MOTOROLA

microsystems

EXORset 30
XDOS Assembler
User's Guide

M6809ASM09 (D)

Eks nr 2

EXORset 30
XDOS Assembler
User's Guide

Printed in Switzerland

EUROPEAN MOTOROLA SEMICONDUCTOR SALES OFFICES

DENMARK
Motorola A/S
Gladsaxevej 370
2860 Soborg
Tel. (01) 67 44 22

FRANCE
Motorola Semiconducteurs S.A.
Headquarter
16-17, avenue de Ségur
75007 Paris
Tel. 551 50 61

Sales Office
42, avenue de La Plaine Fleurie
38240 Meylan (Grenoble)
Tel. (76) 90 22 81

WEST GERMANY
Motorola GmbH,
Geschäftsbereich Halbleiter
Headquarter
Munchner Strasse 18
8043 Unterföhring
Tel. (089) 92 481

Sales Offices
Hans Bockler Strasse 30
3012 Langenhagen — Hannover
Tel. (0511) 78 20 37/38

Vinnebergerstrasse 4,1
8500 Nürnberg
Tel. (0911) 6 57 61

Stuttgarter Strasse 1
7032 Sindelfingen
Tel. (07031) 18 30 74/75

Abraham-Lincoln-Strasse 28
6200 Wiesbaden
Tel. (06121) 76 19 21

HOLLAND
Motorola B.V.
Emmalaan 41
Utrecht
Tel. (030) 51 02 07

ITALY
Motorola S.p.A
Headquarter
Via Ciro Menotti 11
20129 Milano
Tel. 738 61 41/2/3

Motorola S.p.A.
Divisione Semiconduttori
Via del Barroccio 2
40138 Bologna
Tel. (051) 53 34 46

Sales Office
Via Portanova 10
40123 Bologna — Tel. 26 69 05

Sales Office
Via Costantino Maes 68
00162 Roma — Tel. 831 47 46

NORWAY
Motorola A/B (Service Office)
Brugt. 1
Oslo 1 — Tel. (02) 41 91 40

SOUTH AFRICA
Motorola South Africa (Pty) Ltd.
P.O. Box 39586
Bramley 2018
Tel. 786 11 84

SPAIN
Motorola Espana S.A.
Capitan Haya 55
Madrid 20 — Tel. 279 08 62

SWEDEN
Motorola AB.
Vretbergsvaegen 19
17140 Solna — Tel. (08) 82 02 95

SWITZERLAND
Motorola Semiconductor Products S.A.
Alte Landstrasse 101
8702 Zollikon — Tel. (01) 65 56 56

UNITED KINGDOM
Motorola Ltd.
Headquarter
York House, Empire Way
Wembley Middlesex
Tel. (01) 902 88 36

Sales Office
16-12, Mount Street,
Television House
Manchester M2 5WS, Lancs
Tel. (61) 833 07 31/833 07 34

Sales Office
Colvilles Road, Kelvin Estate
East Kilbride, Scotland
Tel. (3552) 3 91 01

HEADQUARTERS, EUROPEAN OPERATIONS SWITZERLAND
Motorola Inc.
Semiconductor Group
16, chemin de la Voie-Creuse
P.O. Box 8 — 1211 Genève 20
Tel. (022) 99 11 11

FRANCHISED MOTOROLA SEMICONDUCTOR DISTRIBUTORS

AUSTRIA
Elbatex GmbH
Endresstrasse 54 — 1238 Wien
Tel. (222) 88 56 11

BELGIUM
Diode Belgium
Rue Picard 202-204 — 1020 Bruxelles
Tel. (02) 428 51 05

DENMARK
Distributoren Interelko Aps
Hovedgaden 16 — 4622 Havdrup
Tel. (03) 38 57 16

FINLAND
Field Oy
Vevoentekijantie 18 — 00210 Helsinki 21
Tel. (90) 692 25 77

FRANCE
Bellion Electronique
Zone Industrielle de Kerscao/Brest
29219 Le Relecq-Kerhuon — B.P. 16
Tel. (98) 28 03 03

Celdis S.A.
53, rue Charles Fréret — 94250 Gentilly
Tel. (01) 581 00 20

Ets. F. Feutrier S.A. (Main Office)
Rue des Trois-Glorieuses
42270 St-Priest-en-Jarez (St Etienne)
Tel. (77) 74 67 33

Ets. F. Feutrier S.A.
Avenue Laplace — Zone industrielle
13470 Carnoux
Tel. (42) 82 16 41

Feutrier Ile de France
29, rue Ledru-Rollin — 92150 Suresnes
Tel. (01) 772 46 46

Ets. Gros S.A. (Main Office)
13, rue Victor-Hugo — B.P. 63
59050 Saint-André-lez-Lille
Tel. (20) 51 21 33

Ets. Gros S.A.
14, avenue du Général-Leclerc — 54000 Nancy
Tel. (83) 35 17 35

Ets. Gros S.A.
5, rue Pascal — 94800 Villejuif
Tel. (01) 678 27 27

S.C.A.I.B. S.A.
80, rue d'Arcueil — Silic 137
94523 Rungis Cedex
Tel. (01) 687 23 13

Sté Commerciale Toutélectric (Main Office)
15-17, Boulevard Bonrepos — 31008 Toulouse
Tel. (61) 62 11 33

Sté Commerciale Toutélectric
80-83, quai des Queyries — 33100 Bordeaux
Tel. (56) 86 50 31

GERMANY
Alfred Neye — Enatechnik GmbH
Schillerstrasse 14 — D-2085 Quickborn/Hamburg
Tel. (04106) 6121

Alfred Neye — Enatechnik GmbH
Brunowstrasse 7 — D-1000 Berlin 27
Tel. (030) 433 30 52

Alfred Neye — Enatechnik GmbH
Hildesheimerstrasse 31 — D-3000 Hannover
Tel. (0511) 88 60 86

Alfred Neye — Enatechnik GmbH
Birkenstrasse 107 — D-4000 Düsseldorf
Tel. (0211) 66 61 45

Alfred Neye — Enatechnik GmbH
Rheinstrasse 24 — D-6100 Darmstadt
Tel. (06151) 2 64 46

