) de acordo com o conteúdo do último bit que for deslocado para este flag.

V = Manterá zerado

Z = Setado se o resultado ficar igual a zero, senão será resetado

N = Setado se o resultado for negativo, senão será resetado

X = Será alterado (set ou reset) de acordo com o conteúdo do último bit que for deslocado para este flag.

M O V E Move dados da origem para o destino

Resumo : Origem ⟹ Destino

Sintaxe do Assembler : MOVE <ea>,<ea>

Tamanho : Byte, word ou long word

Descrição : Move para a localização destino o conteúdo da localização origem.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	Tamanho		Destino Registro e Modo						Origem Modo			Registro		

Tamanho : 01 = byte
11 = word
10 = long word

Endereçamento destino : Especifica o modo e o registrador destino através da tabela 1.16.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

Tabela 1.16 - Modo de endereçamento.

Endereçamento Origem : Especifica o modo e o registrador origem, através da tabela 1.17.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011

*For byte size operation, address register direct is not allowed.

Tabela 1.17 - Modos de endereçamento.

Importante:

Existem outras variações da instrução "MOVE", que serão estudadas logo a seguir.

Note que MOVE A é utilizado quando o destino é um registrador de endereço, e MOVE Q pode também ser usado para certas operações, utilizando o registrador de dados.

Flags : X N Z V C

-	*	*	Ø	Ø

C = Será zerado
V = Será zerado
Z = Será setado se o resultado for. zero. Resetado em caso contrário
N = Será setado se o resultado for negativo. Resetado em caso contrário
X = Não será alterado

MOVE A Move endereço

Resumo : Origem (endereço efetivo) ⟹ Destino (An)

Sintaxe do Assembler : MOVE A <ea>, An

Tamanho : Word, long word

Descrição : Move o conteúdo de um endereçamento efetivo, para um destino dado por um registrador de endereço.

Formato da Instrução:

15	14	13 12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	Tamanho	Registro Destino	0	0	1	Origem Modo	Origem Registro

Tamanho : 11 = word
10 = long word
Registro destino: qualquer registrador de endereço An.
Origem : especifica a localização origem, dada pelo modo de endereçamento da tabela 1.10.

Flags : Não são afetados.

MOVE FROM CCR Move do CCR para uma localização dada pela < ea>

Resumo : CCR ⟶ Destino <ea>

Sintaxe do Assembler : MOVE CCR, <ea>

Tamanho : word

Descrição : O conteúdo do registrador de código de condições (flag), é colocado no operando destino.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	Endereçamento Modo			Registrador		

Endereçamento = Especifica a localização destino. São permitidos os modos da tabela 1.16.

Flags : Não são afetados.

MOVE TO CCR Move para o CCR

Resumo : Origem <ea> ⟶ CCR

Sintaxe do Assembler : MOVE <ea> , CCR

Tamanho : Word

Descrição : O conteúdo do operando origem é colocado no registrador de códigos de condições (flags).

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	Origem Modo			Registrador		

Origem = Especifica a localização origem. Os modos permitidos estão na tabela. 1.18.

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	111	100
(An)+	011	reg. number:An			
-(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	010
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	011

Tabela 1.18 - Modos de endereçamento.

Flags : X N Z V C

*	*	*	*	*

X = Valor igual ao bit 4 do operando origem
N = Valor igual ao bit 3 do operando origem
Z = Valor igual ao bit 2 do operando origem
V = Valor igual ao bit 1 do operando origem
C = Valor igual ao bit 0 do operando origem

MOVE FROM SR Move do SR para <ea>

Resumo : SR ⟹ Destino <ea>

Sintaxe do Assembler : MOVE SR, <ea>

Tamanho : Word

Descrição : O conteúdo do status register é movido para a localização destino. Se o processador estiver no estado supervisor, a instrução será executada, caso contrário acontece um TRAP.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Destino Modo			Registrador		

Destino = Especifica o modo de endereçamento que será utilizado como destino. Os modos permitidos estão na tabela 1.16.

Flags : Não são afetados

MOVE TO SR Move o conteúdo do <ea> para SR

Resumo : Se estado supervisor então origem <ea>→SR senão TRAP

Sintaxe do Assembler : MOVE <ea>, SR

Descrição : O conteúdo do operando origem é colocado no status register. Esta é uma instrução privilegiada, ou seja, só pode ser executada no estado supervisor.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	Origem Modo			Registrador		

Origem = Especifica o modo de endereçamento origem. Os modos permitidos estão na tabela 1.10.

Flags : Serão setados ou resetados de acordo com o operando origem.

MOVE USP Move o stack do usuário

Resumo : Se estado supervisor então USP→An ou An→USP senão TRAP.

Sintaxe do Assembler : MOVE USP, An
MOVE An, USP

Tamanho : Long word

Descrição : O conteúdo do stack pointer do usuário é transferido para o registrador An escolhido ou vice-versa.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	DR	Registrador An		

DR = Especifica a direção
Ø = Do reg An para USP
1 = Do USP para reg An

Registrador An = Especifica o registrador

Flags : Não são afetados

MOVEC Move registrador de controle

Resumo : Se estado supervisor então Rc → Rn ou Rn para Rc senão TRAP.

Sintaxe do Assembler : MOVEC Rc, Rn
MOVEC Rn, Rc

Tamanho : Long Word

Descrição : Copia o conteúdo do registrador de controle RC para um registrador de uso geral Rn, ou copia o conteúdo de um registrador de uso geral Rn para o registrador de controle RC. A operação é de 32 bits.

Formato :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	DR
A/D	Registrador			Registrador de controle											

DR ⟶ especifica a direção
Ø = Rc ⟶ Rn
1 = Rn ⟶ Rc

A/D ⟶ especifica o tipo de registrador
Ø = dados
1 = endereços

Registrador especifica o número do registrador

Registrador de controle : 000 = Registrador SFC
001 = Registrador DFC
800 = Registrador USP
801 = Registrador UBR

Qualquer outro código irá causar um processamento de exceção do tipo instrução ilegal.

MOVEM Move múltiplos registradores

Resumo : Registrador ⟶ Destino
Origem ⟶ Registrador

Sintaxe do Assembler: MOVEM Lista de Registradores, <ea>
MOVEM <ea> , Lista de Registradores

Tamanho : Word e long word

Descrição : Move os conteúdos dos registradores Rx (Dn e/ou An) que forem citados na lista, para as posições consecutivas da memória, a contar do endereço efetivo declarado na instrução (<ea>) em diante.
O conteúdo de cada registrador pode ser de 16 bits (word) ou 32 bits (long).
A sequência de transferência inicia-se pelos registradores de dados e em seguida os de endereços (DØ a D7, AØ a A7), na ordem crescente.
Quando forem declarados os registradores na instrução, devem ser separados pelo sinal "/" e "-", por

exemplo: D_1/D4/A1-A6. No código, objeto será indicado pelo bit 1 colocado no segundo word da instrução, assim formada:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	AØ	D7	D6	D5	D4	D3	D2	D1	DØ

As transferências dos registradores são feitas da direita para a esquerda, ou seja, do bit Ø para bit 15.

A exceção à regra ocorre quando o modo de endereçamento utilizado for 0 - (An), ou seja, indireto pré-decrementado. Neste caso, os registradores são transferidos para posições decrescentes da memória, neste caso a máscara é vista da seguinte maneira:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DØ	D1	D2	D3	D4	D5	D6	D7	AØ	A1	A2	A3	A4	A5	A6	A7

MOVEM <ea>, Lista de registradores

Esta instrução opera de modo contrário, ou seja, move valores consecutivos da memória, a partir do endereço <ea> em diante, para os registradores Dn ou/e An que forem induna na lista. Também neste caso, pode-se movimentar informações de 16 bits ou 32 bits.

Para este caso não é permitido o modo indireto pré-decrementado, não havendo neste caso, a inversão da sequência dos registradores.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	DR	0	1		SZ	Endereço Efetivo Modo			Registrador		
Lista de registrador															

DR ⟶ Especifica a direção da transferência

Ø = Rn ⟶ <ea>

1 = <ea> ⟶ Rn

SZ ⟶ Especifica o tamanho do registrador

Ø = words

1 = long

Endereço efetivo ⟶ Especifica o modo de endereçamento utilizado. Para transferência de registrador para memória, utilizar a tabela 1.19.

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

Tabela 1.19 - Transferência registrador para memória.

Para transferência de memória para registrador, utilizar a tabela 1.20.

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
—	011	reg. number:An
– (An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011

Tabela 1.20 - Transferência memória para registrador.

Flags : Não serão afetados

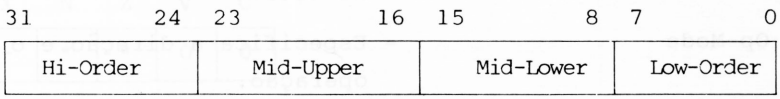
MOVEP Move dados para periféricos

Resumo : Origem ⟶ Destino

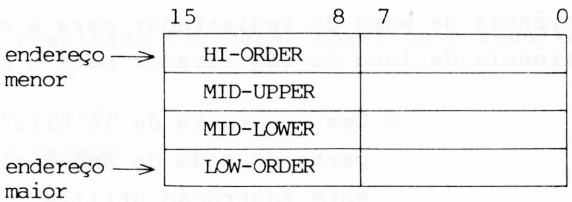
Sintaxe do Assembler : MOVEP Dx (d,Ay)
MOVEP (d,Ay),Dx

Tamanho : Word e Long word

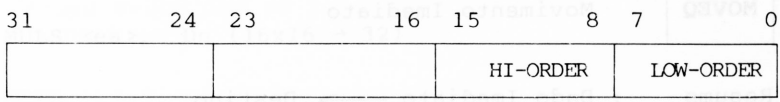
Descrição : Transferência de dados entre um registrador de dados e um endereço de memória. Começando pelo endereço especificado e incrementando de dois em dois endereços. A transferência é feita em bytes, sendo o primeiro o byte mais significativo e em segundo o menos significativo. O modo de endereçamento utilizado é o indireto, mais um deslocamento de 16 bits. Esta instrução foi elaborada para permitir o acesso a periféricos de 8 bits em um data bus de 16 bits.
Para uma transferência de long word de/ou para memória em endereços pares teremos:



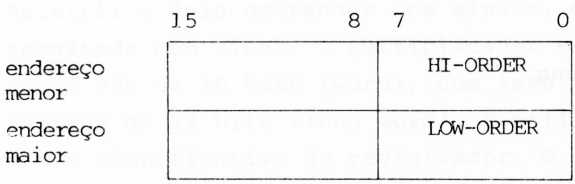
Organização dos bytes na memória



Transferência de word de/ou para memória em endereços ímpares.



Organização dos bytes na memória



Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Registrador de dados			Op-Mode			0	0	1	Registrador de endereço		
Deslocamento															

Registrador de dados = Especifica o registrador de dados que será utilizado na transferência de informações.

Registrador de Endereço = Especifica o registrador de endereço, que será utilizado no modo de endereçamento indireto, com deslocamento de 16 bits.

Op-Mode = Especifica a direção e o tamanho da operação.

100 = Transferência de word da memória para o registrador
101 = Transferência de long da memória para o registrador
110 = Transferência de word do registrador para a memória
111 = Transferência de long do registrador para a memória

Deslocamento = Uma constante de 16 bits a qual faz parte do modo de endereçamento, que esta instrução utiliza.

Flags : Não são afetados

MOVEQ Movimento Imediato

Resumo : Dado Imediato ⟶ Destino

Sintaxe do Assembler : MOVEQ #<data> , Dn

Tamanho : long

Descrição : Movimento imediato de um dado de 8 bits para um registrador de dados. Esta é uma instrução de 32 bits, pois o bit de sinal do dado imediato é estendido em todos os 32 bits de Dn.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	Registro			0	Data							

Registro = Especifica o registrador de dados utilizado na instrução.

Data = 8 bits de dados que são estendidos em sinal para um operando long word.

Flags : X N Z V C

X	N	Z	V	C
-	*	*	0	0

X = não é afetado

N = set se o resultado for negativo, reset se o resultado for positivo

Z = set se o resultado for zero, reset se o resultado for diferente de zero

V = permanece zerado

C = permanece zerado

MULS Multiplicação com sinal

Resumo : Origem * Destino ⟹ Destino

Sintaxe do Assembler : MULS <ea>, Dn (16x16 → 32)

Tamanho : Word

Descrição : Multiplica dois operandos com sinais, obtendo um resultado com sinal. O multiplicador e o multiplicando são de 16 bits (word), com isso temos um resultado de 32 bits (long word). É utilizado o word menos significativo do registrador. O resultado será salvo no operando destino.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Registrador Dn			1	1	1	Endereço Efetivo Modo			Registrador		

Registrador Dn = Especifica um dos registradores da operação. Este campo é definido também como destino.

Endereço Efetivo = Especifica um dos operandos da multiplicação. Os modos de endereçamento permitidos na tabela 1.21.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011

Tabela 1.21 - Modos de Endereçamento.

Flags : X N Z V C

-	*	*	*	0

X = Não é afetado.

N = Set se o resultado for negativo. Reset se o resultado for positivo.

Z = Set se o resultado for zero. Reset se o resultado for diferente de zero.

V = Set se houve overflow. Reset se não houve overflow.

C = Permanece zerado.

MULU Multiplicação sem sinal

Resumo : Origem * Destino ⟶ Destino

Sintaxe do Assembler : MULS <ea>.Dn (16x16 = 32)

Tamanho : Word

Descrição : Executa a multiplicação de dois operandos sem sinal. O multiplicando e o multiplicador são operados com o comprimento de um word, sendo o resultado um operando com long word.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Registrador Dn			0	1	1	Endereço Efetivo Modo			Registrador		

Registrador Dn = Especifica um dos operandos origem. Após a operação, este registrador passa a ser destino.

Endereço Efetivo = Especifica um dos operandos origem. Os modos permitidos de acesso estão na tabela 1.21.

Flags :

X	N	Z	V	C
-	*	*	*	0

X = Não é afetado.
N = Set se resultado negativo. Reset se positivo.
Z = Set se resultado igual a zero. Reset se diferente de zero.
V = Set se overflow. Reset em caso contrário.
C = Permanece zerado.

NBCD Negativas valor decimal com extensão

Resumo : $\emptyset$ - Destino$_{10}$ - x ⟶ Destino

Sintaxe do Assembler : NBCD <ea>

Tamanho : Byte

Descrição : O operando endereçado pelo <ea> e a extensão são subtraídos de $\emptyset$. A operação é realizada em decimal. O resultado é salvo no endereço <ea>. Esta instru

ção produz um complemento de dez do operando destino, se o flag de extensão for zero, e complemento de nove, se o flag de extensão for um. Esta operação tem o comprimento de um byte.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	Endereço Efetivo Modo				Registrador	

Endereço Efetivo = Especifica o operando onde se realizará a subtração. Os modos de endereçamento permitidos estão na tabela 1.22.

Flags :

X	N	Z	V	C
*	I	*	I	*

X = Set ou reset da mesma forma que o carry.

C = Set se houve o "empresta um". Reset em caso contrário.

N = Indefinido.

V = Indefinido.

Z = Reset se o resultado não é zero. Set em caso contrário

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	—	—
(An)+	011	reg. number:An			
-(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	—
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	—

Tabela 1.22 - Modos de endereçamento.

N E G Negativo

Resumo : Ø - Destino ⟶ Destino

Sintaxe do Assembler : NEG <ea>

Tamanho : Byte, word e long word

Descrição : O operando endereçado por <ea> é subtraído de zero. O resultado é guardado na localização destino. Esta operação permite trabalhar com byte, word e long word.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	Tamanho		Endereço Efetivo Modo			Registrador		

Tamanho = Especifica o tamanho da operação
00 = byte
01 = word
10 = long

Endereço efetivo = Especifica a localização destino. Os modos de endereçamento permitidos estão na tabela 1.22.

Flags : X N Z V C

X	N	Z	V	C
*	*	*	*	*

X = Set da mesma forma que o flag carry.
N = Set se o resultado for negativo. Reset em caso contrário.
Z = Set se o resultado for zero. Reset em caso contrário.
V = Set se houve overflow. Reset em caso contrário.
C = Reset se o resultado for zero. Set em caso contrário.

NEGX Negativa com extensão

Resumo : 0 - Destino - x ⟹ Destino

Sintaxe do Assembler : NEGX <ea>

Tamanho : Byte, word e long word

Descrição : O operando <ea> é subtraído de zero e da extensão. O resultado é salvo no endereço <ea>.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	Tamanho		Endereço Efetivo Modo			Registrador		

Tamanho = 00 = byte
01 = word
10 = long

Endereço efetivo = Especifica o modo de endereçamento do operando <ea>. Os modos permitidos estão na tabela 1.22.

Flags : X N Z V C

X	N	Z	V	C
*	*	*	*	*

X = Set da mesma forma que o carry (c)
N = Set se o resultado for negativo. Reset se o resultado for positivo.
Z = Set se o resutlado for zero. Reset se o resultado for diferente de zero.
V = Set se houve overflow. Reset se não houve overflow.
C = Set se houve o "empresta um". Reset se não houve.

N O P — Não há operação

Resumo : Não há nenhuma execução

Sintaxe do Assembler : NOP

Tamanho : Não há

Descrição : Nenhuma operação ocorre. O processador segue para a próxima instrução.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

Flags : Não são afetados

N O T — Complemento lógico

Resumo : ~Destino ⟹ Destino

Sintaxe do Assembler : NOT <ea>

Tamanho : Byte, word e long word

Descrição : Uma operação de complemento de um é feita no operando <ea>, sendo o resultado salvo no próprio endereço <ea>.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Tamanho		Endereço Efetivo Modo			Registrador		

Tamanho ⟹ 00 = byte
01 = word
10 = long

Endereço efetivo = Os modos de endereçamento permitidos estão na tabela 1.22.

Flags :

X	N	Z	V	C
-	*	*	Ø	Ø

X = Não é afetado.
N = Set se o resultado for negativo. Reset se resultado for positivo.
Z = Set se o resultador for zero. Reset se o resultado for diferente de zero.

V = Permanece zerado.

C = Permanece zerado.

O R Lógica ou

Resumo : Origem v Destino ⟹ Destino

Sintaxe do Assembler : OR <ea> , Dn
OR Dn, <ea>

Tamanho : Byte, word e long

Descrição : Executa a função lógica "ou" entre um operando origem e um destino. Os registradores de endereço não podem ser utilizados como operandos.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Registrador Dn			Op-Mode			Endereço Efetivo Modo			Registrador		

Registrador Dn ⟹ = Especifica um dos operandos

Op-Mode =

Byte	Word	Long	Operação
000	001	010	<ea> v Dn ⟹ Dn
100	101	110	Dn v <ea> ⟹ <ea>

Endereço Efetivo = Especifica um dos operandos. Os modos de endereçamento permitidos estão na tabela 1.23.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
–(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011

Tabela 1.23 - Modos de endereçamento.

Flags : X N Z V C

-	*	*	Ø	Ø

X = Não é afetado.

N = Set se o bit mais significativo for setado. Reset se o bit mais significativo for resetado.

Z = Set se o resultado é zero. Reset se o resultado for diferente de zero.

V = Permanece zerado.

C = Permanece zerado.

O R I — Lógica ou Imediata

Resumo : Dado v Destino ⟹ Destino

Sintaxe do Assembler : ORI #<data>, <ea>

Tamanho : Byte, word e long

Descrição : Executa a função lógica "ou" entre um dado imediato e o operando destino.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Tamanho		Endereço Efetivo Modo			Registrador		
Word Imediato								Byte Imediato							
long Imediato															

Tamanho : 00 ⇒ byte
01 ⇒ word
10 ⇒ long

Endereço efetivo = Especifica o operando destino. Os modos permitidos estão na tabela 1.24.

Addr. Mode	Mode	Register
Dn	000	reg. number:An
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

Tabela 1.24 - Modos de endereçamento.

Flags : X N Z V C

-	*	*	Ø	Ø

X = Não é afetado.

N = Set se o bit mais significativo for setado. Reset se o bit mais significativo for resetado.

Z = Set se o resultado for zero. Reset em caso contrário.

V = Permanece zerado.

C = Permanece zerado.

Campo do Dado Imediato

Se o dado imediato for um byte, será utilizado o byte menos significativo do próximo word.

Se o dado imediato for um word, será utilizado o word imediato à instrução.

Se o dado imediato for um long word, serão utilizados os dois próximos words depois da instrução.

ORI TO CCR — Lógica ou imediata com o registrador de código de condição (Flags).

Resumo : Origem (dado) v CCR⟹ CCR

Sintaxe do Assembler : ORI #<data>, CCR

Tamanho : Byte

Descrição : Executa a lógica "ou" entre uma constante de 8 bits (dado imediato) e o registrador de código de condição. Esta instrução permite "setar" or flags.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Imediato							

Flags : X N Z V C

X	N	Z	V	C
*	*	*	*	*

X = Set se o bit 4 do operando imediato for nível lógico 1. Não é alterado se o bit 4 for zero.

N = Set se o bit 3 do operando imediato for nível lógico 1. Não é alterado se o bit 3 for zero.

Z = Set se o bit 2 do operando imediato for nível lógico 1. Não é alterado se o bit 2 for zero.