Alfred Neye — Enatechnik GmbH
Breitwiesenstrasse 25 — D-7000 Stuttgart 80
Tel. (0711) 73 63 57

Alfred Neye — Enatechnik GmbH
Maria-Theresia-Strasse 6 — D-8000 München 80
Tel. (089) 47 30 23

Distron GmbH & Co.
Behainstrasse 3 — D-1000 Berlin 10
Tel. (030) 342 10 41/45

EBV Elektronik Vertriebs-GmbH (Main Office)
Oberweg 6 — D-8025 Unterhaching
Tel. (089) 611 051

EBV Elektronik Vertriebs-GmbH
In den Meinerorth 9 a — D-3006 Burgwedel 1/Hannover
Tel. (05139) 5038

EBV Elektronik Vertriebs-GmbH
Oststrasse 129 — D-4000 Düsseldorf
Tel. (0211) 8 48 46

EBV Elektronik Vertriebs-GmbH
Myliusstrasse 54 — D-6000 Frankfurt 1
Tel. (0611) 72 04 16

EBV Elektronik Vertriebs-GmbH
Alexanderstrasse 42 — D-7000 Stuttgart 1
Tel. (0711) 24 74 81

Jermyn GmbH
Postfach 1180 — D-6277 Camberg
Tel. (06434) 23 1

Jermyn GmbH
Baumgartenring 32 — D-7403 Ammerbuch 1
Tel. (07073) 6041/6042

Jermyn GmbH
Rathelbeckstrasse 282 — D-4000 Düsseldorf 12
Tel. (0211) 20 30 94/20 30 95

MÜTRON, Müller & Co. KG
Bornstrasse 22 — D-2800 Bremen 1
Tel. (0421) 3104 85

MÜTRON, Müller & Co. KG
Theodor-Heuss-Ring 28 — D-5000 Köln 1
Tel. (0221) 12 24 24

MÜTRON, Müller & Co. KG
Asbeckstrasse 18 — D-2100 Hamburg 90
Tel. (040) 765 30 28

RTG, E. Springorum GmbH + Co. (Main Office)
Postfach 426 — D-4600 Dortmund 1
Tel. (0231) 5 49 51

RTG, E. Springorum GmbH + Co.
Friedrich-Ebert-Damm 112 — D-2000 Hamburg 70
Tel. (040) 693 70 61/62

RTG, E. Springorum GmbH + Co.
Ungererstrasse 43 — D-8000 München 40
Tel. (089) 36 65 00

RTG, E. Springorum GmbH + Co.
Reutlinger Strasse 87 — D-7000 Stuttgart-Degerloch
Tel. (0711) 76 64 28

RTG, E. Springorum GmbH + Co.
Mendelssohn-Bartholdy-Strasse 6 — D-6200 Wiesbaden
Tel. (06121) 52 73 09

SASCO Vertrieb von elektronischen Bauelementen GmbH (Main Office)
Hermann-Oberth-Strasse 16 — D-8011 Putzbrunn b. München
Tel. (089) 46 11 211

SASCO Vertrieb von elektronischen Bauelementen GmbH
Postfach 3066 — D-4005 Düsseldorf/Meerbusch 3
Tel. (02150) 14 33

SASCO Vertrieb von elektronischen Bauelementen GmbH
Postfach 270 214 — D-3000 Hannover
Tel. (0511) 66 25 86

SASCO Vertrieb von elektronischen Bauelementen GmbH
Lorenzer Strasse 15 — D-8500 Nürnberg
Tel. (0911) 20 41 52

SASCO Vertrieb von elektronischen Bauelementen GmbH
Stafflenbergstrasse 24 — D-7000 Stuttgart 1
Tel. (0711) 24 45 21

SPOERLE Electronic
Otto-Hahn-Strasse 13 — D-6072 Dreieich b. Frankfurt
Tel. (06103) 3 04 1

SPOERLE Electronic
Grosse Witschgasse 9-11 — D-5000 Köln 1
Tel. (0221) 23 50 98

SPOERLE Electronic
Zweibrueckenstrasse 1 — D-8000 München 2
Tel. (089) 22 74 17

Technoprojekt GmbH (Main Office)
Heinrich-Ebner-Strasse 13 — D-7000 Stuttgart-Bad Cannstatt
Tel. (0711) 56 17 12

Technoprojekt GmbH
Vertriebsbüro Dortmund
Schildstrasse 4 — D-4600 Dortmund 30
Tel. (0231) 43 36 82

GREECE
Macedonian Electronics Ltd.
Chantaou — P.O. Box 240 — Thessaloniki
Tel. 30 68 88

Macedonian Electronics Ltd.
Lloyd George 10 — Athens
Tel. (21) 360 95 71

HOLLAND
B.V. Diode
Hollantlaan 22 — Utrecht
Tel. (030) 88 42 14

Manudax Nederland B.V.
Meerstraat 7
5473 ZG Heeswijk (N.B.) — P.O. Box 25
Tel. (41) 39 12 52

HUNGARY
Interag Co., Ltd.
XIII Rajk Laszlo u. 11 — P.O. Box 184
1330 Budapest
Tel. (1) 32 93 40

IRAN
Milcom LTD, Motorola Building
Niloo Street, Vanak Square — Teheran
Tel. 66 12 14/15

ITALY
Celdis Italiana S.p.A. (Main Office)
Via F.lli Gracchi 36 — 20092 Cinisello Balsamo (MI)
Tel. (02) 612 00 41

Celdis Italiana S.p.A.
Via Lorenzo il Magnifico 109 — 00162 Roma
Tel. (06) 42 38 55/42 71 550

Celdis Italiana S.p.A.
Via Turati 33 — 40055 Castenaso (Bologna)
Tel. (051) 78 80 78/78 70 34

Celdis Italiana S.p.A.
Via Mombarcaro 96 — 10136 Torino
Tel. (11) 35 93 12/35 93 69

Celdis Italiana S.p.A.
Via Anconitano 6/4 — Padova
Tel. (049) 68 77 09

Cramer Italia S.p.A. (Main Office)
Via Cristoforo Colombo 134 — 00147 Roma
Tel. (06) 51 79 81

Cramer Italia S.p.A.
Via S. Simpliciano 2 — 20100 Milano
Tel. (02) 80 93 26

Cramer Italia S.p.A.
Terza Traversa Domenico Fontana, 22A/B — Napoli
Tel. 25 53 09

Cramer Italia S.p.A.
Via Ferraiese 192 — Bologna
Tel. (051) 37 27 77

Cramer Italia S.p.A.
Corso Traiano 109 — 10135 Torino
Tel. (011) 619 20 62/619 20 67

Silverstar Ltd. S.p.A.
Via dei Gracchi 20 — 20146 Milano
Tel. (02) 49 96

Silverstar Ltd. S.p.A.
Via Palsiello 30 — 00198 Roma
Tel. (06) 844 88 41

Silverstar Ltd. S.p.A.
Piazza Adriano 9 — 10139 Torino
Tel. (011) 44 32 7/8

Silverstar Ltd. S.p.A.
Via S. Sofia 15 — 35100 Padova
Tel. 22 33 8

NIGERIA
Beckron International
14, Alhaji Basheron Street
SW Ikoyi — P.O. Box 1896 — Lagos
Tel. 5 66 29

NORWAY
Ola Tandberg Elektro A/S
Skedsmogaten 25 — Oslo 25
Tel. (02) 19 70 30

POLAND
PHZ Transpol S.A. (Intraco Building)
Ul. Stawki 2 — 00-950 Warsaw 1
Tel. (004822) 39 50 79

PORTUGAL
Equipamentos de Laboratorio LDA
Rua Pedro Nemes 47 — Lisbon 1
Tel. 97 02 51

SOUTH AFRICA
L'Electron
704 Main Pretoria Road, Wynberg Tvl
P.O. Box 10544, Johannesburg 2000
Tel. 40 62 96

SPAIN
Hispano Electronica S.A. (Main Office)
Polígono Industrial "Ultimsa"
Apartado de Correos 48 — Alcorcón (Madrid)
Tel. (91) 619 41 08

Hispano Electronica S.A.
Figols, 27-29
Barcelona 14 — Tel. 259 05 22/23

SWEDEN
Interelko AB.
Box 32 — 12221 Enskede
Tel. (08) 13 21 60

AB Gösta Bäckström
Alströmergatan 22 — Box 12099
10221 Stockholm
Tel. (08) 54 10 80

SWITZERLAND
Elbatex AG
Alb. Zwyssig-Strasse 28 — 5430 Wettingen
Tel. (056) 26 56 41

Omni Ray AG
Dufourstrasse 56 — 8008 Zürich
Tel. (01) 34 07 66

TURKEY
ERA Elektronik Sanayi Ve Ticaret A.S.
Eski Buyukdere Cad. 49/A. 4 Levent
Istanbul
Tel. 64 65 00

ERA Elektronik Sanayi Ve Ticaret A.S.
Gazi Mustafa Kemal Bul. 12
Onur Is Hani Kat4 D79 — Yenisehir/Ankara
Tel. 25 49 33

UNITED KINGDOM
A.M. Lock & Co. Ltd.
Neville Street, Middleton Road
Oldham, Lancs OL96LF
Tel. (061) 652 04 31

Celdis Ltd.
37-39 Lowreck Road
Reading, Berks, RG3, 1ED
Tel. (0734) 585 171

Hawke Electronics Ltd.
Hawke House Green Street
Sunbury on Thames, Middlesex, England
Tel. (9327) 8 55 77

Crellon Electronics Ltd.
380, Bath Road
Slough, Berks SL1 6JE
Tel. (06286) 44 34

ITT Electronic Services
Edinburgh Way
Harlow, Essex CM20 (2DF)
Tel. Harlow (0279) 26 777

Jermyn Industries
Vestry Estate — Sevenoaks, Kent
Tel. (732) 5 11 74

Macro-Marketing Ltd.
396, Bath Road
Slough, Berks SL1 6JD
Tel. (06286) 4422

YUGOSLAVIA
Elektrotehna Ljubljana
Export Import
Titova 51 — P.O. Box 34-1
61000 Ljubljana
Tel. (61) 32 02 41

Elektrotehna Ljubljana
Filiala Beograd
Marsala Tita 6/1
11000 Beograd
Tel. (011) 69 69 24

EUROPEAN SEMICONDUCTOR FACTORIES

FRANCE
Motorola Semiconducteurs S.A.
Canto Lauzetto — Le Mirail
31023 Toulouse CEDEX
Tel. (61) 40 11 88

GERMANY
Motorola GmbH
Munchner Strasse 18
8043 Unterföhring
Tel. (089) 92 481

UNITED KINGDOM
Motorola Semiconductors Ltd.
Colvilles Road, Kelvin Estate
East Kilbride/Glasgow (Scotland)
Tel. (3552) 39 101

TABLE OF CONTENTS Page

TABLE OF CONTENTS Page

* * * * * * * * * *

CHAPTER 1

1.0 INTRODUCTION

The M6809 Absolute Assembler is a program that processes source program statements written in M6800 or M6809 assembly language. The assembler translates these source statments into object programs compatible with XDOS or EXORbug loaders, and produces a listing of the source program. The M6809 Absolute Assembler has been designed to operate on Motorola's EXORset development system.

1.1 Assembly Language

The symbolic language used to code source programs to be processed by the assembler is called assembly language. The language is a collection of mnemonic symbols representing : operations (i.e, machine instruction mnemonics, or directives to the assembler), symbolic names, operators, and special symbols.

The assembly language provides mnemonic operation codes for all machine instructions in the M6809 instruction set. The M6809 instructions are defined and explained in the "M6809 Programming Reference Manual". The assembly language also contains mnemonic directives which specify auxiliary actions to be performed by the assembler. These directives are not always translated into machine language.

1.2 Operating Environment

The minimum hardware requirements for the Absolute Assembler include :

Motorola's EXORset development system with EXORbug monitor

M6809 XDOS version -- 24k RAM

1.3 Assembler Processing

The M6809 Absolute Assembler is a two-pass assembler. During the first pass the source program is read to develop the symbol table. During the second pass the object file is created (assembled) with reference to the table developed in pass one. It is during the second pass that the source program listing is also produced.