V = Set se o bit 1 do operando imediato for nível lógico 1. Não é alterado se o bit 1 for zero.

C = Set se o bit Ø do operando imediato for nível lógico 1. Não é alterado se o bit Ø for zero.

ORI TO SR Lógica ou entre um operando imediato e o SR (Instrução Privilegiada)

Resumo : Se estado supervisor então origem VSR ⟹ SR senão TRAP

Sintaxe do Assembler : ORI #<Data>, SR

Tamanho : word

Descrição : Executa uma função lógica ou então um operando imediato e o SR resultado permanecerá no SR.

Formato : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
Dado Imediato (word)															

Flags : X N Z V C

*	*	*	*	*

X = Set se o bit 4 do operando for nível lógico 1. Não será alterado se o bit 4 do operando for nível Ø.

N = Set se o bit 3 do operando for nível lógico 1. Não será alterado se o bit 3 do operando for nível Ø.

Z = Set se o bit 2 do operando for nível lógico 1. Não será alterado se o bit 2 do operando for nível Ø.

V = Set se o bit 1 do operando for nível lógico 1. Não será alterado se o bit 1 do operando for nível Ø.

C = Set se o bit Ø do operando for nível lógico 1. Não será alterado se o bit Ø do operando for nível Ø.

P E A Salva um endereço efetivo

Resumo : SP-4 → SP; EA → (SP)

Sintaxe do Assembler : PEA <ea>

Tamanho : long

Descrição : O endereço efetivo é computado e salvo no stack. Um long word que forma um endereço é colocado no topo do stack.

Formato da Instrução: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	1	0	0	1	0	0	0	0	1	Endereço Efetivo Modo	Endereço Efetivo Registrador

Endereço efetivo = Especifica o endereço a ser salvo no stack. Os modos permitidos estão na tabela 1.25.

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
-(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011

Tabela 1.25 - Modos de endereçamento.

Flags : Não são afetados.

RESET Reset dos dispositivos externos (Instrução Privilegiada)

Resumo : Se estado supervisor então impõe um pulso de nível Ø na linha de reset, senão TRAP.

Sintaxe do Assembler : RESET

Tamanho : Não há

Descrição : A linha de reset é levada a nível Ø durante 124 pulsos de clock, causando um reset geral em todos os periféricos que estejam ligados a esta linha. O processador continua seu trabalho normalmente. Nenhuma parte interna do processador sofrerá qualquer alteração. Esta instrução é um reset **externo**.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

Flags : Não são afetados.

ROL, ROR Rotação (sem extensão)

Resumo : Destino será relacionado em n bits à esquerda ou direita ⟹ Destino.

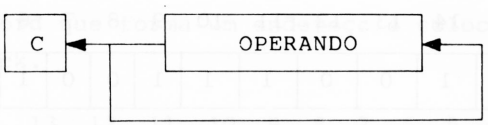
Sintaxe do Asembler : Rod Dx,Dy
Rod #<data>,Dy
Rod <ea>
Onde d é a direção L ou R

Tamanho : Byte, word e long word

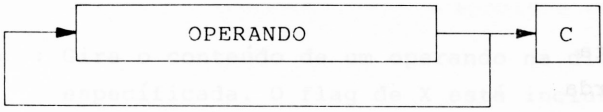
Descrição : Gira o conteúdo de um operando na direção (L ou R) especificada. O flag x não é incluso na rotação. O número de rotações podem ser especificadas de duas formas:

1. Imediato: O número de contagem é especificado na própria instrução. A faixa é de 1 a 8 deslocamentos.
2. Registrador: O número de contagem é especificado num registrador de dados.

Se for utilizado como operando, um endereço de memória, só poderá ser executado um giro por vez, e o tamanho está restrito a um word.
Para a instrução ROL o operando é rotacionado à esquerda. O bit mais significativo e colocado no flag C e no bit menos significativo ao mesmo tempo:



Para a instrução ROR o operando é rotacionado à direita. O bit menos significativo é colocado no flag C e no bit mais significativo ao mesmo tempo:



Formato da: (Para rotação em registrador)
Instrução

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Registrador de contagem			DR	Tamanho		I/R	1	1	Registrador		

DR = especifica a direção
- Ø = direita
- 1 = esquerda

Tamanho = especifica o tamanho do operando
- 00 = byte
- 01 = word
- 10 = long

I/R = especifica o tipo de contagem de giros.
- Ø = contagem de giros imediato
- 1 = contagem de giros através de registrador

Registrador = especifica o registrador de dados que contém a informação, que será rotacionada no seu conteúdo.

Registrador de Contagem:

Se I/R = Ø — A contagem dos giros são especificadas neste campo. Os valores entre Ø e 7 representam os giros entre 1 e 8.

Se I/R = 1 — A contagem dos giros (em módulo 64) estão contidos no registrador de dados especificado.

Formato da: (Para rotação na memória)
Instrução

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	DR	1	1	Endereço Efetivo Modo			Registrador		

DR = direção
Ø = direita
1 = esquerda

Endereço Efetivo = especifica o operando a ser rotacionado. Os modos permitidos estão na tabela 1.26.

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	—	—	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	—	—
(An)+	011	reg. number:An			
-(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	—
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	—

Tabela 1.26

Flags : X N Z V C

-	*	*	Ø	*

X = Não é afetado.
N = Set se o bit mais significativo do resultado for nível 1. Será resetado em caso contrário.
Z = Set se o resultado for zero. Reset se o resultado for diferente de zero.
V = Permanece zerado.
C = Set de acordo com o último bit que for girado para fora do operando.

ROXL, ROXR Rotação (com extensão)

Resumo : Destino será girado em n bits através do flag X à esquerda ou direita ⟹ Destino

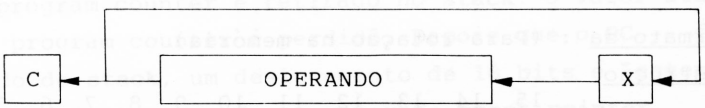
Sintaxe do Assembler : RoXd Dx,Dy
RoXd #<DATA>,Dy
RoXd <ea>

Tamanho : Byte, word e long

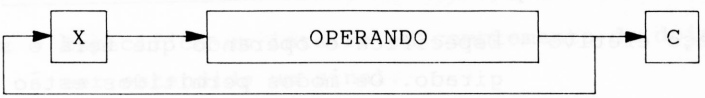
Descrição : Gira o conteúdo de um operando na direção (L ou R) especificada. O flag de X está incluso na rotação. O número de rotações pode ser especificado de duas formas:

1. Imediato = O número de contagem é especificado na própria instrução. A faixa é de 1 a 8 deslocamentos.
2. Registrador= O número de contagem é especificado num registrador de dados.

Se for utilizado, como operando, um endereço de memória, só poderá ser executado um giro por vez e o tamanho está restrito a um word.
Para a instrução ROL o operando é girado à esquerda. O bit mais significativo é colocado no flag C e no flag X e o valor anterior do flag X será colocado no bit menos significativo.



Para a instrução ROR, o operando é girado à direita. O bit menos significativo é colocado no flag C e X e o valor anterior do flag X é colocado no bit mais significativo.



Formato da Instrução : (Para rotação em registrador)

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Registrador de contagem	DR	Tamanho	I/R	1	0	Registrador

DR = Especifica a direção.

Tamanho : 00 = byte
01 = word
10 = long word

I/R⇒ Especifica o tipo de contagem dos giros:
Ø = Contagem de giros imediato
1 = Contagem de giros através de registrador

Registrador = Especifica o registrador de dados, que contém a informação, a qual será girada no seu conteúdo.

Registrador de contagem:

Se I/R = Ø⇒ A contagem dos giros são especificadas neste campo. Os valores entre Ø e 7 representam os giros entre 1 e 8.

Se I/R = 1⇒ A contagem dos giros (em módulo) estão contidos no registrador de dados especificado.

Formato da Instrução : (Para rotação na memória)

15	14	13	12	11	10	9	8	7	6	5 4 3 2 1 0
1	1	1	0	0	1	0	DR	1	1	Endereço Efetivo Modo Registrador

DR ⇒ Especifica a direção
DR = Ø —> direita
DR = 1 —> esquerda

Endereço efetivo = Especifica o operando que terá o seu conteúdo girado. Os modos permitidos estão na tabela 1.26.

Flags : X N Z V C

*	*	*	Ø	*

X = Set ou Reset de acordo com o último bit é girado para fora do operando.

N = Set se o bit mais significativo do resultado for nível 1. Permanece resetado se o bit for nível Ø.

Z = Set se o resultado for zero. Reset se o resultado for diferente de zero.

V = Permanece zerado.

C = Igual ao X.

R T D

Retorno, e realoca parâmetros
(Exclusivo do 68010 e 68012)

Resumo : (SP) → PC; SP+4+d → SP

Sintaxe do Assembler : RTD #<deslocamento>

Tamanho : Não há

Descrição : O program counter é retirado no **stack**. O valor atual do program counter é perdido. Depois que o PC for lido do **stack**, um deslocamento de 16 bits é estendido em sinal e somado ao valor do **stack pointer.**

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0
Deslocamento															

Deslocamento = Especifica um inteiro em complemento de dois ao ser estendido um sinal.

Flags : Não são afetados.

R T E — Retorno de exceção (Instrução Privilegiada)

Resumo : Se supervisor então (SP)→SR; SP+2→SP; (SP)→PC; SP+4→SP. Restabelece o estado. Se não supervisor →trap.

Sintaxe do Assembler : RTE

Tamanho : não há

Descrição : As informações, que estão salvas no topo do stack, são relocadas para dentro do processador. O campo de formação do stack é examinado para determinar quantas informações devem ser restabelecidas.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

Formato do offset word: (no quadro do stack)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Formato				Ø	Ø	Vector offset									

Formato = Estes 4 bits determinam a quantidade de informações a serem restabelecidas pelo processador.

0000 Formato curto, somente quatro words são removidas do top do stack. O SR e o PC são retirados do stack.

1000 Somente no 68010 e 68012 é possível este formato, pois o mesmo permite remover 29 words do top do stack.

Se for tentado, qualquer outro formato ocasionará um erro de exceção.

R T R — Retorno e Restabelece os flags

Resumo : (SP)→CCR; SP+2→SP; (SP)→PC; SP+4→SP

Sintaxe do Assembler : RTR

Tamanho : Não há

Descrição : O CCR e o PC são retirados do top do stack. O CCR e o PC antigo são perdidos. A porção supervisor do SR não é afetada.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

Flags : São alterados de acordo com o word que foi retirado do stack.

R T S

Retorno de uma sub-rotina

Resumo : (SP) $\longrightarrow$ PC; SP+4 $\Longrightarrow$ SP

Sintaxe do Assembler : RTS

Tamanho : Não há

Descrição : O PC é retirado do top do stack. O PC antigo é perdido.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

Flags : Não são afetados.

S B C D

Subtração decimal com extensão

Resumo : $Destino_{10}$ - $Origem_{10}$ - x $\Longrightarrow$ Destino

Sintaxe do : SBCD Dx,Dy
Assembler SBCD - (Ax), -(Ay)

Tamanho : Byte

Descrição : Subtrai o operando origem do operando destino com o bit de extensão e guarda o resultado no destino. Esta subtração é realizada na notação decimal.
O operando pode ser acessado de duas formas diferentes.

1 - Registrador de dados com registrador de dados.
2 - Memória com memória. Sendo que o único modo de endereçamento permitido é o pré-decrementado.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Registrador Dy/Ay			1	0	0	0	0	R/M	Registrador Dx/Ax		

R/M ⇒ Especifica se a operação é entre registrador ou memória.
R/M = Ø ⟶ registrador
R/M = 1 ⟶ memória

Registrador Dy/Ay ⇒ Especifica o operando destino.
Registrador Dx/Ax ⇒ Especifica o operando origem.

Flags : X N Z V C

X	N	Z	V	C
*	U	*	U	*

X = Operação igual ao do C.
N = Indefinido.
Z = Resetado se o resultado não for zero. Permanece inalterado se o resultado for igual a zero.
V = Indefinido.
C = Set se um "empresta um" acontece. Reset em caso contrário.

Scc Impor nível lógico de acordo com a condição **cc**

Resumo : Se condição cc verdadeira então 1s → Destino senão Øs → Destino.

Sintaxe do Assembler : Scc <ea>

Tamanho : Byte

Descrição : O especificado flag (códigos de condição) é testado. Se a condição for verdadeira, o byte especificado pelo operando <ea> recebe o valor $FF. Se não for verdadeira, o conteúdo do operando recebe $ØØ.
O termo cc pode ser:

CC	Carry clear	0100	$\overline{C}$
CS	Carry set	0101	C
EQ	Equal	0111	Z
F	Never True	0001	Ø
GE	Greater or equal	1100	$N.V + \overline{N}.\overline{V}$
GT	Greater then	1110	$N.V.\overline{Z} + \overline{N}.\overline{V}.\overline{Z}$
HI	High	0010	$\overline{C}.\overline{Z}$
LE	Less or equal	1111	$Z+N.\overline{V}+\overline{N}.V$
LS	Low or same	0011	C+Z
LT	Less than	1101	$N.\overline{V}+\overline{N}.V$
MI	Minus	1011	N
NE	Not equal	0110	$\overline{Z}$
PL	Plus	1010	$\overline{N}$
T	Always true	0000	1
VC	Overflow clear	1000	$\overline{V}$
VS	Overflow set	1001	V

Formato da Instrução:

15	14	13	12	11 10 9 8	7	6	5 4 3 2 1 0
0	1	0	1	Condição	1	1	Endereço Efetivo Modo Registrador

Condição = Qualquer uma das dezesseis condições.

Endereço Efetivo = Especifica o operando que receberá os níveis lógicos. Os modos permitidos estão na tabela 1.27.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

Tabela 1.27

Flags : Não são afetados.

STOP Carrega o SR e Stop (Instrução Privilegiada)

Resumo : Se estado supervisor então (dado imediato→SR e Stop) senão TRAP.

Sintaxe do Assembler : STOP #<DATA>

Tamanho : Word

Descrição : Um operando imediato é colocado no **Status Register**. O **Program Counter** é avançado para a próxima instrução. O processador para a busca e execução de instruções. A execução e busta retorna quando acontecer um TRACE, interrupção ou reset externo. O trace vai acontecer se o bit de trace estiver ativado. Se acontecer uma requisição de interrupção e a mesma for de prioridade maior, que a imposta pelo dado imediato, então o processador responderá, em caso contrário a interrupção não tem efeito.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
Dado Imediato															

Dado Imediato = Especifica o dado que será colocado no SR.

Flags : Set de acordo com o dado imediato.

S U B Subtração Binária

Resumo : Destino - Origem ⟹ Destino

Sintaxe do Assembler : SUB <ea>, Dn
SUB Dn, <ea>

Tamanho : Byte, word, long

Descrição : Subtrai um operando origem de um operando destino, colocando então o resultado no operando destino.

Formato :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Registrador			Modo			Endereço Efetivo Modo			Registrador		

Byte	Word	Long	Operação
000	001	010	Dn - <ea> Dn
100	101	110	<ea> - Dn <ea>

Endereço efetivo = Os modos permitidos estão na tabela 1.28.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011

Tabela 1.28

Flags : X N Z V C

*	*	*	*	*

X = Operação idêntica ao carry (C)
N = Set se o resultado for negativo. Reset em caso contrário.
Z = Set se o resultado for zero. Reset em caso contrário.
V = Set se um overflow for gerado. Reset em caso contrário.
C = Set se ocorrer um "empresta um". Reset em caso contrário.

SUBA Subtrai endereço

Resumo : Destino - Origem ⇒ Destino

Sintaxe do Assembler : SUBA <ea>, An

Tamanho : Word, long word

Descrição : Subtrai o operando origem do operando destino e guarda o resultado no registrador de endereço, que realiza a função de destino.

Formato : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	0	0	1	Registrador	Op-Mode	Endereço Efetivo Modo	Endereço Efetivo Registrador

Registrador = Especifica o registrador de endereço (An)

Op-Mode = 011 - word
111 - long

Endereço Efetivo = Especifica o operando origem. Na tabela 1.28, mostra os modos de operação.

Flags : Não são afetados.

SUBI Subtração Imediata

Resumo : Destino - Dado Imediato = Destino

Sintaxe do Assembler : SUBI #<Dado>, <ea>

Tamanho : Byte, word, long

Descrição : Subtrai um dado imediato do conteúdo do operando destino. O destino será sempre um endereço efetivo.

Formato :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Tamanho		Endereço Efetivo Modo			Registrador		
Word data										Byte					
Long Data															

Tamanho = 00 - byte
01 - word
10 - long

Endereço efetivo = Especifica os modos de endereçamentos permitidos. A tabela 1.29 mostra os modos de endereçamentos.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

Tabela 1.29

Flags : X N Z V C

*	*	*	*	*

X = Operação idêntica ao carry (C).
N = Set se o resultado for negativo. Reset em caso contrário.
Z = Set se o resultado for zero. Reset em caso contrário.
V = Set se houve overflow. Reset em caso contrário.
C = Set se ocorrer um "empresta um". Reset em caso contrário.

SUBQ Subtração rápida

Resumo : Destino - Dado Imediato ⟹ Destino

Sintaxe do Assembler : SUBQ #<data>, <ea>

Tamanho : Byte, word ou long

Descrição : Subtrai um dado imediato do conteúdo operando destino. A faixa do dado imediato vai de 1 bit até 8 bits. Para o operando destino a faixa vai de byte até long word.

Formato :

15	14	13	12	11 10 9	8	7 6	5 4 3	2 1 0
0	1	0	1	Dado Imediato	1	Tamanho	Endereço Efetivo Modo	Registrador

Dado imediato = Os três bits indicam dados que vão do número 1 até o número 8.

Tamanho: 00 - byte
01 - word
10 - long

Endereço efetivo = Especifica o operando destino. Os modos permitidos estão na tabela 1.30.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

Tabela 1.30

Flags : X N Z V C

*	*	*	*	*

X = Operação idêntica do carry (C).
N = Set se o resultado for negativo. Reset em caso contrário.
Z = Set se o resultado for zero. Reset em caso contrário.
V = Set se houve overflow. Reset em caso contrário.
C = Set se ocorrer um "empresta um". Reset em caso contrário.

SUBX Subtração com extensão

Resumo : Destino - Origem - X $\Longrightarrow$ Destino

Sintaxe do Assembler : SUBX Dx,Dy
SUBX - (Ax), -(Ay)

Tamanho : Byte, word ou long

Descrição : Subtrai o operando origem e o flag X do conteúdo do operando destino. Coloca o resultado no operando destino. A operação pode ser feita em duas formas:

1 - Registrador de dados com registrador de dados.
2 - Memória com Memória. Neste caso, o único modo do permitido é o pré-decremento.

Formato : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	0	0	1	Registrador Dy/Ay	1	Tamanho	0	0	R/M	Registrador Dx/Ax

Registrador Dy/Ay = Especifica o operando destino.

Tamanho = 00 - byte
01 - word
10 - long

R/M = Ø - Indica operação com registradores de dados Dx e Dy.
1 - Indica operação de memória -(Ay) / -(Ax).

Registrador Dx/Ax = Especifica o operando destino.

Flags : X N Z V C

*	*	*	*	*

X = Operação idêntica ao carry (C)
N = Set se resultado for negativo. Reset em caso contrário.
Z = Set se resultado for zero. Reset em caso contrário.
V = Set se houve overflow. Reset em caso contrário.
C = Set se ocorrer um "empresta um". Reset em caso contrário.

SWAP

Troca metade de um registrador de dados

Resumo : Registrador [31-16] ⟺ Registrador [15-Ø]

Sintaxe do Assembler : SWAP Dn

Tamanho : word

Descrição : Troca metade do registrador (bits Ø a 15) com a outra metade (bits 15-31).

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	Registrador		

Registrador = Especifica o registrador que sofrerá o SWAP.

Flags : X N Z V C

-	*	*	Ø	Ø

X = Não é afetado.

N = Set se o bit mais significativo for setado. Reset se o bit for resetado.

Z = Set se o resultado for zero. Reset se for diferente de zero.

V = C = Permanece zerado.

T A S Testa e Seta um operando

Resumo : Destino testado $\Longrightarrow$ Códigos de condição; 1 $\Longrightarrow$ bit 7 do Destino

Sintaxe do Assembler : TAS <ea>

Tamanho : Byte

Descrição : Testa e seta o operando indicado pelo <ea>. O valor corrente do operando é testado e os flags N e Z são alterados, de acordo com o resultado do teste. A operação é indivisível, ou seja, enquanto o processador está operando a instrução, as linhas de dados e endereços estão ocupadas, não permitindo que outro processador tenha acesso ao endereço <ea> da instru

ção. Esta instrução é utilizada para multiprocessamento, sendo então uma instrução do tipo leitura e escrita modificada (Read - Modify - Write).

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	Endereço Efetivo Modo			Registrador		

Endereço efetivo ⇒ Especifica a localização a ser testada. Os modos permitidos estão na tabela 1.31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

Tabela 1.31 - Modos de endereçamento.

Flags : X N Z V C

-	*	*	Ø	Ø

X = Não é afetado.