Each source statment is processed completely before the next statment is read. As each statment is processed, the assembler examines the label, operation code, and operand

fields. The operation code table is scanned for a match with a known opcode.

During the processing of a standard operation code mnemonic, the machine code is inserted into the object file.

Any errors that are detected by the assembler are displayed before the actual line containing the error is printed. Errors are accumulated and the total number of errors is printed at the end of each source listing. If no source listing is being produced, error messages are still displayed to indicate that the assembly process did not proceed normally.

CHAPTER 2

2.0 CODING ASSEMBLY LANGUAGE PROGRAMS

Programs written in assembly language consist of a sequence of source statements. Each source statement consists of a sequence of ASCII characters ending with a carriage return. Appendix A contains a list of the supported character set.

2.1 Source Statement Format

Each source statement may include up to 5 fields : a sequence number, a label (or "*" for a comment line), an operation, an operand, and a comment.

2.1.1 Sequence number

The sequence number field is an optional field provided as a programming convenience. The sequence number field starts at the beginning of the source line and consists of up to five decimal digits. The value of the number must be less than 65536. Sequence numbers must be followed by a space.

Although sequence numbers are optional, they must be consistently used or not used for an entire program. If the first source statement has a sequence number, then every succeeding source statement must also have a sequence number. If the first source statement does not have a sequence number, then no other source statement may be numbered.

2.1.2 Label field

The label field occurs as the first field of a source program. The label field can take one of the following forms:

1. An asterisk (*) as the first character in the label field indicates that the rest of the source statement is a comment. Comments are ignored by the assembler and are printed on the source listing only for the programmer's information.

2. A space as the first character indicates that the label field is empty. The line has no label and is not a comment.

3. A symbol character as the first character indicates that the line has a label. Symbol characters are the upper case letters A-Z, the digits 0-9, and the special characters period (.), dollar sign ($), and underscore (_). Symbols consist of 1 to 6 characters,

the first of which must be alphabetic or the special character period (.). Certain special symbols are reserved by the assembler and will cause an error to be generated if they appear in a label field. These reserved symbols are : A, B, D, CC, DP, PC, PCR, S, U, X, and Y.

A symbol may occur only once in a label field unless it is used with the SET directive. If a symbol does occur more than once in a label field, then each reference to that symbol will be flagged with an error.

With the exception of some directives, a label is assigned the value of the program counter of the first byte of the instruction or data being assembled.

Each unique label and undefined symbol in a program is allocated a ten-byte block in the symbol table. In addition, a ten-byte block is allocated for every four references to a symbol, if the cross reference option (section 3.8) is in effect.

2.1.3 Operation field

The operation field occurs after the label field and must be preceded by at least one space. The operation field must contain a symbol. Thus, the rules governing labels apply to the operation field as well. Entries in the operation field may be one of two types :

Op code — These correspond to the machine instructions. The operation code includes the "A", "B" or "D" character for the accumulator specification. For compatibility with other M680X assemblers, a single space may separate the operation code from the accumulator designator. For example, "LDA A" is the same as "LDAA". Although the M6809 assembler recognizes the above instruction forms (Appendix 3.5), the proper form for the M6809 instruction "load accumulator A" is "LDA".

Directive — These are special operation codes known to the assembler which control the assembly process rather than being translated into machine instructions.

The assembler first searches for operation codes in an internal table of machine operation codes and assembler directives. If neither of the tables holds the specified operation code, an error message is printed. If code is being generated, three bytes of zeros are generated for an invalid operation code.

2.1.4 Operand field

The interpretation of the operand field is dependent on its contents. The operand field, if required, must follow the operation field, and must be preceded by at least one space. The operand field may contain a symbol, an expression, or a combination of symbols and expressions separated by commas.

The operand field of machine instructions is used to specify the addressing mode of the instruction. The format of the operand field for M6809 instructions is summarized in the following table :

Operand Format	M6809 Addressing Mode
<<expression>	direct
><expression>	extended
[<expression>]	extended indirect
<expression>,R	indexed
<<expression>,R	8-bit offset indexed
><expression>,R	16-bit offset indexed
[<expression>,R]	indexed indirect
<[<expression>,R]	8-bit offset indexed indirect
>[<expression>,R]	16-bit offset indexed indirect
Q+	auto increment by 1
Q++	auto increment by 2
[Q++]	auto increment indirect
-Q	auto decrement by 1
--Q	auto decrement by 2
[--Q]	auto decrement indirect

where R is one of the registers PCR, S, U, X, or Y and Q is one of the registers S, U, X, or Y.

The operand fields of assembler directives are described in Chapter 3.

2.1.4.1 M6809 addressing modes

The M6809 includes some instructions which require no operands. These instructions are self-contained and employ the inherent addressing or the accumulator addressing mode.

IMMEDIATE ADDRESSING. Immediate addressing refers to the use of one or two bytes of information that immediately follow the operation code in memory. Immediate addressing is indicated by preceding the operand field with the pound sign character "#" (i.e., #<expression>). The expression following the "#" will be assigned one or two bytes of storage depending on the instruction. All instructions referencing the accumulator "A" or "B", or the condition code register "CC" will generate a one byte immediate value. Also,

immediate addressing used with the PSHS, PULS, PSHU, and PULU instructions generates a one byte immediate value. Immediate operands used in all other instructions generate a two byte value.

The register list operand does not take the form #<expression> but still generates one byte of immediate data. The form of the operand is :

R1[,R2,....,Rn]

where Ri (i=1 to n) is one of the symbols A, B, CC, D, DP, PC, S, U, X, or Y. The number and type of symbols vary depending on the specific instruction.

For the instructions PSHS, PULS, PSHU, and PULU any of the above register names may be included in the register list. The only restriction is that "U" cannot be specified with PSHU or PULU, and "S" cannot be specified with PSHS and PULS. The one byte immediate value assigned to the operand is determined by the registers specified. Each register name sets a bit in the immediate byte as follows :

Register	Bit
PC	7
U,S	6
Y	5
X	4
DP	3
B,D	2
A,D	1
CC	0

(Section 3.11 contains a detailed explanation of immediate expressions with the PSH/PUL instructions.)

For the instructions EXG and TFR, exactly two of the above register names must be included in the register list. The other restriction is the size of the registers specified. For the EXG instruction, the two registers must be the same size, or the first can be a 16-bit register and the second an 8-bit register. The 8-bit registers are A, B, CC, and DP. The 16-bit registers are D, PC, S, U, X, and Y. The one byte immediate value assigned to the operand is determined by the register names. The most significant 4 bits of the immediate byte contain the value of the first register name; the least significant 4 bits contain the value of the second register as shown by the following table :

Register	Value (hex)
D	0
X	1
Y	2
U	3
S	4
PC	5
A	8
B	9
CC	A
DP	B

RELATIVE ADDRESSING. Relative addressing is used by branch instructions. There are two forms of the branch instruction. The short branch can only be executed within the range -126 to +129 bytes relative to the first byte of the branch instruction. The actual branch offset is put into the second byte of the branch instruction. The long branch can execute in the full range of addressing from 0000 to FFFF (hexadecimal) because a two byte offset is calculated and put into the operand field of the branch instruction. The offset is the two's complement of the difference between the location of the byte immediately following the branch instruction and the location of the destination of the branch.

DIRECT AND EXTENDED ADDRESSING. Direct and extended addressing utilize one (direct) or two (extended) bytes to contain the address of the operand. Direct and Extended addressing are indicated by only having an expression in the operand field (i.e., <expression>). Direct addressing will be used by the M6809 assembler whenever possible. References to expressions with values having the most significant byte of the expression the same as the current value of the direct page pseudo register (section 3.13) will automatically be assembled with the direct addressing mode. If a symbol is referenced before it has been defined, the instruction will be assembled with the extended addressing mode in order to avoid phasing errors. All other cases will result in extended addressing mode being used.

Regardless of the criteria described above, it is possible to force the assembler to use the direct addressing mode by preceding the operand with the "<" character. Similarly, extended addressing can be forced by preceding the operand with the ">" character. These operand forms are : <<expression> and ><expression>. There is no restriction on the latter form. It will always generate extended addressing. If direct addressing is forced, the following check is made : the most significant byte of the expression is compared with the direct page pseudo register. If they are not the same, a warning message is generated. Again, the user must ensure that the direct page register is set up properly at execution time.

INDEXED ADDRESSING. Indexed addressing is relative to one of the index registers. The general form is <expression>,R. The address is calculated at the time of instruction execution by adding the value of <expression> to the current contents of the index register. The other general form is [<expression>,R]. In this indirect form, the address is calculated at the time of instruction execution by first adding the value of <expression> to the current value of the index register and then retrieving the two bytes from the calculated address and address+1. This two byte value is used as the effective address of the operand. The allowable forms of indexed addressing are described below. Appendix B.5 describes the format of the post-byte (i.e., the byte immediately following the op-code) for each of the indexed addressing modes. In the description below, R refers to one of the index registers S, U, X, or Y.

The accumulator offset mode allows one of the accumulators to be specified instead of an <expression>. Valid forms are :

<acc>,R and [<acc>,R]

where <acc> is one of the accumulators A, B, or D. This form generates a one byte operand (post-byte only).

The valid forms for the automatic increment/decrement mode are shown below. For each row, the three entries shown are equivalent.

R+	,R+	0,R+
-R	,-R	0,-R
R++	,R++	0,R++
--R	,--R	0,--R
[R++]	[,R++]	[0,R++]
[--R]	[,--R]	[0,--R]

In this form the only valid expression is 0. Like the accumulator offset mode, this form generates a one byte operand (post-byte only).

The valid forms for the expression offset mode are :

R	,R	<expression>,R
[R]	[,R]	[<expression>,R]
<R	<,R	<<expression>,R
<[R]	<[,R]	<[<expression>,R]
>R	>,R	><expression>,R
>[R]	>[,R]	>[<expression>,R]

The "<" and ">" characters force an 8 or 16-bit offset, respectively, and are described bellow. If no expression is specified, only the post-byte of the operand is generated. If

an expression with a value in the range -16 to +15 is specified without indirection, a one byte operand is generated which contains the expression's value as well as the index register indicator. At execution time, the expression's value is expanded to 16-bits with sign extension before being added to the index register.

All other forms will generate a post-byte as well as either a one or two byte offset which contains the value of the expression. The size of the offset is determined by the type and size of the expression. Expressions with values in the range -128 to +127 generate an 8-bit offset. If an expression that follows the above rules contains a symbol that is referenced before it has been defined, the instruction will be assembled using a 16-bit offset in order to avoid phasing errors. All other cases will result in a 16-bit offset being generated. In the case where an 8-bit offset is generated, the value is expanded to 16-bit with sign extension at execution time.

Regardless of the criteria described above, it is possible to force the assembler to generate an 8-bit offset by preceding the operand the operand with the "<" character. Similarly, a 16-bit offset can be forced by preceding the operand with the ">" character. There is no restriction on the ">" form. It always generates a post-byte followed by a 16-bit offset. If an 8-bit offset is forced, the expression must not have a value outside the -128 to +127 range or a byte overflow error is generated.

The valid forms for the program counter relative mode are exactly the same as the expression offset mode with the exeption that the index register specification must be "PCR". However, the manner in which the offset is generated by the assembler differs. The assembler generates a relative address which is then used as the 8 or 16-bit offset following the post-byte. The relative address is the two's complement of the difference between the location of the byte immediately following the indexed instruction and the value of the expression.

If the relative address calculated is not in the range -128 to +127, or if the expression references a symbol that has not yet been defined, a two byte offset is generated after the post-byte. A one byte offset is generated if the relative address is in the range -128 to +127.

Like the expression offset mode, a one byte offset can be forced by preceding the operand with a "<". A ">" forces a two byte offset. A byte overflow error is generated if a one byte offset is forced when the relative address is not in the range -128 to +127.

The extended indirect mode has the form [<expression>]. Although extended indirect is a logical extension of the extended addressing mode, this mode is implemented using an encoding of the post-byte under the indexed addressing mode. A post-byte is generated as well as a two-byte offset which

contains the value of the expression.

2.1.4.2 Expressions

An expression is a combination of symbols, constants, algebraic operators, and parentheses. The expression is used to specify a value which is to be used as an operand. Expressions follow the conventional rules of algebra.