N = Set se o bit mais significativo do operando for setado. Resetado em caso contrário.

Z = Set se o operando for zero. Resetado em caso contrário.

V = Permanece zerado.

C = Permanece zerado.

TRAP Armar e Processar exceção

Resumo : SSP-2⇒ SSP; Format/Vector offset⟶(SSP);
SSP-4⇒ 4 ⇒SSP; PC⇒(SSP); SSP-2⇒SSP
SR⇒ (SSP); Vector Address⇒PC

Sintaxe do Assembler : Trap #< Vector >

Tamanho : Não há

Descrição : O processador inicializa um processamento de exceção. O número de #Ø a #15 determina qual o vetor que deverá ser utilizado. O conteúdo do PC e o SR são preservados na pilha do supervisor. O endereço correspondente ao vetor é colocado no PC.

Formato da Instrução:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vetor			

Vetor = Especifica qual o vetor que contém o novo PC a ser carregado.

Flags : Não são afetados.

TRAPV Trap se overflow

Resumo : Se V então TRAP

Sintaxe do Assembler : TRAPV

Tamanho : Não há

Descrição : Se o flag V (overflow) é setado o processador executa a exceção. O número do vetor já está pré-definido como exceção de TRAPV.

Formato :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

Flags : Não são afetados.

T S T Testa um Operando

Resumo : Destino testado $\Longrightarrow$ flags

Sintaxe do Assembler : TST <ea>

Tamanho : Byte, word ou long word

Descrição : Compara o operando com zero. Nenhum resultado é preservado. Entretanto os flags são alterados, de acordo com o resultado da operação de comparação.

Formato :

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	1	0	0	1	0	1	0	Tamanho	Endereço Efetivo Modo	Registrador

Tamanho: 00 = byte
01 = word
10 = long

Endereço efetivo = Especifica o operando que será testado. Na tabela 1.22.

Flags :

X	N	Z	V	C
-	*	*	Ø	Ø

X = Não é afetado.
N = Set se o operando for negativo. Reset em caso contrário.
Z = Set se o operando for zero. Reset em caso contrário.
V = Mantem-se zerado.
C = Mantem-se zerado.

UNLK

Resumo : An $\longrightarrow$ SP; (SP) $\Longrightarrow$ An; SP+2 $\Longrightarrow$ SP

Sintaxe do Assembler : UNLK An

Tamanho : Não há

Descrição : O SP é carregado do registrador de endereço especificado. O registrador de endereço recebe um long word vindo do topo do stack.

Formato :

15	14	13	12	11	10	9	8	7	6	5	4	3	2 1 0
0	1	0	0	1	1	1	0	0	1	0	1	1	Registrador

Registrador = Especifica o registrador de endereço a ser utilizado.

Flags : Não são afetados.

CAP. 2

LINGUAGEM ASSEMBLER

2.1 Introdução

O objetivo deste capítulo é dar ao leitor uma visão geral sobre a linguagem assembly da família 68000, bem como, as técnicas básicas de fluxograma e alguns programas exemplos.

O programa em assembly a princípio pode assustar, mas mediante um estudo criterioso vamos sentir o potencial desta linguagem.

Uma passagem importante para um bom desenvolvimento de um programa é a elaboração do fluxograma. Muitos programadores não gostam de elaborar o fluxograma de seu programa, dizendo ser cansativo e achar que construindo o programa diretamente, da idéia básica para a instrução seria mais rápido. Isto é uma falsa concepção, pois quando vamos construir um sistema, ou até mesmo um programa, é importante elaborar o fluxograma, pois o mesmo nos permitirá ter uma idéia global do algoritmo, que está sendo desenvolvido, bem como descobrir problemas, falhas que o mesmo venha a ter.

Não há uma regra básica para desenvolver um fluxograma. Tudo depende de treino, da análise, do que se deseja que o programa execute. Poderíamos definir a elaboração de um programa em 8 fases:

1 - Estudo do problema, analisar as necessidades, procurar juntar todas as variáveis, escrever se possível, todo o conjunto de requisitos que o programa deverá ter.

2 - Construir um fluxograma sequencial, ou seja, passo a passo tudo o que deverá o programa executar. Este fluxograma irá sofrer muitas mudanças, até chegar a uma idéia final. Este fluxograma será global e superficial.

3 - Construir fluxogramas detalhados a partir do fluxograma básico. Esta fase será um detalhamento de cada bloco do fluxograma anterior.

4 - Montagem das instruções a partir dos fluxogramas detalhados. Nesta fase, precisamos lançar mão de um editor de texto e um computador. Para a edição do programa, é necessário respeitar certas normas utilizadas na programação em assembly. Estas normas vamos abordar logo à frente.

5 - Trabalho de compilação do programa que está em "assembly" para códigos hexadecimais. Se o compilador, chamado normalmente de ASSEMBLER, encontrar algum erro de sintaxe, será anunciado. Se isto ocorrer, o programador deverá recorrer ao editor de texto para as devidas correções.

6 - Após a compilação vem a operação de ligação (LINKER). Nesta fase, o programa que estava em código objeto hexadecimal não executável, passa agora a código objeto hexadecimal executável.

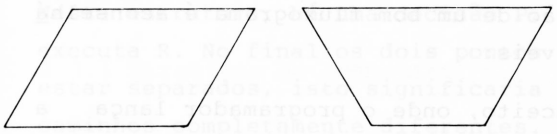
7 - Nesta fase, é feita a depuração do programa. Existem algumas ferramentas de software, para ajudar o programador a analisar o programa e descobrir os problemas.

8 - Última fase, não menos importante que as anteriores, onde o programador deverá documentar todo o trabalho, para futura alteração e também permitir que outras utilizem este programa.

2.2 Fluxograma

Um fluxograma (Flow chart) é uma representação gráfica, utilizada para definir, analisar, ou solucionar um problema.

Os símbolos mais comuns são:

SÍMBOLO	DESIGNAÇÕES
	Entrada ou Saída

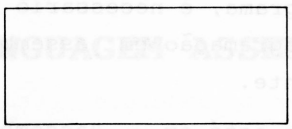
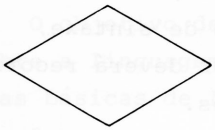
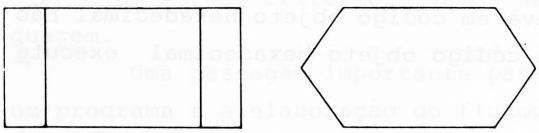
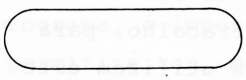
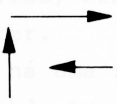
SÍMBOLO	DESIGNAÇÕES
	Processamento
	Decisão
	Sub-rotinas
	Ponto de ligação
	Terminal (início ou fim)
	Sentidos da seqüência do programa

Figura 2.1 - Figuras de fluxograma.

A partir dos símbolos da figura 2.1 é que iremos construir os fluxogramas. No projeto de um programa, a elaboração de um fluxograma pormenorizado é compreensível para a solução do problema envolvido, não é normalmente conseguida logo na primeira tentativa. Para a elaboração de um bom fluxograma é aconselhável construir 3 diferentes níveis:

1 - Fluxograma a nível de conceito, onde o programador lança a idéia.

2 - Fluxograma a nível de algoritmo, onde o programador define a estratégia de solução do programa.

3 - Fluxograma a nível de instrução, onde o programdor ataca a parte final do programa.

2.2.1 Estruturas Básicas

Os elementos dos fluxogramas da figura 2.1 admitem 3 combinações básicas:

1 - Figura seqüência (SEQUENCE)
2 - Figura se-então-senão (IF-THEN-ELSE)
3 - Figura faça enquanto (DO-WHILE)

Quando temos uma instrução após a outra, há o que se chama estrutura aberta, pois o processamento especificado nela é executado apenas uma vez.

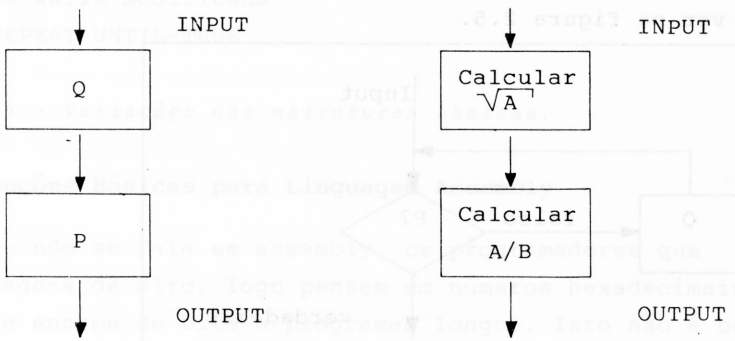


Figura 2.2 - Estrutura "seqüência".

Na figura 2.2, podemos observar a estrutura chamada seqüência. Primeiro calcula-se Q em seguida P e assim por diante, se caso houvesse mais itens.

Na figura 2.3, há a estrutura if-then-else, que também é uma estrutura aberta, pois o bloco é executado apenas uma vez: Nesta estrutura, há uma decisão P. Se P então executa Q senão executa R. No final os dois pontos estão juntos como poderiam estar separados, isto significaria que o programa seguiria dois caminhos completamente diferentes.

A última figura é a Do-While que está demonstrada na figura 2.4. Na figura, pode-se ver que "faça Q enquanto P for falso".

Esta estrutura é fechada, pois permanece em Q, até que P seja satisfeita.

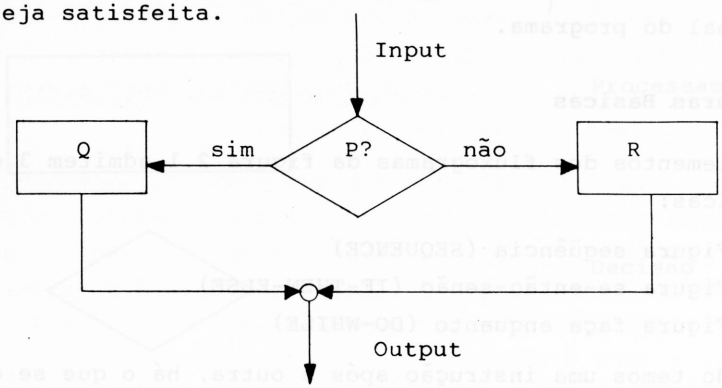


Figura 2.3 - Estrutura if-then-else.

Estas estruturas permitem algumas variações, conforme podemos ver na figura 2.5.

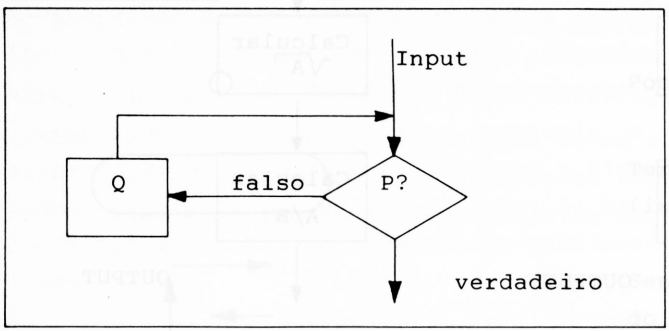


Figura 2.4 - Estrutura Do-while.

Dependendo do fluxograma, outras variações são possíveis, mas se for analisar a fundo, sempre estaremos amarrados nas 3 estruturas básicas.

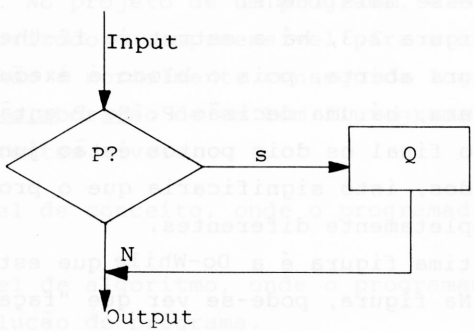
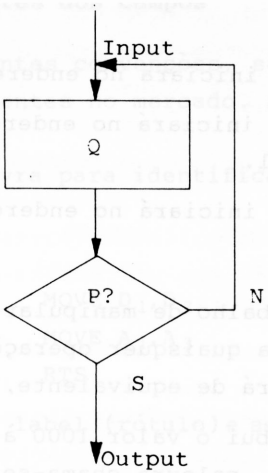


Figura 2.5 - If-them-else modificada.



Do-while modificada
REPEAT-UNTIL-TRUE

Figura 2.5 - Variações das estruturas básicas.

2.3 Convenções Básicas para Linguagem Assembly

Quando se fala em assembly, os programadores que utilizam linguagens de alto, logo pensam em números hexadecimais, uma quantidade enorme de bits e programas longos. Isto não é bem verdade, pois quando trabalhamos com assembly, existe o que chamamos de pseudo-instruções ou instruções diretivas, há também as macroinstruções, que são poderosas ferramentas para a programação.

Como pôde ser visto no capítulo 1, na família dos processadores 68000, exsite uma gama de instruções muito poderosas, sendo que algumas aproximam-se das instruções de linguagem de alto nível.

2.3.1 Diretivas do Assembler

Quando vamos iniciar um programa, é necessário indicar o início do mesmo, como também a origem da área da montagem do programa. A instrução diretiva utilizada para este fim é o ORG.

Exemplos:

ORG 2000 O programa iniciará no endereço 2000 decimal

ORG $2000 O programa iniciará no endereço 2000 em hexadecimal.

ORG TESTE O programa iniciará no endereço simbólico TESTE.

Para simplificar o trabalho de manipular endereços, constantes, dados que servirão para quaisquer operações, devemos utilizar uma palavra que servirá de equivalente, por exemplo:

DADO EQU 1000 Atribui o valor 1000 à palavra dado. Esta palavra chama-se "Label" ou rótulo em português.

VALOR EQU $5000 Atribui o valor 5000 em hexa à palavra valor.

Quando você precisa reservar uma área de memória para qualquer aplicação, deve-se utilizar a diretiva DS.
Exemplo:

AREA DS 200 Reserva área de 200 posições ou bytes, a partir do endereço área.

Quando você precisa definir uma constante ou uma String na memória, deve-se utilizar a diretiva DC.
Exemplos:

VALOR DC.B $FF Coloca FF no endereço valor

VALOR1 DC.W $2222 Coloca o valor 2222 no endereço valor 1.

PALAVRA DC 'TESTE' Coloca a partir do endereço "palavra" os caracteres T, E, S, T, E, em notação ASCII.

Quando terminar o programa, é necessário colocar a diretiva "END"

2.3.2 Delimitadores dos Campos

As seguintes convenções, são utilizadas pela maioria dos assemblers existentes no mercado. São elas:

Label → Uma palavra para identificar o início de uma rotina ou sub-rotina.
Exemplo:

```
INÍCIO    MOVE D1,D2
          MOVE A1,A3
          RTS
```

Entre o label (rótulo) e as instruções deve ter no mínimo um espaço.

O label precisa iniciar com uma letra, o restante pode ser símbolos de números, e pode ter no máximo sete caracteres.

Espaços → Devem ser utilizados após um label, entre o código da instrução e o operando. Espaços a mais são ignorados. Não utilizar espaço no meio de um rótulo.
Exemplo:

```
TESTE MOVE  D2,D4
      ↓     ↓
    espaço espaço
```

Ponto → Utilizar imediatamente após o código da instrução, para definir o comprimento do operando.

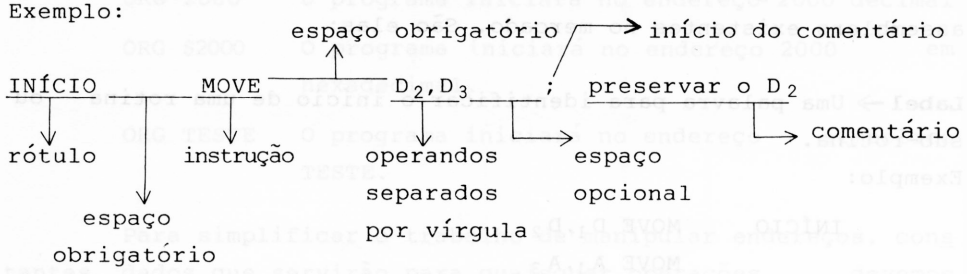
.B → byte (8 bits)
.W → word (16 bits)
.L → long word (32 bits)

Se nada for especificado, é assumido pelo assembler o .W (word).

Vírgula → Uma vírgula deve ser utilizada, para separar o 1º operando de um segundo operando.
Exemplo:

```
MOVE D2,D4
MOVE A1,A2
```

Comentário→ Após cada instrução, pode-se colocar um comentário. Para tal, deve-se colocar um ponto e vírgula (;).
Exemplo:



Expressões - Pode-se utilizar para definir "labels", endereços simbólicos ou constantes simbólicas, as seguintes expressões.

1 - Rótulo simples: TESTE, VALOR ___ DOIS
2 - Rótulo com extensão: VALOR ___ DOIS + 5, TESTE-$ABCD

2.3.3 Definições de Constantes

Pode-se utilizar como operandos constantes tais como:

Decimal : 100, -1, -50, +2000
Hexadecimal : $200, $0F, $1001FB, $ABCDEF00
ASCII : 'TESTE', 'A', 'MICRO'

É importante observar que todos os valores utilizandos quando forem convertidos em binário, não podem ultrapassar 32 bits.

Utilizar a forma decimal apenas para valores pequenos, por exemplo:

100, 200, 2000, -200 e etc.

Valores elevados podem criar problemas quando forem traduzir o problema, usando o assembler.

2.3.4 Simbologia e Notação Utilizada

As simbologias mais utilizadas são:

Dn ⟶ defini os registradores de dados
An ⟶ defini os registradores de endereço

Xn,Rn,Rx —> defini qualquer registro.

d —> defini qualquer valor ou dado constante. Pode ser binário, hexa ou ASCII.

dI —> dado ou um valor de até 1 bit por exemplo: d8 —> dado de 8 bits.

#d ou #di —> dado imediato.

rótulo —> endereço simbólico para definir o início de uma sub-rotina, ou uma posição de memória de dados.

(rótulo) —> endereço simbólico utilizado como um operando para as instruções de desvio.

(ea) —> defini um endereço efetivo, podendo ser: Dn, An, (An), (An)+, -(An), d16(An), d8(An,Ri), absoluto curto, absoluto longo, d16(PC), d8(PC) ou #d.

2.4 Exemplos de Rotinas

Depois de uma análise em cada instrução, no capítulo anterior, e um estudo sobre o assembler neste capítulo, vamos desenvolver alguns exemplos de programas.

2.4.1 Exercícios Resolvidos

1 - Calcular o número de caracteres que há numa seqüência (STRING).

Vamos admitir que tenhamos na memória, a partir do rótulo (Label) "String", uma mensagem e deseja-se saber quantos caracteres há até encontrar o "CR" (Carriage Return).

O resultado deverá ficar no registrador D1.

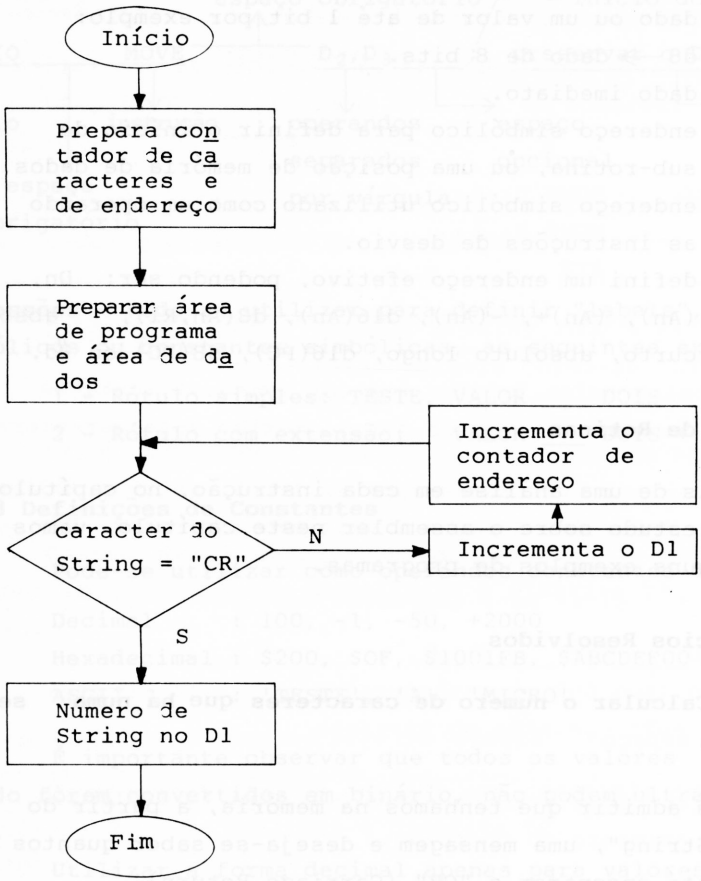
1A) Algoritmo

Início - Determinar um contador para somar o número de caracteres. Este contador irá iniciar com zero.

Loop - Comparar um caracter da string. Se for igual a ØDh ("CR"), então vai para o fim do programa, senão incrementa o contador e retorna ao "loop".

Fim - Valor final deverá estar o registrador D1.