2.1.4.3 Operators

The precedence of the various operators is as follows : parenthetical expressions are evaluated first, with the inner most parentheses being processed before the outer ones. Next, the multiplication (*), division (/), and all two-character operators have precedence. Of the lowest precedence are the addition (+) and subtraction (-) operators. Unary minus can only occur at the beginning of an expression or immediately before a left parenthesis. Unary minus is equivalent in evaluation to putting a zero directly before the minus sign. For example, the following expressions are all equivalent :

```
 -TAG1*INDEX+3
0-TAG1*INDEX+3
-(TAG1*INDEX)+3
```

Operators of the same precedence are evaluated from left to right. All intermediate results in the computation of an expression are truncated to a 16-bit integer value. The result of an expression is also a 16-bit integer. Operators can operate on numeric constants, single character ASCII literals, and symbols.

In addition to the normal operators for multiplication, division, addition, and subtraction, the assembler recognizes certain two-character operators. These operators are infix operators and have the same precedence as multiplication or division. Each two-character operator begins with an exclamation mark (!) and takes two operands. The following two-character operators are defined :

!^ - exponentiation	The left operand is raised to the power specified by the right operand. If the right operand is zero, the resulting value will be "1", regardless of the value of the left operand.
!. - logical AND	Each bit in the left operand is logically "ANDed" with the corresponding bit in the right operand.

!+ - inclusive OR	Each bit in the left operand is inclusively "ORed" with the corresponding bit in the right operand.
!X - exclusive OR	Each bit in the left operand is exclusively "ORed" with the corresponding bit in the right operand.
!< - shift left	The left operand is shifted to the left by the number of bits specified by the right operand. The left operand is zero-filled from the right.
!> - shift right	The left operand is shifted to the right by the number of bits specified by the right operand. The left operand is zero-filled from the left.
!L - rotate left	The left operand is rotated left by the number of bits specified by the right operand. The most significant bit is rotated into the least significant bit position of the left operand.
!R - rotate right	The left operand is rotated right by the number of bits specified by the right operand. The least significant bit is rotated into the most significant bit position of the left operand.

2.1.4.4 Symbols

Each symbol is associated with a sixteen-bit, integer value which is used in place of the symbol during the expression evaluation.

The asterisk (*) is a special symbol to the assembler and can only be used in an expression. The asterisk used in an expression as a symbol represents the current value of the location counter (the first byte of a multi-byte instruction).

2.1.4.5 Constants

Constants represent quantities of data that do not vary in value during execution of a program. The numeric constants can be in one of for bases : decimal, hexadecimal, binary, or octal.

A decimal constant consists of a string of numeric digits. The value of a decimal constant must fall in the range 0-65535, inclusive. Optionally, decimal constants may be preceded by the ampersand character "&". The following example shows both valid and invalid decimal constants :

VALID	INVALID	REASON INVALID
12	123456	more than 5 digits
12345	12.3	invalid character
&65201	67800	out of range (> 65535)

A hexadecimal constant consists of a maximum of four characters from the set of digits (0-9) and the upper case, alphabetic letters (A-F) and is preceded by a dollar sign "$". Hexadecimal constants can also be designated by being followed by the letter "H". In this case, the first digit of the hexadecimal constant must be a numeric so that the constant can be distinguished from a symbol name. Hexadecimal constants must be in the range $0000 to $FFFF. The following example shows both valid and invalid hexadecimal constants :

VALID	INVALID	REASON INVALID
$12	ABCD	no preceding "$"
0ABCDH	$G2A	invalid character
$001F	$2F018	too many digits

A binary constant consists of a maximum of 16 ones or zeroes preceded by a percent sign "%". Binary constants can also be represented by a series of ones and zeroes succeeded by the letter "B". The following example shows both valid and invalid binary constants :

VALID	INVALID	REASON INVALID
%00101	1010101	missing percent
%1	%10011000101010111	too many digits
10100B	%210101	invalid digit

An octal constant consists of a maximum of six numeric digits, excluding the digits 8 and 9, preceded by a commercial at-sign "@". Octal constants can also be designated by ending in the letter "O" or "Q". Octal constants must be in the range @0 to @177777. The following example shows both valid and invalid octal constants :

VALID	INVALID	REASON INVALID
@17634	@2317234	too many digits
377Q	@277272	out of range
177600O	23914Q	invalid character

Character constants can be used in expressions if they are single characters. Character constants are preceded by a

single quote. Any character, including the single quote, can be used as a character constant. The following example shows both valid and invalid character constants :

VALID	INVALID	REASON INVALID
'*	'VALID	too long

2.1.5 Comment field

The last field of an assembler source statment is the comment field. This field is optional and is only printed on the source listing for documentation purposes. The comment field is separated from the operand field (or from the operation field if no operand field is required) by at least one space. The comment field can contain any printable ASCII characters.

2.2 Assembler Output

The assembler output includes an optional listing of the source program and an optional object file which is in one of the following formats : EXORciser loadable format, XDOS loadable memory image. Appendix E contains the description of the source listing formats.

The assembler will normally suppress the printing of the source listing and select the generation of an object output file. These conditions, as well as others, can be overridden via options supplied on the command line that invoked the assembler.

The assembly source program listing contains the original source statments, formatted for easier reading, as well as additional information which is generated by the assembler. Most lines in the listing correspond directly to a source statment. Lines which do not correspond directly to source statments include : page headings, error messages, directives as FCB, FCC, and FDB.

The assembly listing may optionally contain a symbol table or a cross reference table of all symbols appearing in the program. These are always printed after the END directive if either the symbol table or cross reference table options (section 3.8) are in effect. The symbol table contains the name of each symbol along with its defined value. The cross reference table additionally contains the assembler-maintained source line number of every reference to every symbol. The format of the cross reference table is shown in Appendix E.3

CHAPTER 3

3.0 ASSEMBLER DIRECTIVES

The assembler directives are instructions to the assembler rather than instructions to be directly translated into object code. This chapter describes the directives that are recognized by the assembler. Detailed descriptions of each directive are arranged alphabetically. The notation used in this chapter is :

{ } Contains a list of elements, one of which must be selected. Each choice will be separated by a vertical bar.

[ ] Contains an optional element. If one of a series of elements may be selected, the available list of choices will be contained within the brackets. Each choice will be separated by a vertical bar

XYZ The names of the directives are printed in capital letters. The required parts of directive operands will also be printed in capital letters. All elements outside of the angle brackets (<>) must be specified as-is. For example the syntactical element [<number>,] requires the comma to be specified if the optional element <number> is selected.

< > The element names are printed in lower case and contained in angle brackets. The following elements are used in the subsequent descriptions:

<comment>	Statement comment field
<label>	Statement label
<expression>	Assembler expression
<expr>	Assembler expression
<number>	Numeric constant
<string>	String of ASCII characters
<delimiter>	String delimiter
<option>	Assembler option
<symbol>	Assembler symbol
<sym>	Assembler symbol
<reg list>	M6809 register list
<reg exp>	M6809 register expression

In the following descriptions of the various directives, the syntax, or format, of the directive is given first. This will be followed by the directive's description.

3.1 BSZ -- Block Storage Of Zeroes

[<label>] BSZ <expression> [<comment>]

The BSZ directive causes the assembler to allocate a block of bytes. Each byte is assigned an initial value of zero. The number of bytes allocated is given by the expression in the operand field. If the expression contains symbols that are undefined, forward references, or if the expression has a value of zero, an error will be generated.

3.2 END -- End Of Source Program

END [<expression> [<comment>]]

The END directive indicates that the logical end of the source program has been encountered. Any statements following the END directive are ignored. If the END directive is not encountered before the physical end of the source file is found, an error message will be generated. However, this error message is only a warning. The optional expression in the operand field can be used to specify the starting execution address of the program.

3.3 EQU -- Equate Symbol To A Value

<label> EQU <expression> [<comment>]

The EQU directive assigns the value of the expression in the operand field to the label. The EQU directive is one of the directives that assignes a value other than the program counter to the label. The label cannot be redefined anywhere else in the program. The expression cannot contain any forward references or undefined symbols.

3.4 FCB -- Form Constant Byte

[<label>] FCB {<expr>[,<expr>,...,<expr>]} [<comment>]

The FCB directive may have one or more operands separated by commas. The value of each operand is truncated to eight bits and is stored in a single byte of the object program. Multiple operands are stored in successive bytes. The operand may be a numeric constant, a character constant, a symbol, or an expression. If multiple operands are present, one or more of them can be null (two adjacent commas), in which case a single byte of zero will be assigned for that operand. An error will occur if the upper eight bits of the evaluated operand's values are not all ones or all zeroes.

3.5 FCC -- Form Constant Character Strings

[<label>] FCC <number>,<string> [<comment>]

or

[<label>] FCC <delimiter><string><delimiter> [<comment>]

The FCC directive is used to store ASCII strings into consecutive bytes of memory. Any of the printable ASCII characters can be contained in the string. The FCC directive has two formats. The first format requires that <number> be a decimal constant in the range 5. The comma is required after the decimal constant. The <number> specifies the number of characters contained in <string> which begins immediately after the comma. Should <number> be greater than the number of characters in the string (e.g., carriage return encountered in line before the specified number of characters are found), then spaces will be inserted to fill the remainder of the string.

The second format of the FCC directive specifies the string between two identical delimiters. The delimiters can be any printable ASCII character. The first non-blank character after the FCC directive will be used as the delimiter. Thus, if the delimiter happens to be a decimal digit, the first character of the string cannot be a comma.

3.6 FDB -- Form Double Byte Constant

[<label>] FDB {<expr>[,<expr>,...,<expr>]} [<comment>]

The FDB directive may have one or more operands separated by commas. The sixteen-bit value corresponding to each operand is stored into two consecutive bytes of the object program. Multiple operands are stored in successive bytes. The operand may be a numeric constant, a character constant, a symbol, or an expression. If multiple operands are present, one or more of them can be null (two adjacent commas), in which case two bytes of zeroes will be assigned for that operand.

3.7 NAM -- Assign Program Name

NAM [<string> [<comment>]]

The NAM directive is generally used as the first statment of an assembly language program. It's use, however, is optional. More than one NAM directive can be used in a program. The <string> specified will be printed on the heading line of each page of the source program listing. It will be used as the name in the S0 record if an EXORciser-loadable program is being created. The <string> consists of a maximum of six printable ASCII characters.

3.8 OPT -- Assembler Output Options

OPT <option>[,<option>,...,<option>] [<comment>]

The OPT directive is used to control the format of the assembler output. The options are specified in the operand field, separated by commas. All options have a default condition. Some options are not reset to their default conditions at the end of pass one. Some options are allowed to have the prefix "NO" attached to them, which then reverses their meaning. Most options can be initialized from the command line that invoked the assembler. In the following descriptions, the parenthetical inserts specify "DEFAULT", if the option is the default condition, and "RESET", if the option is reset to its default condition at the end of pass one. The text in the OPTION column of the following table indicates the minimum characters that are required to identify the option. Additional characters can be appended to the end of an option to make it more readable depending on programmer preference. For example, NOG can be NOGEN, L can be LIST, U can be UNASSEMBLE, etc.

OPTION	MEANING
CRE	Print a cross reference table at the end of the source program listing. This option, if used, must be specified before the first symbol in the source program is encountered.
G	Print the code generated for multiple operands of the FCB, FCC, and FDB directives.
NOG (DEFAULT, RESET)	Do not print the code generated for multiple operands of the FCB, FCC, and FDB directives.
L	Print the listing from this point on. The "L" option causes an internal list counter to be incremented. As long as the list counter is greater than zero, the subsequent source listing will be printed. If the source listing is not specified on the command line that invoked the assembler, the "L" option has no effect when encountered within the source program.
NOL (DEFAULT, RESET)	Do not print the listing from this point on (including the OPT NOL directive). The "NOL" option causes an internal list counter to be decremented. As long as the list counter is less than or equal to zero, the subsequent source listing will not be printed. Thus, the "NOL" and "L" options can be nested.
LLE=<number>	Change the number of characters to be printed per line to the decimal number specified. The default value is 72; the minimum value is 50 and the maximum value is 120.
M	Direct the output to memory. The assembler will only allow memory to be used for the object output that is beyond the end of the available, contiguous memory. If an error occurs while placing object code into memory (non-existant memory or assembler memory), an error message will be displayed and the "M" option will be disabled. This option is not to be confused with "M" command line option (see Appendix G.1).
O (DEFAULT)	Create output module. Since this option is normally selected, it needs not be specified. It instructs the assembler to create an object module (either in memory or in a file). This option can only be used once in a source program.
NOO	Do not create object output module. This option will suppress the creation of the object module even if the creation of one was specified on the command line that invoked the assembler.
P=<number>	Change the number of source statements printed per page to the decimal number specified. The default value is 58; the minimum value is 10 and the maximum value is 255.
NOP	Suppress pasing; ignore PAGE directives and do not print headings or page numbers.
W (DEFAULT, RESET)	Print warning messages.
NOW	Do not print warning messages.