1B) Fluxograma



1C) Programa

```
STRING EQU $5000; Endereço do início do string
PROGRA EQU $2000; Endereço do início do programa
       ORG PROGRA
INÍCIO MOVE A.L STRING,Ao ; Coloca endereço no Ao
       MOVEQ   # Ø, D1    ; Coloca Ø no D1
LOOP   CMPI.B  #$ØD,(Ao)+ ; Compara ØD com ; endereço apontado
                           por Ao e encrementa
       BEQ FIM             ; Se for igual salta para fim
```

```
        ADDQ.W  #1,D1      ; soma 1 ao D1
        BRA  Loop          ; volta a Loop para comparar com ; o pró
                             ximo caracter
FIM  RTS                   ; Retorna à rotina principal pois o núme
                             ro de caracteres está no D1
        ORG     STRING     ; Tabela com o texto
        DC.B     'C'
        DC.B     'O'
        DC.B     'M'
        DC.B     'P'
        DC.B     'U'
        DC.B     'T'
        DC.B     'A'
        DC.B     'D'
        DC.B     'O'
        DC.B     'R'
        DC.B     $ØD
        END INÍCIO
```

2 - Calcular qual o maior valor existente numa série de números

Neste exercício, nós temos números entre ØØ a 99 distribuídos numa série de 1Ø números aleatórios. Descobrir qual o número de maior valor. Esta série começa no endereço "SÉRIE". O resultado deverá ficar no registrador D1.

2A) Algoritmo

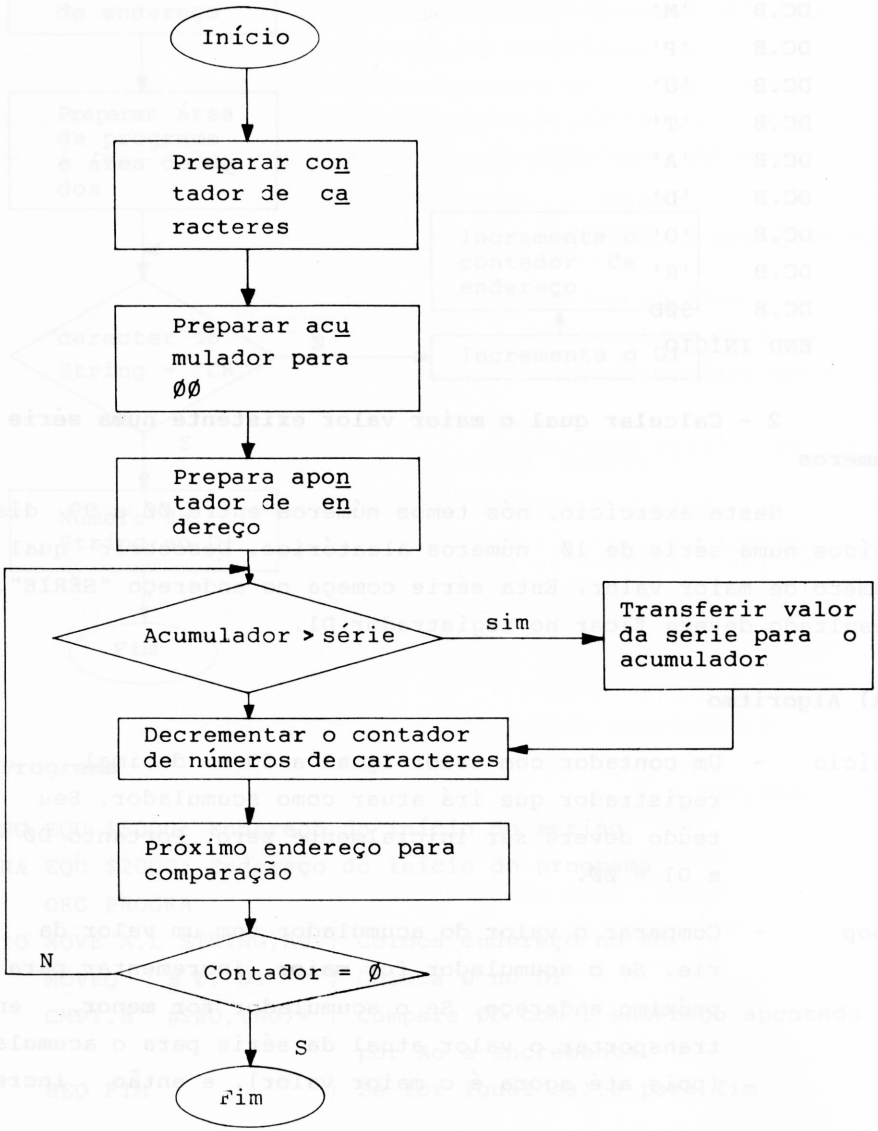
Início - Um contador com valor igual a 50 em decimal. Um registrador que irá atuar como acumulador. Seu conteúdo deverá ser inicialmente zero. Portanto DØ = 1Ø e D1 = ØØ.

Loop - Comparar o valor do acumulador com um valor da série. Se o acumulador for **maior**, incrementar para o próximo endereço. Se o acumulador for **menor**, então transportar o valor atual da série para o acumulador (pois até agora é o maior valor), e então incremen

tar para o próximo endereço.
Após a operação vista, decrementar em uma unidade, o contador de número de caracteres da série. Se o contador chegar a zero então FIM, senão voltar a "loop".

Fim - O maior valor deverá estar o registrador D1.

2B) Fluxograma



2C) Programa

```
SERIE   EQU $2000          ; Endereço do início da série
PROGRA  EQU $1000          ; Endereço do início do programa
        ORG PROGRA
INÍCIO  MOVEA.L SERIE, Ao ;
        MOVEQ  # Ø,D1      ; zerar o acumulador
        MOVEQ  #50,Do      ; preparar o contador
LOOP    MOVE.B (Ao)+,D3    ; transfere o número para D3
        CMP.B  D3,D1       ; comparar D3 com D1
        BCC CONTINUA       ; se acumulador (D1)
                           ; for maior que o número atual  da
                           ; série (D3) então saltar para "continua"
        MOVE.B  D3,D1      ; se acumulador for
                           ; menor que o número atual da série
                           ; então copiar o número
CONTINUA SUBQ.W #1,D0      ; subtrair um do
                           ; contador de caracteres
        BNE LOOP           ; voltar a loop se o
                           ; resultado for maior que zero
FIM     RTS                ; retornar a rotina principal
        ORG     SERIE
        DC.B     '15'
        DC.B     '12'
        DC.B     '50'
        DC.B     '10'
        DC.B     '90'
        DC.B     '17'
        DC.B     '25'
        DC.B     '01'
        DC.B     '21'
        DC.B     '46'
        END     INÍCIO
```

3 - Somar dois valores decimais que constam numa série

Neste exemplo, vamos somar os valores que compõem a série A com a série B. Colocando o resultado na série C.

Exemplo: série A 524532 [11] → 1 byte
série B 112145 [13] → 1 byte
série C 636677 [24] → 1 byte

Admitir que a série é composta de 4 bytes.

3A) Algoritmo

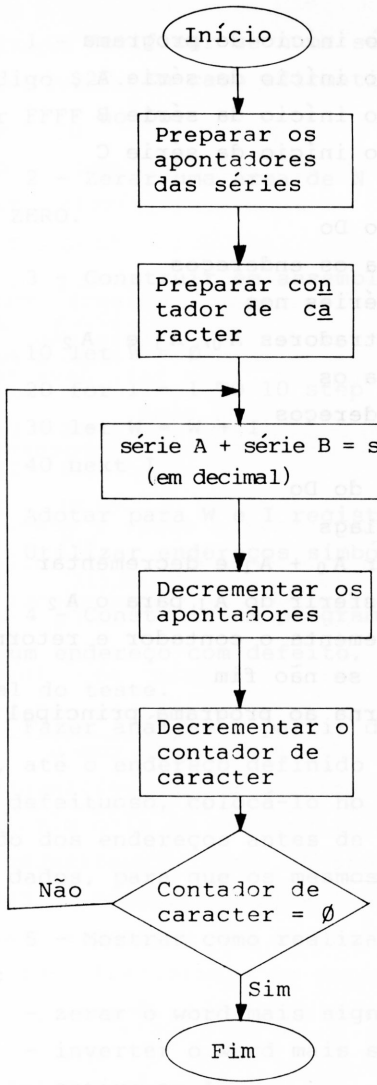
Início - Como serão somados 4 bytes, devemos colocar no DØ=4 para indicar 4 somas.
Criar apontadores para as séries a serem somadas, a saber:

série A → 1ª parcela
série B → 2ª parcela
série C → resultado

Loop - Somar o caracter da série A com o da série B (em decimal), colocar o resultado na série C. Decrementar os apontadores. Decrementar 1 bit o contador de caracter. Se contador = Ø, então FIM senão retornar a "loop".

Fim - O resultado da soma está na série C.

3B) Fluxograma



OBS.: Os bytes menos significativos da série estão nos endereços mais significativo, isto facilita o programa.

sérieA 2000 = 52	sérieB 2004 = 11	sérieC 2008 = 63
2001 = 45	2005 = 21	2009 = 66
2002 = 32	2006 = 45	200A = 77
2003 = 11	2007 = 13	200B = 24

3C) Programa

```
PROGRA  EQU $1000   ; Endereço do início do programa
série A EQU $2000   ; Endereço do início da série A
série B EQU $2004   ; Endereço do início da série B
série C EQU $2008   ; Endereço do início da série C
        ORG PROGRA
INÍCIO  MOVEQ #4,Do ; Coloca 4 no Do
        MOVE.L série A,A0; Coloca os endereços
        MOVE.L série B,A1; das séries nos
        MOVE.L série C,A2; registradores A0, A1 e  A2
        LEA 4(A0),A0; aponta para os
        LEA 4(A1),A1; últimos endereços
        LEA 4(A2),A2;
        SUBQ #1,Do  ; subtrair 1 do Do
        MOVE #Ø,CCR ; zerar os flags
LOOP    ABCD -(A0), -(A1) ; somar A0 + A1 e decrementar
        MOVE.B (A1), (A2) ; transferir do A1 para o A2
        DBF Do, loop      ; decrementa o contador e retorna    a
                            loop se não fim
        RTS               ; retorna ao programa principal
        ORG  série A
        DC.B     '52'
        DC.B     '45'
        DC.B     '32'
        DC.B     '11'
        ORG  série B
        DC.B     '11'
        DC.B     '21'
        DC.B     '45'
        DC.B     '13'
        ORG  série C
        DC.B     '0'
        DC.B     '0'
        DC.B     '0'
        DC.B     '0'
        END  INÍCIO
```

2.4.2 Exercícios para Resolver

1 - Verificar se numa série de N caracteres existe um com código $2F. Em caso afirmativo, zerar o Do em caso negativo colocar FFFF no Do.

2 - Zerar uma área de N bytes, a partir do endereço simbólico ZERO.

3 - Construir em assembler, o programa em BASIC, a seguir:

```
10 let W = Ø
20 for I = 1 TO 10 step 2
30 let W = W + I
40 next I
```

Obs.: Adotar para W e I registradores interno do processador. Utilizar endereços simbólicos para facilitar.

4 - Construir um programa para detectar num bloco de memória, um endereço com defeito, sem alterar o conteúdo da mesma no final do teste.

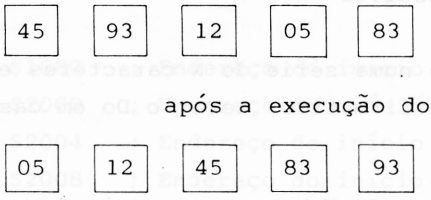
Fazer análise a partir do endereço definido no registrador A_0, até o endereço definido no registrador A_1. Se houver endereço defeituoso, colocá-lo no endereço A_2. Procure salvar o conteúdo dos endereços antes de fazer os testes e depois retornar os dados, para que os mesmos não sejam alterados.

5 - Mostrar como realizar em assembler, as seguintes operações:

- zerar o word mais significativo de Do
- inverter o word mais significativo com o menos significativo em A_0.
- se D5 = AAAABBBB fazer D5 = 00AABB00

6 - Colocar em ordem crescente, uma série de 5 números de ØØ a 99.

Exemplo:



2.5 Linguagem de Alto Nível

Quando forem desenvolvidos sistemas complexos, muitas vezes a linguagem assembly torna-se de difícil utilização, programas muito longos e de difícil depuração. Para tal, deve-se lançar mão de linguagens apropriadas para cada caso.

As linguagens tais como Pascale "C", são largamente utilizadas.

A elaboração do projeto do software segue o mesmo esquema adotado neste capítulo, apenas que no item 5, a compilação será feita não pelo assembler, mas sim pelo compiler apropriado à linguagem utilizada.

O trabalho do compilador será "traduzir" as instruções que estão em alto nível (PASCAL, C, PL/M, etc.), em códigos objetos (Hexadecimal), para em seguida, ser ligado com o LINKER e seguir o processo normal de elaboração e desenvolvimento.

A grande vantagem em usar linguagem de alto nível, é ter programas pequenos durante a edição de fácil visualização e relativa facilidade para desenvolvê-los em comparação com o assembly. Existe uma desvantagem, sendo que programas compilados costumam resultar e programar longos em hexadecimal. Por isso afirmamos que em casos de programas complexos, é que justificam utilizar linguagem de alto nível.

Apenas para exemplificar, se for necessário criar uma sub-rotina para cálculo da raiz quadrada em assembly, é necessário um programa mais ou menos longo, conforme a figura 2.6, enquanto que se for elaborado em Basic, será apenas uma instrução.

Assembler	BASIC
* Calcula a raiz quadrada do número que está na localização $1000 * Coloca o resultado na localização $1002	SQR(A) = B

RAIZ CLR.L D1 ; zerar o D1
LEA $1002,A1 ; prepara apontador
MOVE $1000,D1 ; carrega o D1
MOVE.L D1,D5 ;
DIVU #200,D1 ; Faz divisão por 200
ADQ #2,D1 ; soma 2
Loop MOVE.L D5,D4 ; guarda em D4
DIVU D1,D4 ;
MOVE D4,(A1) ; guarda o quociente em $1002
MULU D4,D4 ;
CMP D4,D5 ; resultado igual ao original?
BEQ.S FIM ;
ADD (A1),D1 ; Se não, somar as duas
LSR #1,D1 ; últimas aproximações e divide
BRA.S Loop ; por 2 e retorna ao loop.
FIM	

Figura 2.6 - Comparação entre o assembly e uma linguagem de alto nível.

CAP. 3

ARQUITETURA DE UM MICROCOMPUTADOR

3.1 Descrição Básica

Neste capítulo, vamos analisar a arquitetura do micro computador, apresentado no volume 1 desta obra.

As considerações básicas relativas ao circuito foram mostradas no volume 1. Será agora abordado o aspecto do software, que poderia ser instalado neste sistema.

A arquitetura foi dividida em 4 partes:

1) **CPU**—> Utilizando um 68000 como unidade central de processamento. Para uma aplicação mais econômica, poder-se-ia utilizar um 68008, com isso iríamos baratear o bus de dados, facilitando o projeto do circuito impresso.

2) **Memória EPROM** —> Neste bloco, deverá ficar armazenado o programa de controle deste microsistema. O objeto de estudo deste capítulo, é dar alguns exemplos de aplicação para este pequeno sistema, mostrando os programas em blocos, para um posterior desenvolvimento por parte do usuário se assim o desejar.

3) **Memória RAM** —> Bloco disponível para armazenar dados ou programas transientes, dependendo da aplicação.

4) **Periférico** —> Neste projeto, foi usada uma típica pastilha periférica da família dos processadores 68000. Um projetista com grande experiência em sistemas digitais, poderá implementar uma pastilha periférica mais popular, adequando melhor o projeto a sua necessidade.

Na figura 3.1, temos o diagrama em bloco do micro sistema de hardware.

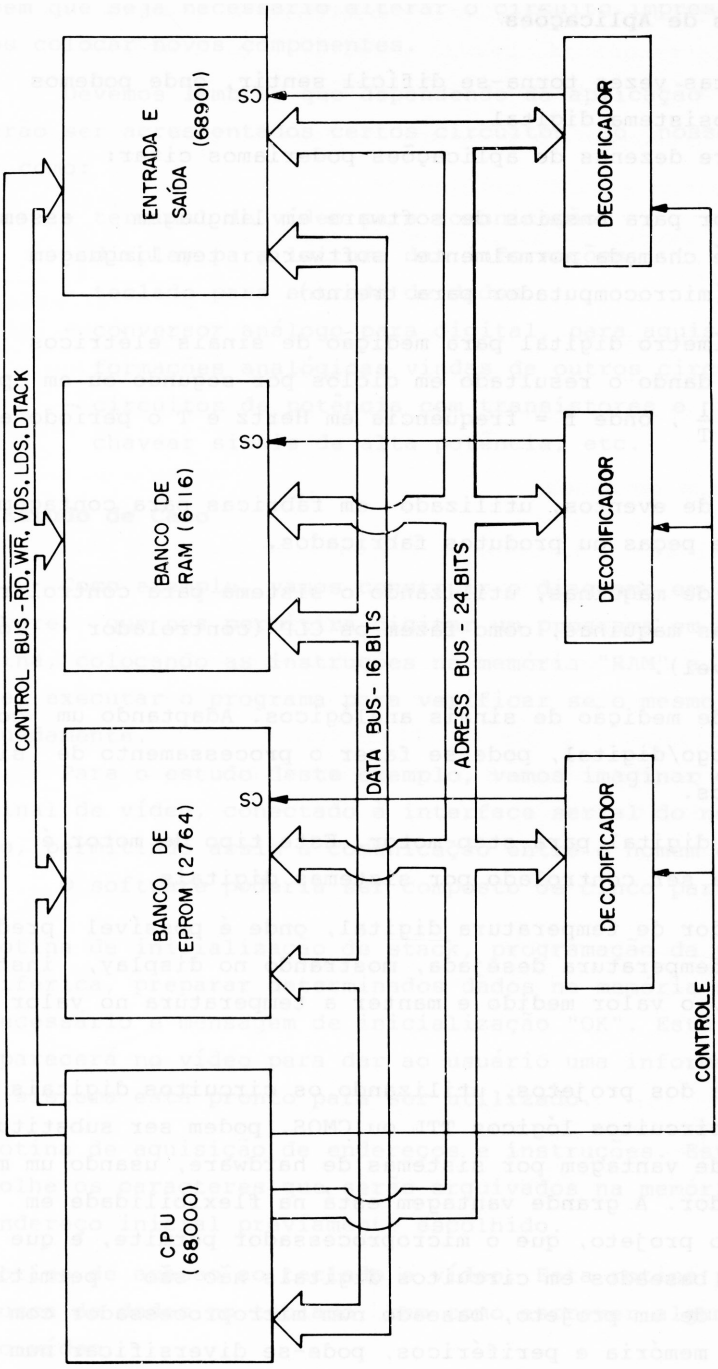


Figura 3.1 - Bloco do sistema.

3.2 Exemplos de Aplicações

Muitas vezes torna-se difícil sentir, onde podemos aplicar um microsistema digital.

Entre dezenas de aplicações poderíamos citar:

1) Computador para ensaios de software em linguagem assembly, ou como é chamada normalmente software, tem linguagem de máquina (microcomputador para treino).

2) Freqüencímetro digital para medição de sinais elétricos repetitivos, dando o resultado em ciclos por segundo ou em períodos. $f = \frac{1}{T}$, onde f = freqüência em Hertz e T o período em segundos.

3) Contador de eventos, utilizado em fábricas para contagem do número de peças ou produtos fabricados.

4) Controle de máquinas, utilizando o sistema para controlar determinadas máquinas, como fazem os CLP (controlador lógico programável).

5) Sistema de medição de sinais analógicos. Adaptando um conversor análogo/digital, pode-se fazer o processamento de sinais analógicos.

6) Controle digital para step-motor. Este tipo de motor é próprio para ser controlado por sistemas digitais.

7) Controlador de temperatura digital, onde é possível predeterminar a temperatura desejada, mostrando no display, instantaneamente, o valor medido e manter a temperatura no valor desejado.

8) A maioria dos projetos, utilizando os circuitos digitais discretos, circuitos lógicos TTL ou CMOS, podem ser substituídos com grande vantagem por sistemas de hardware, usando um microprocessador. A grande vantagem está na flexibilidade em alterações no projeto, que o microprocessador permite, e que nos sistemas baseados em circuitos digitais não são permitidos. A partir de um projeto, baseado num microprocessador com um banco de memória e periféricos, pode-se diversificar num grande leque a sua utilização. Devido ao fato de existir um programa gravado, pode-se alterar o mesmo a qualquer momento,

sem que seja necessário alterar o circuito impresso, retirar ou colocar novos componentes.

Devemos lembrar que dependendo da aplicação desejada, deverão ser acrescentados certos circuitos ao nosso sistema, tais como:

- terminal de vídeo para comunicação.
- display para leitura das informações.
- teclado para entrada de dados.
- conversor análogo para digital, para aquisição de informações analógicas vindas de outros circuitos.
- circuitos de potência com transistores e relés para chavear sinais de alta potência, etc.

3.3 Estudo de Caso

Como exemplo, vamos construir o diagrama em bloco de um software, que nos permitirá digitar um programa em linguagem de máquina, colocando as instruções na memória "RAM" e, posteriormente, executar o programa para verificar se o mesmo funciona adequadamente.

Para o estudo deste exemplo, vamos imaginar que há um terminal de vídeo, conectado à interface serial do nosso equipamento, permitindo assim a comunicação entre o homem e o sistema.

O software poderia ser composto de cinco partes básicas:

1) Rotina de inicialização de stack, programação da pastilha periférica, preparar determinados dados na memória se assim for necessário e mensagem de inicialização "OK". Esta mensagem aparecerá no vídeo para dar ao usuário uma informação de que o sistema está pronto para ser utilizado.
2) Rotina de aquisição de endereços e instruções. Esta rotina recolhe os caracteres que serão arquivados na memória "RAM", no endereço inicial previamente escolhido.
3) Rotina de acesso ao teclado e vídeo. Esta rotina permitirá a busca de dados no teclado, bem como escrever algum caracter no vídeo.

4) Rotina que permite verificar o conteúdo da memória, em blocos ou caracteres únicos. Deverá permitir a alteração de alguma informação, se o usuário assim o desejar.

5) Rotina de execução do programa em duas formas distintas.

5A) Programa com alguns breakpoints.

5B) Programa executará as instruções uma a uma, ou seja, shep a shep.

Um programador experiente poderá criar uma série de outros comandos para este sistema. O nosso objetivo foi criar um programa bem simples para servir como base para o leitor.

3.3.1 Diagrama em Bloco

Na figura 3.2, temos o diagrama em bloco do sistema para ensaios. Esta figura mostra as rotinas num modo global. Na figura 3.3, temos o básico para construção da rotina de recolher o endereço inicial e em seguida, os dados hexadecimais que compõem o programa. Na figura 3.4, temos a rotina que nos permite verificar posteriormente o que foi digitado, e alterar o que venha ser necessário. Na figura 3.5, a rotina para colocar em execução um programa digitado.

Rotina Mestra

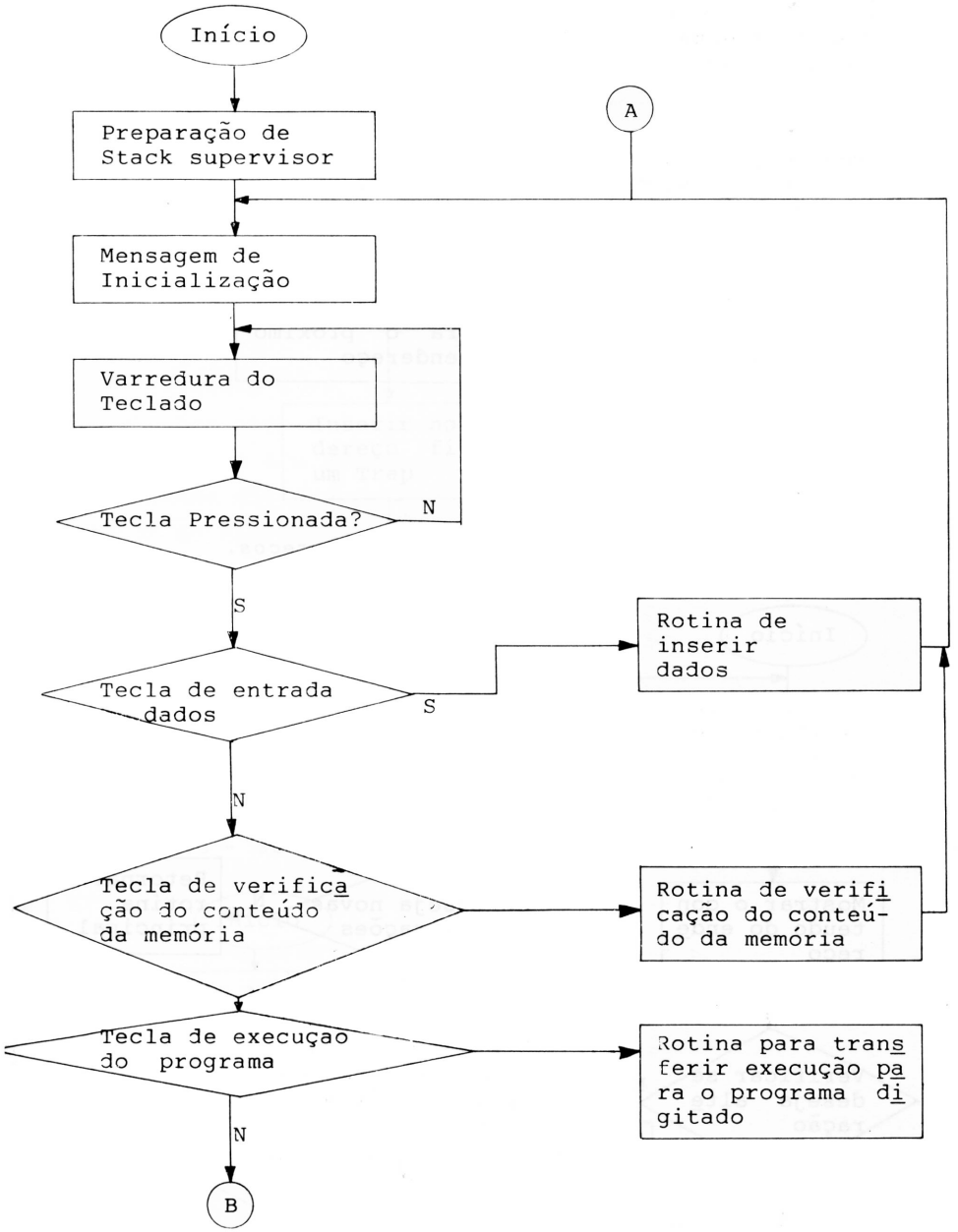


Figura 3.2 - Diagrama Geral.

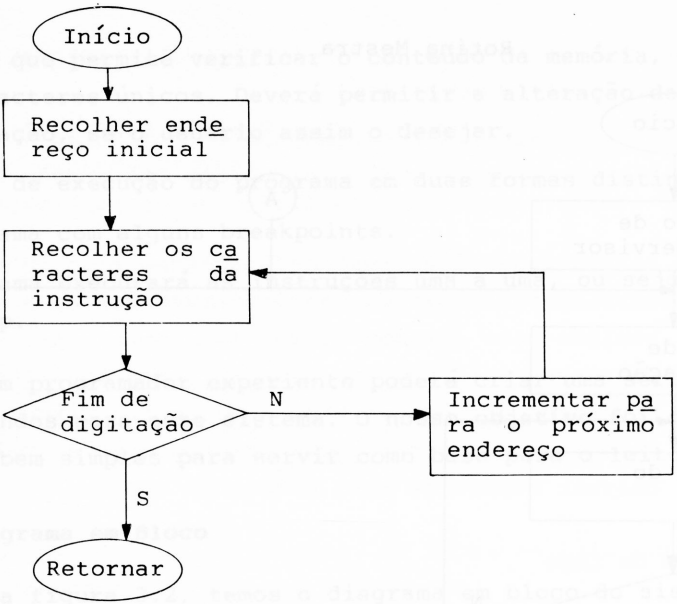


Figura 3.3 → Rotina de inserir os dados e endereços.

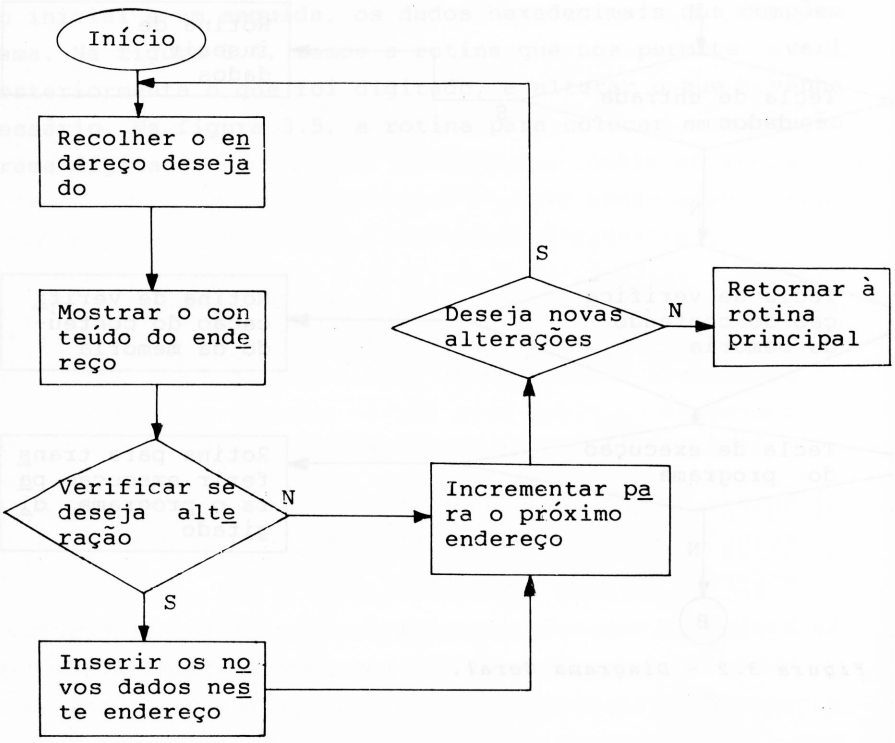


Figura 3.4 → Rotina de alterar o conteúdo da memória.

Figura 3.5 - Rotina para execução do programa digitado.

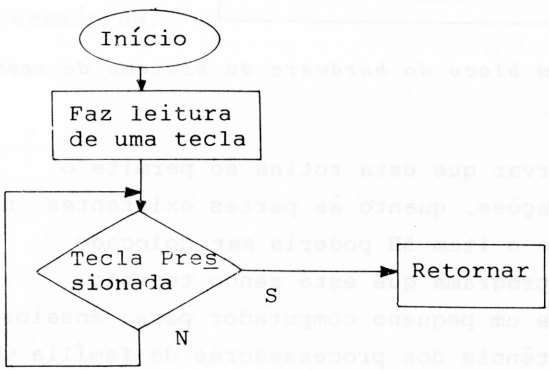


Figura 3.6 - Rotina do trace para execução em shep-by-shep.

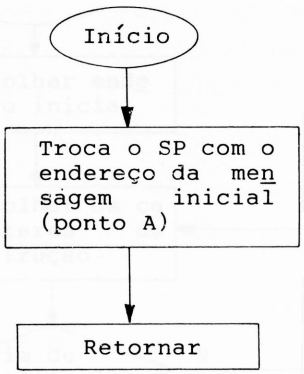


Figura 3.7 - Rotina do Trap para finalizar uma execução.

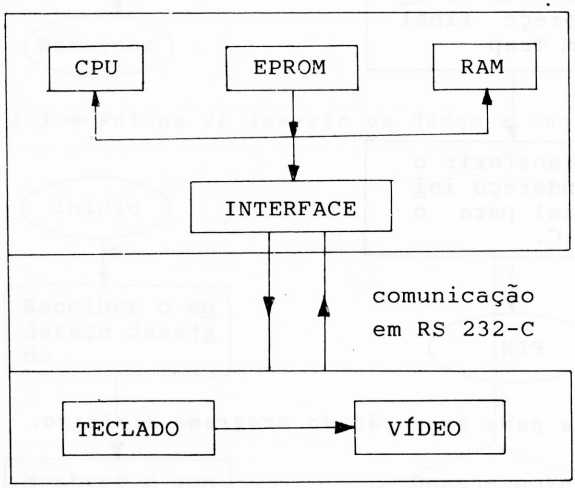


Figura 3.8 - Diagrama em bloco do hardware do sistema de ensaios, em linguagem de máquina.

Importante observar que esta rotina só permite o item 5A, descrito nas observações, quanto às partes existentes neste software. O motivo é que o item 5B poderia ser colocado por software, pelo próprio programa que está sendo testado.

Na construção de um pequeno computador para ensaios, é que se pode sentir a potência dos processadores da família 68000. Devido ao fato de existir na tabela de vetores de exceção, entradas para os mais típicos problemas, que acontecem num sistema com microprocessador, forma o 68000 uma excelente ferramenta pa

ra desenvolvimento.

Os processamentos de exceção tais como, erro de via, instruções ilegais, erro de endereçamento, divisão por zero e etc., facilitam quando da depuração de um programa em desenvolvimento. Por exemplo, se houver no programa um desenvolvimento, uma divisão por zero, acontecerá um salto para o endereço dado pelo vetor nº 5, se anteriormente foi construída uma sub-rotina neste vetor, que informa ao usuário o erro, ficará mais fácil detectar o problema.

3.4 Conclusão

Nesta obra, foi abordado o aspecto software da família 68000, bem como a ferramenta básica para trabalhar com os processadores, que é o assembler. Um breve estudo sobre as figuras de fluxograma foi abordado, para em seguida, fazer uma análise em blocos do hardware do volume 1 desta obra.

Este último capítulo serviu apenas para ilustrar, pois estes processadores abordados nesta obra são muito poderosos, sendo indicados para aplicações onde requer alta velocidade de processamento, acesso a grandes bancos de memória e instruções poderosas para execução de sofisticados sistemas, onde se trabalha com multitarefa e multiusuário.

Este trabalho não é a última palavra sobre a família 68000, mas esperamos ter contribuído com uma parte para informar os estudantes, hobistas e profissionais sobre estes gigantescos processadores.

APÊNDICE A

CONJUNTO DAS INSTRUÇÕES RESUMIDAS

Neste apêndice, nós temos um resumo das instruções do 68000, 68008, 68010 e 68012.

Está estruturado com o word principal e os operandos secundários, quando a instrução exige.

Na tabela A-1, nós temos o "op-code" dos grupos de instruções, que são definidos entre os bits 15 até 12.

Bits 15 through 12	Operation
0000	Bit Manipulation/MOVEP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/Scc/DBcc
0110	Bcc/BSR
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned, Reserved)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate
1111	Coprocessor Interface (MC68020)

Tabela A.1 - Mapa dos códigos de operação.

Na tabela A.2, temos todo o conjunto de possíveis modos de endereçamento.

Address Modes	Mode	Register	Data	Memory	Control	Alterable	Assembler Syntax
Data Register Direct	000	reg. no.	X	–	–	X	Dn
Address Register Direct	001	reg. no.	–	–	–	X	An
Address Register Indirect	010	reg. no.	X	X	X	X	(An)
Address Register Indirect with Postincrement	011	reg. no.	X	X	–	X	(An) +
Address Register Indirect with Predecrement	100	reg. no.	X	X	–	X	– (An)
Address Register Indirect with Displacement	101	reg. no	X	X	X	X	(d_{16},An) or d_{16}(An)
Address Register Indirect with Index	110	reg. no.	X	X	X	X	(d_8,An,Xn) or d_8(An,Xn)
Absolute Short	111	000	X	X	X	X	(xxx).W
Absolute Long	111	001	X	X	X	X	(xxx).L
Program Counter Indirect with Displacement	111	101	X	X	X	–	(d_{16},PC) or d_{16}(PC)
Program Counter Indirect with Index	111	011	X	X	X	–	(d_8,PC,Xn) or d_8(PC,Xn)
Immediate	111	100	X	X	–	–	#<data>

Tabela A.2 - Modos de endereçamento.

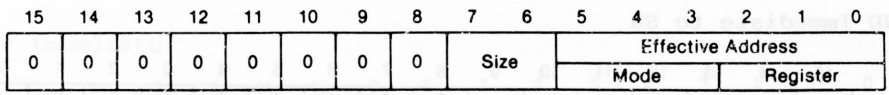
Para as instruções que utilizam condicionais, foi montada a tabela A.3, com os estados lógicos de cada condicional.

Mnemonic	Condition	Encoding	Test
T*	True	0000	1
F*	False	0001	0
HI	High	0010	$\overline{C} \cdot \overline{Z}$
LS	Low or Same	0011	C + Z
CC(HS)	Carry Clear	0100	$\overline{C}$
CS(LO)	Carry Set	0101	C
NE	Not Equal	0110	$\overline{Z}$
EQ	Equal	0111	Z
VC	Overflow Clear	1000	$\overline{V}$
VS	Overflow Set	1001	V
PL	Plus	1010	$\overline{N}$
MI	Minus	1011	N
GE	Greater or Equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	Less Than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	Greater Than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	Less or Equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

Tabela A.3 - Condicionais.

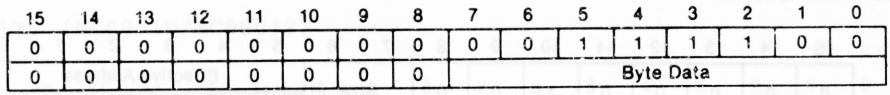
INSTRUÇÕES PADRÕES

OR Immediate

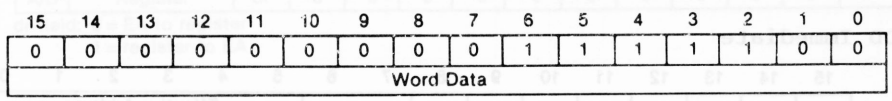


Size field: 00 = byte 01 = word 10 = long

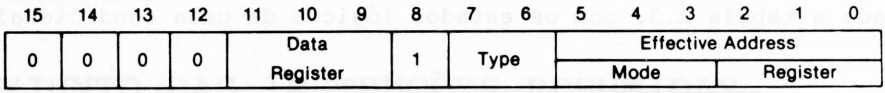
OR Immediate to CCR



OR Immediate to SR

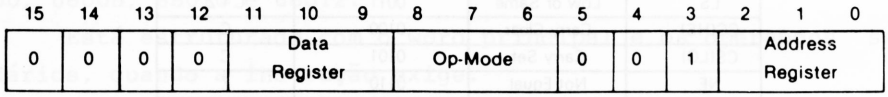


Dynamic Bit



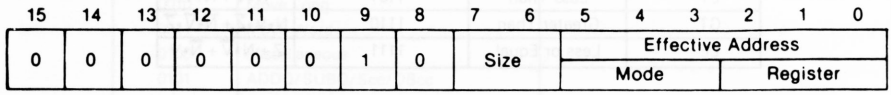
Type field: 00 = TST 10 = CLR
01 = CHG 11 = SET

MOVEP



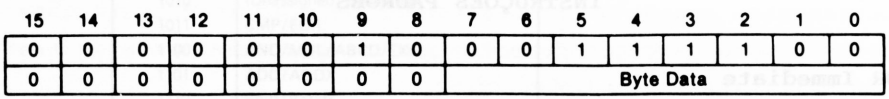
Op-Mode field: 100 = transfer word from memory to register
101 = transfer long from memory to register
110 = transfer word from register to memory
111 = transfer long from register to memory

AND Immediate

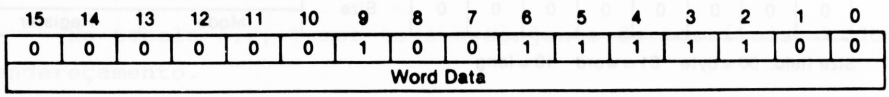


Size field: 00 = byte 01 = word 10 = long

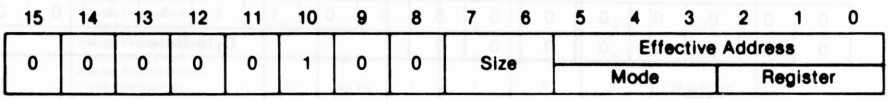
AND Immediate to CCR



AND Immediate to SR

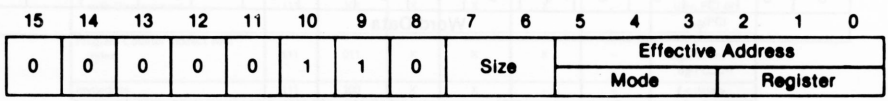


SUB Immediate



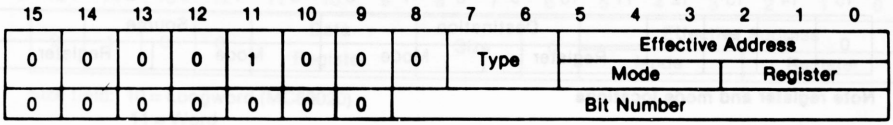
Size field: 00 = byte 01 = word 10 = long

ADD Immediate



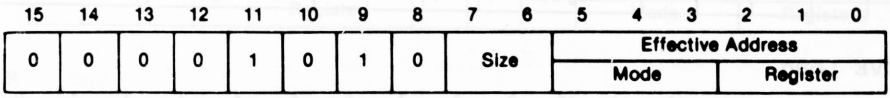
Size field: 00 = byte 01 = word 10 = long

Static Bit



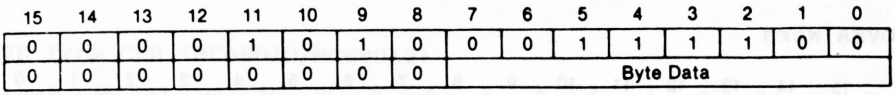
Type field: 00 = TST 10 = CLR
01 = CHG 11 = SET

EOR Immediate

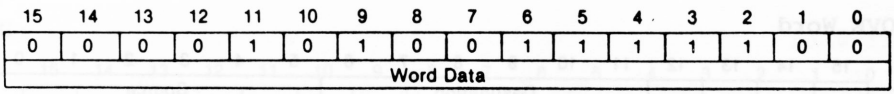


Size field: 00 = byte 01 = word 10 = long

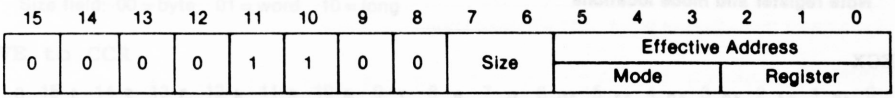
EOR Immediate to CCR



EOR Immediate to SR

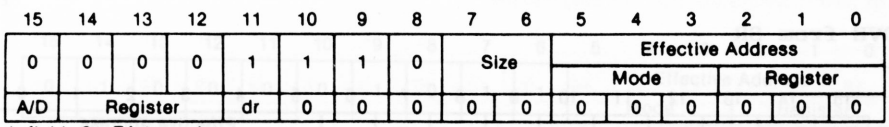


CMP Immediate



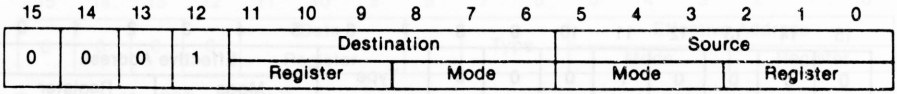
Size field: 00 = byte 01 = word 10 = long

MOVES (MC68010/MC68012)



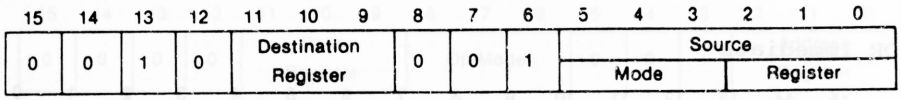
dr field: 0 = EA to register
1 = register to EA

MOVE Byte

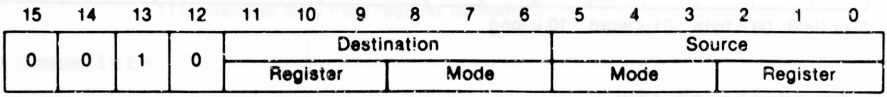


Note register and mode locations

MOVEA Long

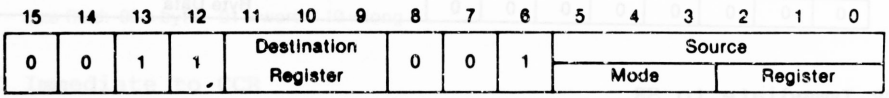


MOVE Long

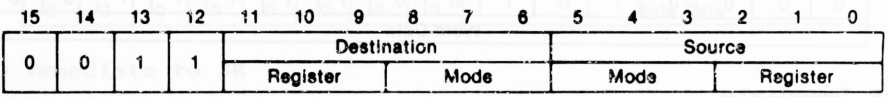


Note register and mode locations

MOVEA Word

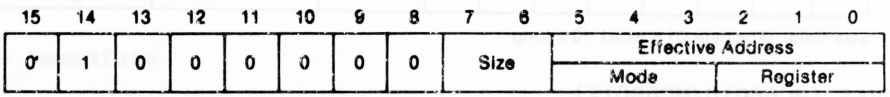


MOVE Word



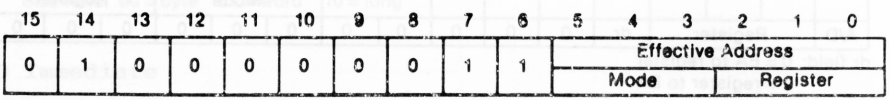
Note register and mode locations

NEGX

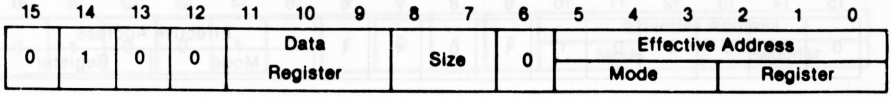


Size field: 00 = byte 01 = word 10 = long

MOVE from SR

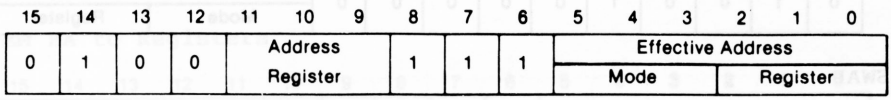


CHK

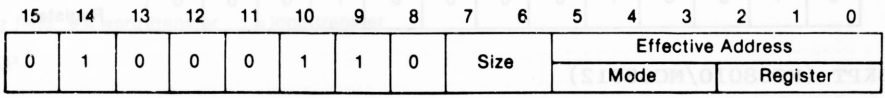


Size field: 10 = Longword (MC68020)
11 = Word

LEA

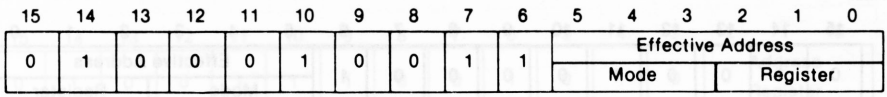


CLR

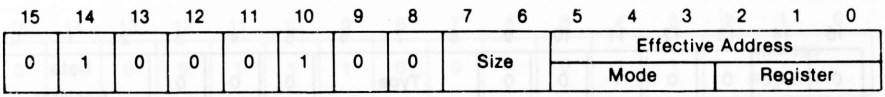


Size field: 00 = byte 01 = word 10 = long

MOVE from CCR (MC68010/MC68012)

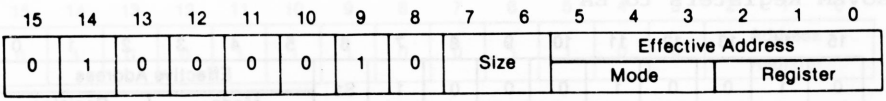


NEG



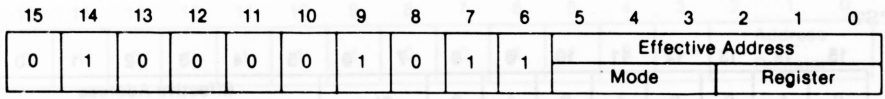
Size field: 00 = byte 01 = word 10 = long

MOVE to CCR

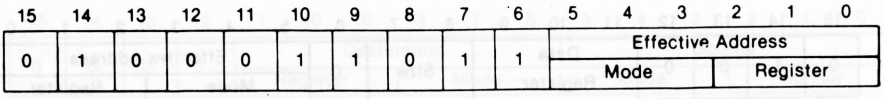


Size field: 00 = byte 01 = word 10 = long

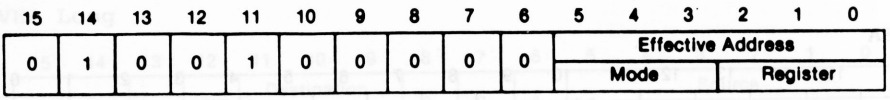
NOT



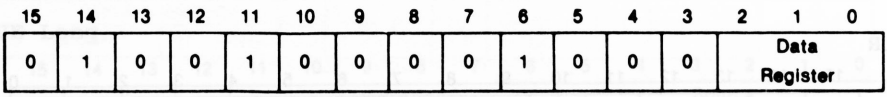
MOVE to SR



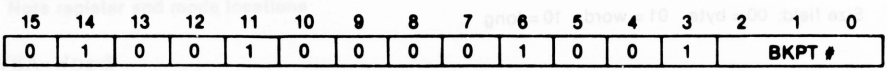
NBCD



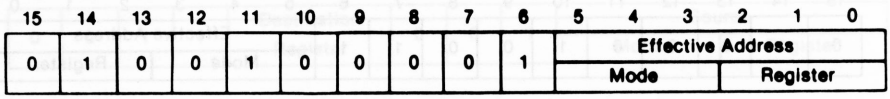
SWAP



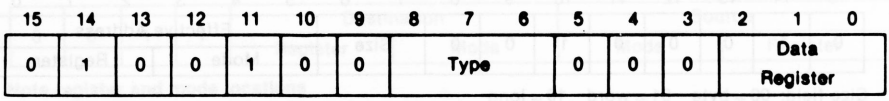
BKPT (MC68010/MC68012)



PEA

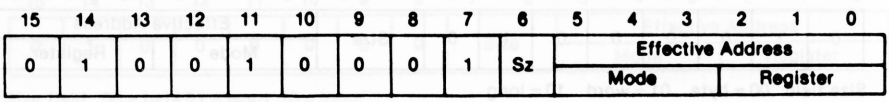


EXT Word



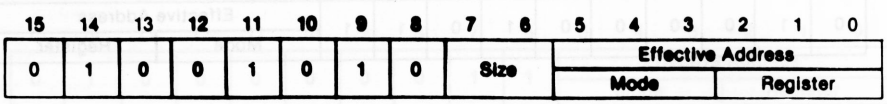
Type Field: 010 = Extend Word 011 = Extend Long

MOVEM Registers to EA



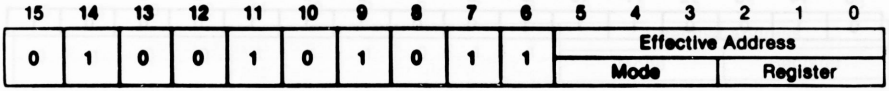
Sz field: 0 = word transfer 1 = long transfer

TST

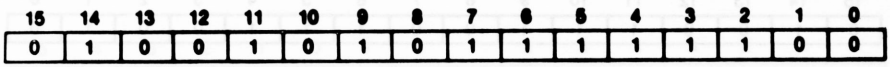


Size field: 00 = byte 01 = word 10 = long

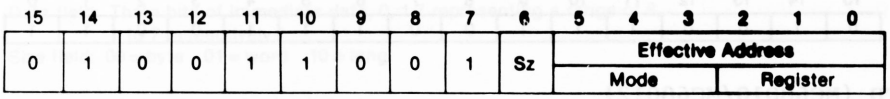
TAS



ILLEGAL

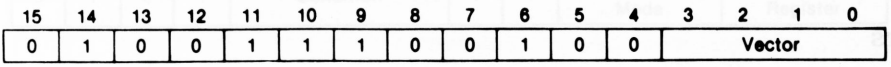


MOVEM EA to Registers

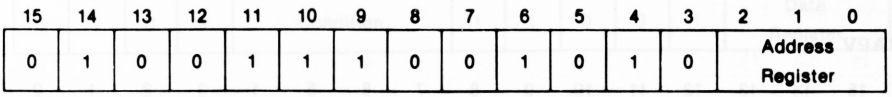


Sz field: 0 = word transfer 1 = long transfer

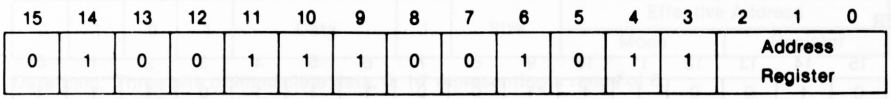
TRAP



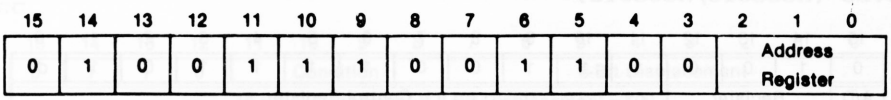
LINK Word



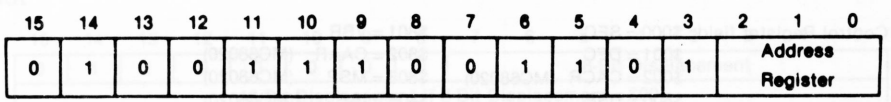
UNLK



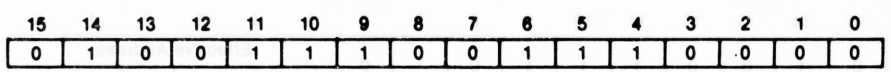
MOVE to USP



MOVE from USP



RESET



NOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

STOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0

RTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTD (MC68010/MC68012)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0

RTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

TRAPV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

RTR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

MOVEC (MC68010/MC68012)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	dr
A/D	Register			Control Register											

dr field: 0 = control register to general register
1 = general register to control register

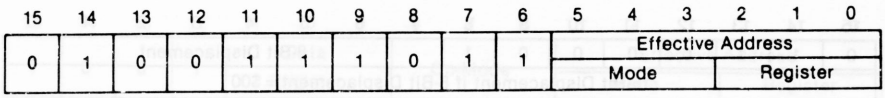
Control Register field:

$000 = SFC		$801 = VBR	
$001 = DFC		$802 = CAAR	(MC68020)
$002 = CACR	(MC68020)	$803 = MSP	(MC68020)
$800 = USP		$804 = ISP	(MC68020)

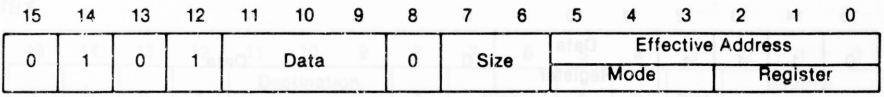
JSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	Effective Address					
										Mode			Register		

JMP

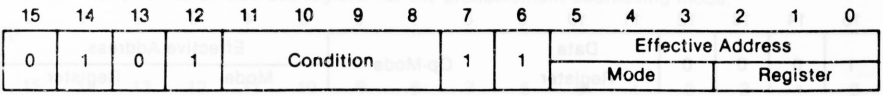


ADDQ

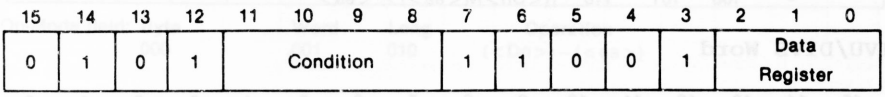


Data field: Three bits of immediate data, 0, 1-7 representing a range of 8, 1 to 7 respectively.
Size field: 00 = byte 01 = word 10 = long

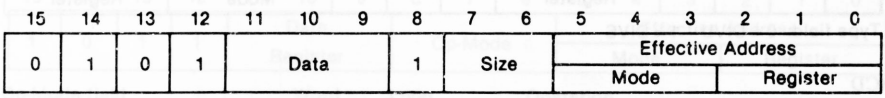
Scc



DBcc

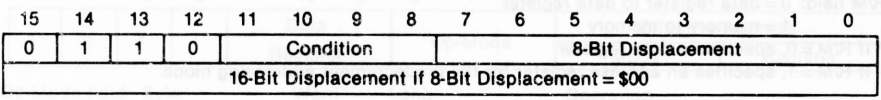


SUBQ

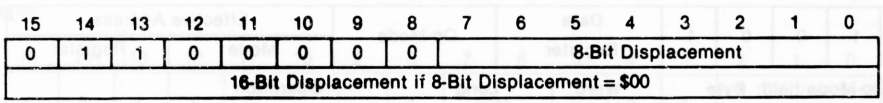


Data field: Three bits of immediate data, 0, 1-7 representing a range of 8, 1 to 7 respectively.
Size field: 00 = byte 01 = word 10 = long

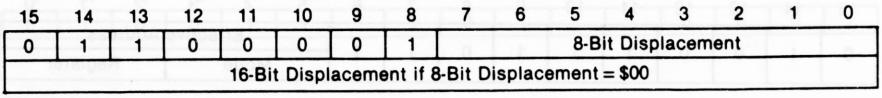
Bcc



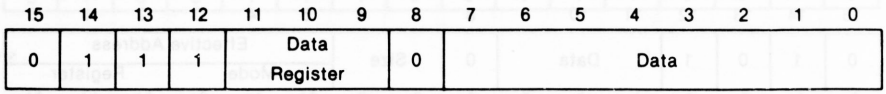
BRA



BSR

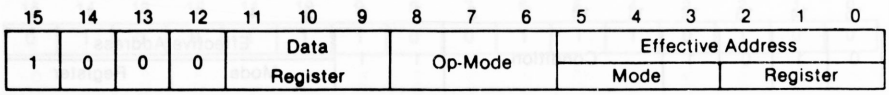


MOVEQ



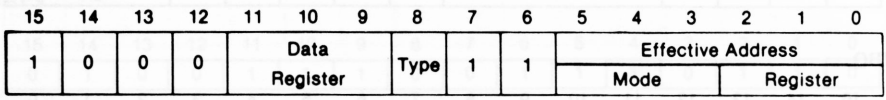
Data field: Data is sign extended to a long operand and all 32 bits are transferred to the data register.

OR



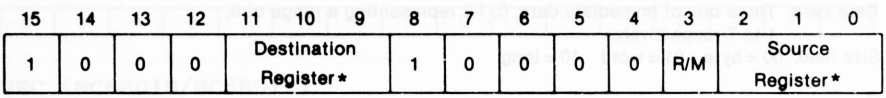
Op-Mode field:	Byte	Word	Long	Operation
	000	001	010	(<ea>)v(<Dn>) → <Dn>
	100	101	110	(<Dn>)v(<ea>) → <ea>

DIVU/DIVS Word



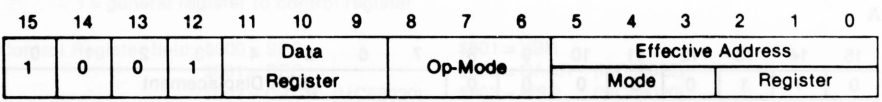
Type field: 0 = DIVU 1 = DIVS

SBCD



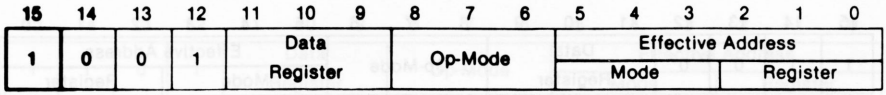
R/M field: 0 = data register to data register
1 = memory to memory
*If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode.

SUB



Op-Mode field:	Byte	Word	Long	Operation
	000	001	010	(<Dn>) − (<ea>) → <Dn>
	100	101	110	(<ea>) − (<Dn>) → <ea>

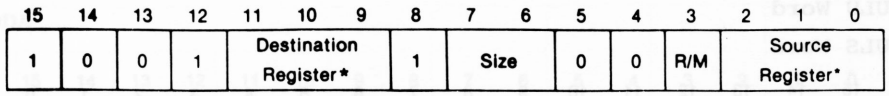
SUBA



Op-Mode field:

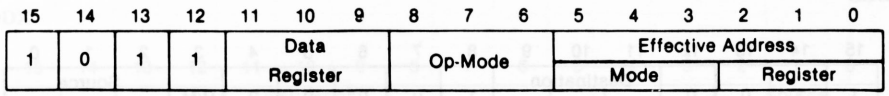
Word	Long	Operation
011	111	(<An>)−(<ea>)→<An>

SUBX



Size field: 00 = byte 01 = word 10 = long
R/M field: 0 = data register to data register 1 = memory to memory
*If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode.

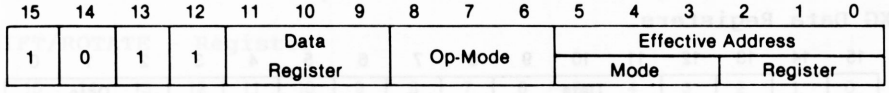
CMP



Op-Mode field:

Byte	Word	Long	Operation
000	001	010	(<Dn>)−(<ea>)

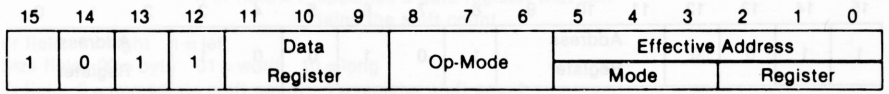
CMPA



Op-Mode field:

Word	Long	Operation
011	111	(<An>)−(<ea>)

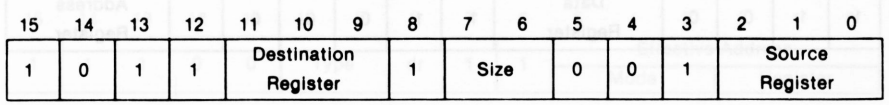
EOR



Op-Mode field:

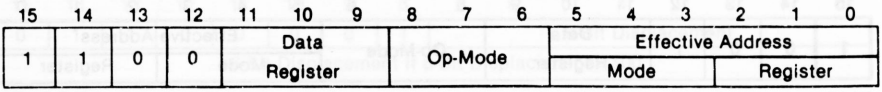
Byte	Word	Long	Operation
100	101	110	(<ea>)⊕(<Dn>)→<ea>

CMPM



Size field: 00 = byte 01 = word 10 = long

AND

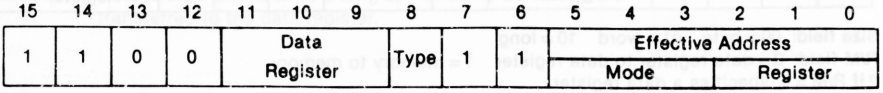


Op-Mode field:

Byte	Word	Long	Operation
000	001	010	(<ea>)Λ(<Dn>) → <Dn>
100	101	110	(<Dn>)Λ(<ea>) → <ea>

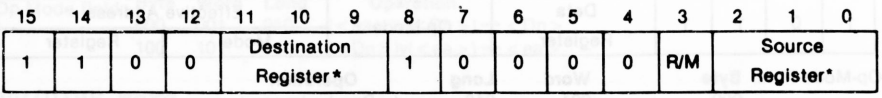
MULU Word

MULS Word



Type field: 0 = MULU 1 = MULS

ABCD

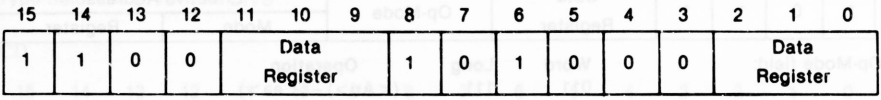


R/M field: 0 = data register to data register 1 = memory to memory

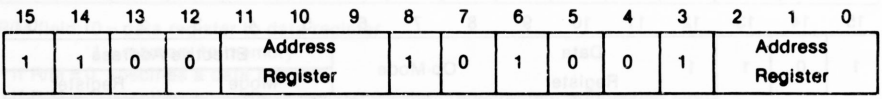
*If R/M = 0, specifies a data register

If R/M = 1, specifies an address register for the predecrement addressing mode.

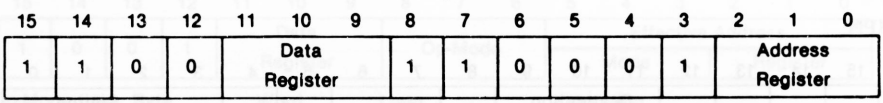
EXG Data Registers



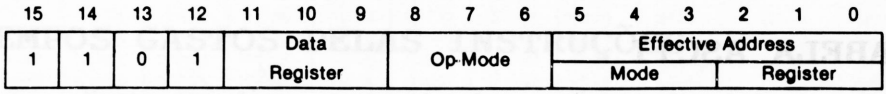
EXG Address Registers



EXG Data Register and Address Register



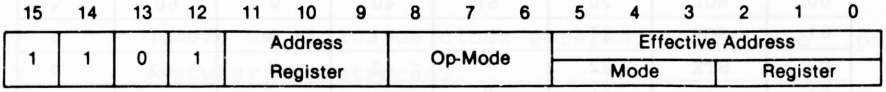
ADD



Op-Mode field:

Byte	Word	Long	Operation
000	001	010	(<ea>)+(<Dn>)→<Dn>
100	101	110	(<Dn>)+(<ea>)→<ea>

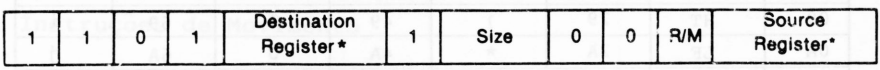
ADDA



Op-Mode field:

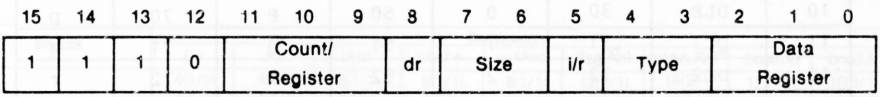
Word	Long	Operation
011	111	(<ea>)+(<An>)→<An>

ADDX



Size field: 00 = byte 01 = word 10 = long
R/M field: 0 = data register to data register 1 = memory to memory
*If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode.

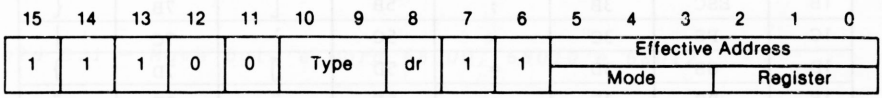
SHIFT/ROTATE - Register



Count/Register field: If i/r field = 0, specifies shift count
If i/r field = 1, specifies a data register that contains the shift count

dr field: 0 = right 1 = left
Size field: 00 = byte 01 = word 10 = long
i/r field: 0 = immediate shift count 1 = register shift count
Type field: 00 = arithmetic shift 10 = rotate with extend
01 = logical shift 11 = rotate

SHIFT/ROTATE - Memory



Type field: 00 = arithmetic shift 01 = logical shift 10 = rotate with extend 11 = rotate
dr field: 0 = right 1 = left

APÊNDICE B

TABELA ASCII

CONJUNTO DOS CARACTERES ASCII

HEXA	ASCII	HEXA	ASCII	HEXA	ASCII	HEXA	ASCII
00	NUL	20	SP	40	@	60	\
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

APÊNDICE C

TEMPOS GASTOS PELAS INSTRUÇÕES

Neste apêndice, vamos abordar o número de ciclos gastos por cada instrução em termos de clock externo.

Nas tabelas, vamos observar a existência de um conjunto de números com a seguinte formação:

a (b/c)

onde: a = número de ciclos de clock externo, necessário para executar a instrução.

b = número de ciclo de clock externo, necessário para executar o ciclo de leitura.

c = número de ciclo de clock externo, necessário para executar o ciclo de escrita.

C.1 Instruções de Movimento

Nas tabelas C.1 e C.2, encontram-se os cilcos gastos para movimentar byte, word e long word. Está incluso o ciclo de busca, operação de leitura e escrita. Esta tabela é válida para os processadores 68000, 68010 e 68012.

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d_{16}(An)	d_8(An,Xn)*	(xxx).W	(xxx).L
Dn	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
(An)	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
(An)+	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
−(An)	10(2/0)	10(2/0)	14(2/1)	14(2/1)	14(2/1)	18(3/1)	20(3/1)	18(3/1)	22(4/1)
d_{16}(An)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d_8(An,Xn)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
(xxx).W	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
(xxx).L	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	24(5/1)	26(5/1)	24(5/1)	28(6/1)
d_{16}(PC)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d_8(PC, Xn)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
#<data>	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)

*The size of the index register (Xn) does not affect execution time.

Tabela C.1 - Move byte e word 68000, 68010 e 68012.

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d_{16}(An)	d_8(An,Xn)*	(xxx).W	(xxx).L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
(An)	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
(An)+	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
−(An)	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
d_{16}(An)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
d_8(An,Xn)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
(xxx).W	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
(xxx).L	20(5/0)	20(5/0)	28(5/2)	28(5/2)	28(5/2)	32(6/2)	34(6/2)	32(6/2)	36(7/2)
d(PC)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(5/2)
d(PC,Xn)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
#<data>	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)

*The size of the index register (Xn) does not affect execution time.

Tabela C.2 - Move long word 68000, 68010 e 68012.

Na tabela C.3 e C.4, temos os ciclos necessários para o 68008.

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d_{16}(An)	d_8(An,Xn)*	(xxx).W	(xxx).L
Dn	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	20(4/1)	22(4/1)	20(4/1)	28(6/1)
An	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	20(4/1)	22(4/1)	20(4/1)	28(6/1)
(An)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	24(5/1)	26(5/1)	24(5/1)	32(7/1)
(An)+	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	24(5/1)	26(5/1)	24(5/1)	32(7/1)
−(An)	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	26(5/1)	28(5/1)	26(5/1)	34(7/1)
d_{16}(An)	20(5/0)	20(5/0)	24(5/1)	24(5/1)	24(5/1)	32(7/1)	34(7/1)	32(7/1)	40(9/1)
d_8(An,Xn)*	22(5/0)	22(5/0)	26(5/1)	26(5/1)	26(5/1)	34(7/1)	36(7/1)	34(7/1)	42(9/1)
(xxx).W	20(5/0)	20(5/0)	24(5/1)	24(5/1)	24(5/1)	32(7/1)	34(7/1)	32(7/1)	40(9/1)
(xxx).L	28(7/0)	28(7/0)	32(7/1)	32(7/1)	32(7/1)	40(9/1)	42(9/1)	40(9/1)	48(11/1)
d_{16}(PC)	20(5/0)	20(5/0)	24(5/1)	24(5/1)	24(5/1)	32(7/1)	34(7/1)	32(7/1)	40(9/1)
d_8(PC,Xn)*	22(5/0)	22(5/0)	26(5/1)	26(5/1)	26(5/1)	34(7/1)	36(7/1)	34(7/1)	42(9/1)
#<data>	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	28(6/1)	30(6/1)	28(6/1)	36(8/1)

*The size of the index register (Xn) does not affect execution time.

Tabela C.3a

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d_{16}(An)	d_8(An,Xn)*	(xxx).W	(xxx).L
Dn	8(2/0)	8(2/0)	16(2/2)	16(2/2)	16(2/2)	24(4/2)	26(4/2)	20(4/2)	32(6/2)
An	8(2/0)	8(2/0)	16(2/2)	16(2/2)	16(2/2)	24(4/2)	26(4/2)	20(4/2)	32(6/2)
(An)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	32(6/2)	34(6/2)	32(6/2)	40(8/2)
(An)+	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	32(6/2)	34(6/2)	32(6/2)	40(8/2)
−(An)	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	34(6/2)	32(6/2)	34(6/2)	42(8/2)
d_{16}(An)	24(6/0)	24(6/0)	32(6/2)	32(6/2)	32(6/2)	40(8/2)	42(8/2)	40(8/2)	48(10/2)
d_8(An,Xn)*	26(6/0)	26(6/0)	34(6/2)	34(6/2)	34(6/2)	42(8/2)	44(8/2)	42(8/2)	50(10/2)
(xxx).W	24(6/0)	24(6/0)	32(6/2)	32(6/2)	32(6/2)	40(8/2)	42(8/2)	40(8/2)	48(10/2)
(xxx).L	32(8/0)	32(8/0)	40(8/2)	40(8/2)	40(8/2)	48(10/2)	50(10/2)	48(10/2)	56(12/2)
d_{16}(PC)	24(6/0)	24(6/0)	32(6/2)	32(6/2)	32(6/2)	40(8/2)	42(8/2)	40(8/2)	48(10/2)
d_8(PC,Xn)*	26(6/0)	26(6/0)	34(6/2)	34(6/2)	34(6/2)	42(8/2)	44(8/2)	42(8/2)	50(10/2)
#<data>	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	32(6/2)	34(6/2)	32(6/2)	40(8/2)

*The size of the index register (Xn) does not affect execution time.

Tabela C.3b

Tabela C.3 - Move byte e word 68008.

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d_{16}(An)	d_8(An,Xn)*	(xxx).W	(xxx).L
Dn	8(2/0)	8(2/0)	24(2/4)	24(2/4)	24(2/4)	32(4/4)	34(4/4)	32(4/4)	40(6/4)
An	8(2/0)	8(2/0)	24(2/4)	24(2/4)	24(2/4)	32(4/4)	34(4/4)	32(4/4)	40(6/4)
(An)	24(6/0)	24(6/0)	40(6/4)	40(6/4)	40(6/4)	48(8/4)	50(8/4)	48(8/4)	56(10/4)
(An)+	24(6/0)	24(6/0)	40(6/4)	40(6/4)	40(6/4)	48(8/4)	50(8/4)	48(8/4)	56(10/4)
−(An)	26(6/0)	26(6/0)	42(6/4)	42(6/4)	42(6/4)	50(8/4)	52(8/4)	50(8/4)	58(10/4)
d_{16}(An)	32(8/0)	32(8/0)	48(8/4)	48(8/4)	48(8/4)	56(10/4)	58(10/4)	56(10/4)	64(12/4)
d_8(An,Xn)*	34(8/0)	34(8/0)	50(8/4)	50(8/4)	50(8/4)	58(10/4)	60(10/4)	58(10/4)	66(12/4)
(xxx).W	32(8/0)	32(8/0)	48(8/4)	48(8/4)	48(8/4)	56(10/4)	58(10/4)	56(10/4)	64(12/4)
(xxx).L	40(10/0)	40(10/0)	56(10/4)	56(10/4)	56(10/4)	64(12/4)	66(12/4)	64(12/4)	72(14/4)
d_{16}(PC)	32(8/0)	32(8/0)	48(8/4)	48(8/4)	48(8/4)	56(10/4)	58(10/4)	56(10/4)	64(12/4)
d_8(PC,Xn)*	34(8/0)	34(8/0)	50(8/4)	50(8/4)	50(8/4)	58(10/4)	60(10/4)	58(10/4)	66(12/4)
#<data>	24(6/0)	24(6/0)	40(6/4)	40(6/4)	40(6/4)	48(8/4)	50(8/4)	48(8/4)	56(10/4)

*The size of the index register (Xn) does not affect execution time.

Tabela C.4 - Move long word 68008.

C.2 Instruções Lógicas e Aritméticas Padrões

Na tabela C.5, estão os ciclos gastos para as instruções lógicas e aritméticas padrões, onde são usadas as seguintes abreviações:

An = registrador de endereço
Dn = registrador de dados
ea = operando especificado por um registrador
M = endereço de memória efetivo

Esta tabela C.5 é válida para o 68000, 68010 e 68012. A tabela C.6 é valida para o 68008.

Instruction	Size	op<ea>, An†	op<ea>, Dn	op Dn, <M>
ADD/ADDA	Byte, Word	8(1/0)+	4(1/0)+	8(1/1)+
	Long	6(1/0)+**	6(1/0)+**	12(1/2)+
AND	Byte, Word	–	4(1/0)+	8(1/1)+
	Long	–	6(1/0)+**	12(1/2)+
CMP/CMPA	Byte, Word	6(1/0)+	4(1/0)+	–
	Long	6(1/0)+	6(1/0)+	–
DIVS	–	–	158(1/0)+*	–
DIVU	–	–	140(1/0)+*	–
EOR	Byte, Word	–	4(1/0)***	8(1/1)+
	Long	–	8(1/0)***	12(1/2)+
MULS	–	–	70(1/0)+*	–
MULU	–	–	70(1/0)+*	–
OR	Byte, Word	–	4(1/0)+	8(1/1)+
	Long	–	6(1/0)+**	12(1/2)+
SUB	Byte, Word	8(1/0)+	4(1/0)+	8(1/1)+
	Long	6(1/0)+**	6(1/0)+**	12(1/2)+

Tabela C.5 - Instruções lógicas e aritméticas para o 68000, 68010 e 68012.

Instruction	Size	op <ea>, An	op <ea>, Dn	op Dn, <M>
ADD/ ADDA	Byte	–	8(2/0) +	12(2/1) +
	Word	12(2/0) +	8(2/0) +	16(2/2) +
	Long	10(2/0) + **	10(2/0) + **	24(2/4) +
AND	Byte	–	8(2/0) +	12(2/1) +
	Word	–	8(2/0) +	16(2/2) +
	Long	–	10(2/0) + **	24(2/4) +
CMP/ CMPA	Byte	–	8(2/0) +	–
	Word	10(2/0) +	8(2/0) +	–
	Long	10(2/0) +	10(2/0) +	–
DIVS		–	162(2/0) + *	–
DIVU		–	144(2/0) + *	–
EOR	Byte	–	8(2/0) + ***	12(2/1) +
	Word	–	8(2/0) + ***	16(2/2) +
	Long	–	12(2/0) + ***	24(2/4) +
MULS		–	74(2/0) + *	–
MULU		–	74(2/0) + *	–
OR	Byte	–	8(2/0) +	12(2/1) +
	Word	–	8(2/0) +	16(2/2) +
	Long	–	10(2/0) + **	24(2/4) +
SUB	Byte	–	8(2/0) +	12(2/1) +
	Word	12(2/0) +	8(2/0) +	16(2/2) +
	Long	10(2/0) + **	10(2/0) + **	24(2/4) +

Tabela C.6 - Instruções lógicas e aritméticas para o 68008.

C.3 Instruções de Operando Imediato

Na tabela C.7, estão os ciclos gastos pelas instruções de operação imediata para os processadores 68000, 68010 e 68012.

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/2) +
ADDQ	Byte, Word	4(1/0)	8(1/0)*	8(1/1) +
	Long	8(1/0)	8(1/0)	12(1/2) +
ANDI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/1) +
CMPI	Byte, Word	8(2/0)	–	8(2/0) +
	Long	14(3/0)	–	12(3/0) +
EORI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/2) +
MOVEQ	Long	4(1/0)	–	–
ORI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/2) +
SUBI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/2) +
SUBQ	Byte, Word	4(1/0)	8(1/0)*	8(1/1) +
	Long	8(1/0)	8(1/0)	12(1/2) +

+ add effective address calculation time
* word only

Tabela C.7 - Instruções de operação imediata para o 68000, 68010 e 68012.

Na tabela C.8, os cilcos gastos para execução das instruções de operação imediata para o processador 68008.

Instruction	Size	op#, Dn	op#,An	op#, M
ADDI	Byte	16(4/0)	–	20(4/1)+
	Word	16(4/0)	–	24(4/2)+
	Long	28(6/0)	–	40(6/4)+
ADDQ	Byte	8(2/0)	–	12(2/1)+
	Word	8(2/0)	12(2/0)	16(2/2)+
	Long	12(2/0)	12(2/0)	24(2/4)+
ANDI	Byte	16(4/0)	–	20(4/1)+
	Word	16(4/0)	–	24(4/2)+
	Long	28(6/0)	–	40(6/4)+
CMPI	Byte	16(4/0)	–	16(4/0)+
	Word	16(4/0)	–	16(4/0)+
	Long	26(6/0)	–	24(6/0)+
EORI	Byte	16(4/0)	–	20(4/1)+
	Word	16(4/0)	–	24(4/2)+
	Long	28(6/0)	–	40(6/4)+
MOVEQ	Long	8(2/0)	–	–
ORI	Byte	16(4/0)	–	20(4/1)+
	Word	16(4/0)	–	24(4/2)+
	Long	28(6/0)	–	40(6/4)+
SUBI	Byte	16(4/0)	–	12(2/1)+
	Word	16(4/0)	–	16(2/2)+
	Long	28(6/0)	–	24(2/4)+
SUBQ	Byte	8(2/0)	–	20(4/1)+
	Word	8(2/0)	12(2/0)	24(4/2)+
	Long	12(2/0)	12(2/0)	40(6/4)+

+ add effective address calculation time

Tabela C.8 - Instruções de operação imediato para o 68008.

C.4 Instruções de Operando Simples

Na tabela C.9, estão os ciclos gastos para execução das instruções de operando simples para os processadores 68000, 68010 68012 e a tabela C.10 para o processador 68008.

Instruction	Size	Register	Memory
CLR	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NBCD	Byte	6(1/0)	8(1/1)+
NEG	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NEGX	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NOT	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
Scc	Byte, False	4(1/0)	8(1/1)+
	Byte, True	6(1/0)	8(1/1)+
TAS	Byte	4(1/0)	10(1/1)+
TST	Byte, Word	4(1/0)	4(1/0)+
	Long	4(1/0)	4(1/0)+

Tabela C.9 - Instruções de operando simples para os processadores 68000, 68010 e 68012.

Instruction	Size	Register	Memory
CLR	Byte	8(2/0)	12(2/1) +
	Word	8(2/0)	16(2/2) +
	Long	10(2/0)	24(2/4) +
NBCD	Byte	10(2/0)	12(2/1) +
NEG	Byte	8(2/0)	12(2/1) +
	Word	8(2/0)	16(2/2) +
	Long	10(2/0)	24(2/4) +
NEGX	Byte	8(2/0)	12(2/1) +
	Word	8(2/0)	16(2/2) +
	Long	10(2/0)	24(2/4) +
NOT	Byte	8(2/0)	12(2/1) +
	Word	8(2/0)	16(2/2) +
	Long	10(2/0)	24(2/4) +
Scc	Byte, False	8(2/0)	12(2/1) +
	Byte, True	10(2/0)	12(2/1) +
TAS	Byte	8(2/0)	14(2/1) +
TST	Byte	8(2/0)	8(2/0) +
	Word	8(2/0)	8(2/0) +
	Long	8(2/0)	8(2/0) +

+ add effective address calculation time.

Tabela C.10 - Instruções de operando simples para o 68008.

C.5 Instruções de Rotação e Deslocamento

Na tabela C.11, estão os ciclos de clocks necessários para execução das instruções de rotação e deslocamento, para os processadores 68000, 68010 e 68012 e na tabela C.12, no processador 68008.

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
LSR, LSL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
ROR, ROL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
ROXR, ROXL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–

+ add effective address calculation time for word operands
n is the shift count

Tabela C.11 - Instruções de deslocamento e rotação nos processadores 68000, 68010 e 68012.

Instruction	Size	Register	Memory
ASR, ASL	Byte	10 + 2n(2/0)	–
	Word	10 + 2n(2/0)	16(2/2) +
	Long	12 + 2n(2/0)	–
LSR, LSL	Byte	10 + 2n(2/0)	–
	Word	10 + 2n(2/0)	16(2/2) +
	Long	12 + 2n(2/0)	–
ROR, ROL	Byte	10 + 2n(2/0)	–
	Word	10 + 2n(2/0)	16(2/2) +
	Long	12 + 2n(2/0)	–
ROXR, ROXL	Byte	10 + 2n(2/0)	–
	Word	10 + 2n(2/0)	16(2/2) +
	Long	12 + 2n(2/0)	–

+ add effective address calculation time for word operands
n is the shift count

Tabela C.12 - Instruções de deslocamento e rotação no processador 68008.

C.6 Instruções de Manipulação de Bits

Na tabela C.13, estão os ciclos de clock necessários para executar as instruções de manipulação de bits, para os processadores 68000, 68010 e 68012. Na tabela C.14, estão as instruções de manipulação de bits para o processador 68008.

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	–	8(1/1) +	–	12(2/1) +
	Long	8(1/0) *	–	12(2/0) *	–
BCLR	Byte	–	8(1/1) +	–	12(2/1) +
	Long	10(1/0) *	–	14(2/0) *	–
BSET	Byte	–	8(1/1) +	–	12(2/1) +
	Long	8(1/0) *	–	12(2/0) *	–
BTST	Byte	–	4(1/0) +	–	8(2/0) +
	Long	6(1/0)	–	10(2/0)	–

+ add effective address calculation time
* indicates maximum value; data addressing mode only

Tabela C.13 - Instruções de manipulação de bits para o 68000, 68010 e 68012.

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	–	12(2/1)+	–	20(4/1)+
	Long	12(2/0)*	–	20(4/0)*	–
BCLR	Byte	–	12(2/1)+	–	20(4/1)+
	Long	14(2/0)*	–	22(4/0)*	–
BSET	Byte	–	12(2/1)+	–	20(4/1)+
	Long	12(2/0)*	–	20(4/0)*	–
BTST	Byte	–	8(2/0)+	–	16(4/0)+
	Long	10(2/0)	–	18(4/0)	–

+ add effective address calculation time
* Indicates maximum value; data addressing mode only

Tabela C.14 - Instruções de Manipulação de bits para o 68008.

C.7 Instruções de Operação Condicional

Na tabela C.15, estão os números de ciclos de clock necessários para executar as instruções de operação condicional. A tabela C.15 é válida para os processadores 68000, 68010 e 68012, enquanto a tabela C.16 é válida para o processador 68008.

Instruction	Displacement	Branch Taken	Branch Not Taken
Bcc	Byte	10(2/0)	8(1/0)
	Word	10(2/0)*	12(2/0)
BRA	Byte	10(2/0)	–
	Word	10(2/0)	–
BSR	Byte	18(2/2)	–
	Word	18(2/2)	–
DBcc	cc true	–	12(2/0)
	cc false, Count Not Expired	10(2/0)	–
	cc false, Counter Expired	–	14(3/0)

+ add effective address calculation time
* indicates maximum base value

Tabela C.15 - Instruções de operação condicional nos processadores 68000, 68010 e 68012.

Instruction	Displacement	Trap or Branch Taken	Trap or Branch Not Taken
Bcc	Byte	18(4/0)	12(2/0)
	Word	18(4/0)	20(4/0)
BRA	Byte	18(4/0)	–
	Word	18(4/0)	–
BSR	Byte	34(4/4)	–
	Word	34(4/4)	–
DBcc	CC True	–	20(4/0)
	CC False	18(4/0)	26(6/0)
CHK	–	68(8/6)+*	14(2/0)+
TRAP	–	62(8/6)	–
TRAPV	–	66(10/6)	8(2/0)

+ add effective address calculation time for word operand
* indicates maximum base value

Tabela C.16 - Instruções de operação condicional no processador 68008.

C.8 Instruções de Salto

Na tabela C.17, estão os ciclos de clock necessários para a execução das instruções de "jump", jump-to-subrotine e movimento de múltiplos registradores. A tabela C.17 é válida para os processadores 68000, 68010 e 68012. A tabela C.18 é válida para o processador 68008.

Instruction	Size	(An)	(An)+	−(An)	d_{16}(An)	d_8(An,Xn)+	(xxx).W	(xxx).L	d_8(PC)	d_{16}(PC,Xn)*
JMP	–	8(2/0)	–	–	10(2/0)	14(3/0)	10(2/0)	12(3/0)	10(2/0)	14(3/0)
JSR	–	16(2/2)	–	–	18(2/2)	22(2/2)	18(2/2)	20(3/2)	18(2/2)	22(2/2)
LEA	–	4(1/0)	–	–	8(2/0)	12(2/0)	8(2/0)	12(3/0)	8(2/0)	12(2/0)
PEA	–	12(1/2)	–	–	16(2/2)	20(2/2)	16(2/2)	20(3/2)	16(2/2)	20(2/2)
MOVEM M→R	Word	12+4n (3+n/0)	12+4n (3+n/0)	–	16+4n (4+n/0)	18+4n (4+n/0)	16+4n (4+n/0)	20+4n (5+n/0)	16+4n (4+n/0)	18+4n (4+n/0)
	Long	12+8n (3+2n/0)	12+8n (3+2n/0)	–	16+8n (4+2n/0)	18+8n (4+2n/0)	16+8n (4+2n/0)	20+8n (5+2n/0)	16+8n (4+2n/0)	18+8n (4+2n/0)
MOVEM R→M	Word	8+4n (2/n)	–	8+4n (2/n)	12+4n (3/n)	14+4n (3/n)	12+4n (3/n)	16+4n (4/n)	– –	– –
	Long	8+8n (2/2n)	– –	8+8n (2/2n)	12+8n (3/2n)	14+8n (3/2n)	12+8n (3/2n)	16+8n (4/2n)	– –	– –
MOVES M→R	Byte/ Word	18(3/0)	20(3/0)	20(3/0)	20(4/0)	24(4/0)	20(4/0)	24(5/0)		
	Long	22(4/0)	24(4/0)	24(4/0)	24(5/0)	28(5/0)	24(5/0)	28(6/0)		
MOVES R→M	Byte/ Word	18(2/1)	20(2/1)	20(271)	20(3/1)	24(3/1)	20(3/1)	24(4/1)		
	Long	22(2/2)	24(2/2)	24(2/2)	24(3/2)	28(3/2)	24(3/2)	28(4/2)		

n is the number of registers to move
* is the size of the index register (ix) does not affect the instruction's execution time

Tabela C.17 - Instruções de salto para os processadores 68000, 68010 e 68012.

Instruction	Size	(An)	(An)+	−(An)	d_{16}(An)	d_8(An,Xn)*	(xxx).W	(xxx).L	d_{16}(PC)	d_8(PC,Xn)
JMP	–	16(4/0)	–	–	18(4/0)	22(4/0)	18(4/0)	24(6/0)	18(4/0)	22(4/0)
JSR	–	32(4/4)	–	–	34(4/4)	38(4/4)	34(4/4)	40(6/4)	34(4/4)	32(4/4)
LEA	–	8(2/0)	–	–	16(4/0)	20(4/0)	16(4/0)	24(6/0)	16(4/0)	20(4/0)
PEA	–	24(2/4)	–	–	32(4/4)	36(4/4)	32(4/4)	40(6/4)	32(4/4)	36(4/4)
MOVEM M→R	Word	24+8n (6+2n/0)	24+8n (6+2n/0)	– –	32+8n (8+2n/0)	34+8n (8+2n/0)	32+8n (10+n/0)	40+8n (10+2n/0)	32+8n (8+2n/0)	34+8n (8+2n/0)
	Long	24+16n (6+4n/0)	24+16n (6+4n/0)	– –	32+16n (8+4n/0)	34+16n (8+4n/0)	32+16n (8+4n/0)	40+16n (8+4n/0)	32+6n (8+4n/0)	34+16n (8+4n/0)
MOVEM R→M	Word	16+8n (4/2n)	– –	16+8n (4/2n)	24+8n (6/2n)	26+8n (6/2n)	24+8n (6/2n)	32+8n (8/2n)	–	–
	Long	16+16n (4/4n)	– –	16+16n (4/4n)	24+16n (6/4n)	26+16n	24+16n (8/4n)	32+16n (6/4n)	–	–

n is the number of registers to move
* is the size of the index register (Xn) does not affect the instruction's execution time

Tabela C.18 - Instruções de salto para o processador 68008.

C.9 Instruções de Multiprecisão Aritmética

Na tabela C.19, estão os ciclos de clock gastos para a execução das instruções de multiprecisão aritmética.

A tabela C.19 é válida para o processador 68000, enquanto a tabela C.20 é válida para o processador 68008 e a tabela C.21 para os processadores 68010 e 68012.

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
CMPM	Byte, Word	–	12(3/0)
	Long	–	20(5/0)
SUBX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
ABCD	Byte	6(1/0)	18(3/1)
SBCD	Byte	6(1/0)	18(3/1)

Tabela C.19 - Instruções de multiprecisão aritmética para o processador 68000.

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte	8(2/0)	22(4/1)
	Word	8(2/0)	50(6/2)
	Long	12(2/0)	58(10/4)
CMPM	Byte	–	16(4/0)
	Word	–	24(6/0)
	Long	–	40(10/0)
SUBX	Byte	8(2/0)	22(4/1)
	Word	8(2/0)	50(6/2)
	Long	12(2/0)	58(10/4)
ABCD	Byte	10(2/0)	20(4/1)
SBCD	Byte	10(2/0)	20(4/1)

Tabela C.20 - Instruções de multiprecisão aritmética para o processador 68008.

				Loop Mode		
				Continued	Terminated	
		Non-Looped		Valid Count, cc False	Valid Count, cc True	Expired Count
Instruction	Size	op Dn, Dn	op M, M*			
ADDX	Byte, Word	4(1/0)	18(3/10)	22(2/1)	28(4/1)	26(4/1)
	Long	6(1/0)	30(5/2)	32(4/2)	38(6/2)	36(6/2)
CMPM	Byte, Word	–	12(3/0)	14(2/0)	20(4/0)	18(4/0)
	Long	–	20(5/0)	24(4/0)	30(6/0)	26(6/0)
SUBX	Byte, Word	4(1/0)	18(3/1)	22(2/1)	28(4/1)	26(4/1)
	Long	6(1/0)	30(5/2)	32(4/2)	38(6/2)	36(6/2)
ABCD	Byte	6(1/0)	18(3/1)	24(2/1)	30(4/1)	28(4/1)
SBCD	Byte	6(1/0)	18(3/1)	24(2/1)	30(4/1)	28(4/1)

* Source and destination ea is (An)+ for CMPM and –(An) for all others.

Tabela C.21 - Instruções de multiprecisão para o 68010 e 68012.

C.10 Instruções de Controle Geral

Na tabela C.22, estão os cilcos gastos para execução de instruções de controle geral, aquelas não enquadradas nas tabelas anteriores.

A tabela C.22 é válida para o processador 68000, a tabela C.23 para o processador 68008 e a tabela C.24 para os processadores 68010 e 68012.

Instruction	Size	Register	Memory
ANDI to CCR	Byte	20(3/0)	–
ANDI to SR	Word	20(3/0)	–
CHK (No Trap)	–	10(1/0) +	–
EORI to CCR	Byte	20(3/0)	–
EORI to SR	Word	20(3/0)	–
ORI to CCR	Byte	20(3/0)	–
ORI to SR	Word	20(3/0)	–
MOVE from SR	–	6(1/0)	8(1/1) +
MOVE to CCR	–	12(1/0)	12(1/0) +
MOVE to SR	–	12(1/0)	12(1/0) +
EXG	–	6(1/0)	–
EXT	Word	4(1/0)	–
	Long	4(1/0)	–
LINK	–	16(2/2)	–
MOVE from USP	–	4(1/0)	–
MOVE to USP	–	4(1/0)	–
NOP	–	4(1/0)	–
RESET	–	132(1/0)	–
RTE	–	20(5/0)	–
RTR	–	20(5/0)	–
RTS	–	16(4/0)	–
STOP	–	4(0/0)	–
SWAP	–	4(1/0)	–
TRAPV	–	4(1/0)	–
UNLK	–	12(3/0)	–

+ add effective address calculation time

Tabela C.22 - Instruções de controle geral do 68000.

Instruction	Register	Memory
ANDI to CCR	32(6/0)	–
ANDI to SR	32(6/0)	–
EORI to CCR	32(6/0)	–
EORI to SR	32(6/0)	–
EXG	10(2/0)	–
EXT	8(2/0)	–
LINK	32(4/4)	–
MOVE to CCR	18(4/0)	18(4/0) +
MOVE to SR	18(4/0)	18(4/0) +
MOVE from SR	10(2/0)	16(2/2) +
MOVE to USP	8(2/0)	–
MOVE from USP	8(2/0)	–
NOP	8(2/0)	–
ORI to CCR	32(6/0)	–
ORI to SR	32(6/0)	–
RESET	136(2/0)	–
RTE	40(10/0)	–
RTR	40(10/0)	–
RTS	32(8/0)	–
STOP	4(0/0)	–
SWAP	8(2/0)	–
TRAPV (No Trap)	8(2/0)	–
UNLK	24(6/0)	–

+ add effective address calculation time for word operand

Tabela C.23 - Instruções de controle geral do 68008.

Instruction	Size	Register	Memory	Register → Destination**	Source** → Register
ANDI to CCR	–	16(2/0)	–	–	–
ANDI to SR	–	16(2/0)	–	–	–
CHK	–	8(1/0) +	–	–	–
EORI to CCR	–	16(2/0)	–	–	–
EORI to SR	–	16(2/0)	–	–	–
EXG	–	6(1/0)	–	–	–
EXT	Word	4(1/0)	–	–	–
	Long	4(1/0)	–	–	–
LINK	–	16(2/2)	–	–	–
MOVE from CCR	–	4(1/0)	8(1/1) + *	–	–
MOVE to CCR	–	12(2/0)	12(2/0) +	–	–
MOVE from SR	–	4(1/0)	8(1/1) + *	–	–
MOVE to SR	–	12(2/0)	12(2/0) +	–	–
MOVE from USP	–	6(1/0)	–	–	–
MOVE to USP	–	6(1/0)	–	–	–
MOVEC	–	–	–	10(2/0)	12(2/0)
MOVEP	Word	–	–	16(2/2)	16(4/0)
	Long	–	–	24(2/4)	24(6/0)
NOP	–	4(1/0)	–	–	–
ORI to CCR	–	16(2/0)	–	–	–
ORI to SR	–	16(2/0)	–	–	–
RESET	–	130(1/0)	–	–	–
RTD	–	16(4/0)	–	–	–
RTE	Short	24(6/0)	–	–	–
	Long, Retry Read	112(27/10)	–	–	–
	Long, Retry Write	112(26/1)	–	–	–
	Long, No Retry	110(26/0)	–	–	–
RTR	–	20(5/0)	–	–	–
RTS	–	16(4/0)	–	–	–
STOP	–	4(0/0)	–	–	–
SWAP	–	4(1/0)	–	–	–
TRAPV	–	4(1/0)	–	–	–
UNLK	–	12(3/0)	–	–	–

+ add effective address calculation time.
* use non-fetching effective address calculation time.
** Source or destination is a memory location for the MOVEP instruction and a control register for the MOVEC instruction

Tabela C.24 - Instruções de controle geral do 68010 e 68012.

C.11 Instrução de Movimento para Periférico

Na tabela C.25, estão os ciclos gastos pela instrução "Movep" para o 68000, 68010 e 68012 e na tabela C.26, estão os ciclos gastos por esta mesma instrução no 68008.

Instruction	Size	Register → Memory	Memory → Register
MOVEP	Word	16(2/2)	16(4/0)
	Long	24(2/4)	24(6/0)

Tabela C.25 - Instrução Movep no 68000, 68010 e 68012.

Instruction	Size	Register → Memory	Memory → Register
MOVEP	Word	24(4/2)	24(6/0)
	Long	32(4/4)	32(8/0)

+ add effective address calculation time

Tabela C.26 - Instrução Movep no 68008.

C.12 Instrução Clear do 68010 e 68012

No caso especial do 68010 e 68012, a instrução CLR (clear) tem um gasto de ciclos de clock diferente dos outros processadores. A tabela C.27 mostra estes ciclos.

	Size	Dn	An	(An)	(An)+	−(An)	d_{16}(An)	d_8(An,Xn)*	(xxx).W	(xxx).L
CLR	Byte, Word	4(1/0)	—	8(1/1)	8(1/1)	10(1/1)	12(2/1)	16(2/1)	12(2/1)	16(3/1)
	Long	6(1/0)	—	12(1/2)	12(1/2)	14(1/2)	16(2/2)	20(2/2)	16(2/2)	20(3/2)

* The size of the index register (Xn) does not affect execution time.

Tabela C.27 - Instrução "Clear" no 68010 e 68012.

C.13 Tempo de Execução dos Processamentos de Exceção

Na tabela C.28, estão os ciclos de clock gastos nos processamentos de exceção para o 68000. Nos números de período de clock, estão inclusos o stack, busca do vetor e etc. Na tabela C.29, estão os ciclos para o 68008 e na tabela C.30, para os processadores 68010 e 68012.

Exception	Periods
Address Error	50(4/7)
Bus Error	50(4/7)
CHK Instruction	40(4/3) +
Divide by Zero	38(4/3) +
Illegal Instruction	34(4/3)
Interrupt	44(5/3)*
Privilege Violation	34(4/3)
RESET**	40(6/0)
Trace	34(4/3)
TRAP Instruction	34(4/3)
TRAPV Instruction	34(5/3)

+ add effective address calculation time

* The interrupt acknowledge cycle is assumed to take four clock periods.

** Indicates the time from when RESET and HALT are first sampled as negated to when instruction execution starts.

Tabela C.29 - Processamento de exceção 68008.

Exception	Periods
Address Error	94(8/14)
Bus Error	94(8/14)
CHK Instruction	68(8/6) +
Divide by Zero	66(8/6) +
Interrupt	72(9/6)*
Illegal Instruction	62(8/6)
Privileged Instruction	62(8/6)
RESET**	64(12/0)
Trace	62(8/6)
TRAP Instruction	62(8/6)
TRAPV Instruction	66(10/6)

+ add effective address calculation time

** Indicates the time from when RESET and HALT are first sampled as negated to when instruction execution starts.

Tabela C.28 - Processamento de exceção 68000.

Exception	
Address Error	126(4/26)
Breakpoint Instruction*	42(5/4)
Bus Error	126(4/26)
CHK Instruction**	44(5/4) +
Divide By Zero	42(5/4) +
Illegal Instruction	38(5/4)
Interrupt*	46(5/4)
MOVEC, Illegal Control Register**	46(5/4)
Privilege Violation	38(5/4)
Reset***	40(6/0)
RTE, Illegal Format	50(7/4)
RTE, Illegal Revision	70(12/4)
Trace	38(4/4)
TRAP Instruction	38(4/4)
TRAPV Instruction	38(5/4)

+ add effective address calculation time.

* The interrupt acknowledge and breakpoint cycles are assumed to take four clock periods.

** Indicates maximum value.

*** Indicates the time from when RESET and HALT are first sampled as negated to when instruction execution starts.

Tabela C.30 - Processamento de exceção do 68010 e 68012.

BIBLIOGRAFIA

- *Introdução aos Microcomputadores,* Joaquim A. Mouras Relvas Editora Figueirinhas - 1980 - Portugal.

- *Microprocessador 68000 Hardware vol. 1,* Wagner Ideali, Editora Érica Ltda, 1987, São Paulo.

- *M68000 8/16/32 bit Microprocessors Programmers Reference Manual,* Morotola INC. 1986, Editora Gordon Osbourne, 15ª edição.

- *Programação Assembler para Microprocessadores 68000, 68010 e* 68020, Tamio Shimizu, Editora McGraw Hill, 1987, 1ª edição, São Paulo.

- *68000 Microprocessor,* Willian Gramer and Gerry Kane, Editora Osborne/McGraw-Hill.

Impressão e acabamento
(com filmes fornecidos):
EDITORA SANTUÁRIO
Fone (0125) 36-2140
APARECIDA - SP

PUBLICAÇÕES ÉRICA

LINGUAGENS E APLICATIVOS

Ivan	- Wordstar auto explicativo
Santos	- Aplicativos
Alexandre	- AutoCad Guia Prático
Arsonval	- Basic para Computadores Pessoais
Ideali	- Sistema Operacional CP/M - 80
Natale	- Probasic - Programação em Basic
Natale	- Basic Avançado
Marcelino	- dBase III - Vol. 1

MICROPROCESSADORES

Ideali	- Microprocessador 68000 - Hardware - Vol. 1
Visconti	- Microprocessadores 8080 e 8085 - Hardware - Vol. 1
Visconti	- Microprocessadores 8080 e 8085 - Software - Vol. 2
Cypriano/Cardinali	- Microprocessadores Z-80 - Hardware - Vol. 1
Cypriano	- Microprocessadores Z-80 - Software - Vol. 2
Vidal	- Microcontroladores

ELETRÔNICA

Cuocolo	- Periféricos
Lando/Serg	- Amplificador Operacional
Capuano/Idoeta	- Elementos de Eletrônica Digital
Almeida	- Eletrônica Industrial
Almeida	- Eletrônica de Potência
Cipelli/Sandrini	- Teoria e Desenvolvimento de Projetos de Circuitos Eletrônicos
Azevedo	- TTL/CMOS - Circuitos Integrados - Vol. 1
Azevedo	- TTL/CMOS - Circuitos Integrados - Vol. 2
Mello	- Projetos de Fontes Chaveadas
Albuquerque	**- Análise de Circuitos em Corrente Contínua**
Capuano/Marino	- Laboratório de Eletricidade e Eletrônica
Aghazarm/Miranda	- Transmissão de dados em Sistemas de Computação

TELECOMUNICAÇÕES

Smit	- Ondas e Antenas
Smit	- Linhas de Comunicação
Smit	- Rádio Propagação
Gomes	- Telecomunicações - Transmissão e Recepção AM FM - Sistemas Pulsados
Smit	- Microondas

OUTROS

Sandrini	- Profissional Ser ou Não Ser
Lence	- Física - Vol. 1
Filkauskas/Guedes	- O Plástico