ORG <expression> [<comment>]

The ORG directive changes the program counter to the value specified by the expression in the operand field. Subsequent statements are assembled into memory locations starting with the new program counter value. If no ORG directive is encountered in a source program, the program counter is initialized to zero. Expressions cannot contain forward references or undefined symbols.

3.10 PAGE -- Top Of Page

PAGE

The PAGE directive causes the assembler to advance the paper to the top of the next page. If no source listing is being produced, the PAGE directive will have no effect. The directive is not printed on the source listing.

3.11 REG -- Define Register List

<label> REG <reg list> [<comment>]

The REG directive assigns a value associated with a register list to the label. The REG directive is one of the directives that assigns a value other than the program counter to the label. The label cannot be redefined anywhere else in the program. <reg list> must be of the form :

R1[,R2,....,Rn]

where Ri (i=1 to n) is one of the symbols A, B, CC, D, DP, PC, S, U, X, or Y. An error message is generated if both U and S are specified. A warning occurs if the same register is specified more than once. Register D is the same as registers A and B.

Although <label> may be used in any expression, its value is only meaningful when used with the instructions PSHU, PULU, PSHS, and PULS. The operand for these instructions can take one of two forms :

{PSHU|PULU|PSHS|PULS} <reg list>

or

{PSHU|PULU|PSHS|PULS} #<reg exp>

<reg list> is in the same format as defined above. An error message is generated if the register list contains a "U" and the instruction is PSHU or PULU. Similarily, an error occurs if the register list contains an "S" and the instruction is PSHS or PULS. <reg exp> is of the form :

<sym 1>[!+<sym 2>!+...!+<sym n>]

where <sym i> (i=1 to n) must be defined by the REG directive. An error occurs if a PSHU/PULU instruction is followed by a <reg exp> that contains a symbol previously defined with the REG directive that contained a "U" in the register list. A similar check is made for PSHS/PULS and "S".

Valid examples

```
ALLREG REG   A,B,CC,DP,X,Y,U,PC
REGXY  REG   X,Y
REGAB  REG   A,B
       PSHS  #ALLREG
       PSHU  #REGXY!+REGAB
```

Invalid examples

```
REGUS  REG   U,S            can't specify both U and S
REGU   REG   U
       PSHU  #REGU          can't push U reg. onto U stack
REGLST REG   A,B,D          duplicate reg. name warning
       PSHS  #REGU!+REGU    duplicate reg. name warning
```

3.12 RMB -- Reserve Memory Bytes

[<label>] RMB <expression> [<comment>]

The RMB directive causes the location counter to be advanced by the value of the expression in the operand field. This directive reserves a block of memory whose length in bytes is equal to the value of the expression. The block of memory reserved is not initialized to any given value. The expression cannot contain any forward references or undefined symbols.

3.13 SETDP -- Set Direct Page Pseudo Register

SETDP <expression> [<comment>]

The SETDP directive is used to assign a value to the direct page pseudo register at assembly time. The value of the least significant byte of the expression is assigned to the direct page pseudo register. This value is then used in determining if a particular memory reference can use the direct addressing mode (section 2.1.4.2). On initialization, the pseudo direct page register is assigned the value zero. The SETDP directive can be used any number of times in an assembly. Each occurrence changes the value of the direct page pseudo register. The expression cannot contain any forward references or undefined symbols. If the most

significant byte of the expression is not zero, a warning occurs. However, the direct page pseudo register is assigned the value of the least significant byte of the expression anyway.

It should be carefully noted that the SETDP directive does not affect the Direct Page Register at execution time. The user must assume responsibility for ensuring that the assembly and run-time values are compatible. In the example :

SETDP $20

the direct page pseudo register would be set to $20, causing absolute addresses in the range $2000-$20FF to be assembled using the direct addressing mode.

3.14 SPC -- Skip Blank Lines

SPC <expression>

The SPC directive causes blank lines to be inserted into the source listing for formatting purposes. The SPC directive is not printed in the listing. The number of lines skipped is determined from the expression in the operand field. If the number of lines to be skipped would cause the listing to cross a page boundary, then the paper will only be advanced to the top of the next page. The expression's value must be greater than zero and less than 256. The expression cannot contain any undefined symbols. A source program line that contains only a carriage return will have the same effect in the source listing as the directive "SPC 1".

3.15 TTL -- Initialize Page Heading

TTL [<string>]

The TTL directive causes the heading to be initialized to the string in the operand field. Up to 60 printable characters can be specified in the string. If a carriage return is found before the 60th character, the heading will be less than 60 characters. The heading will be printed on the top of all succeeding pages until another TTL directive is encountered. The heading is normally blank except for the assembler page number.

3.16 SCALL -- System Function Call

[<label>] SCALL <expression> [<comment>]

The SCALL directive is used to generate the object code for XDOS system function calls. The system functions are accessed via the software interrupt instruction (SWI) followed by a code indicating which function is being called. The generated SWI instruction will be followed by a one byte

value specified by the <expression> in the operand field. The <expression> must have a value in the range 0 to +127 (7-bit data with sign-bit=0). This allows up to 128 functions to be accessed.

3.17 UCALL -- User Function Call

[<label>] UCALL <expression> [<comment>]

The UCALL directive is similar to the SCALL directive. It generates a SWI instruction followed by a one byte value specified by the <expression> in the operand field. In addition the sign-bit (bit-7) of the value is set to 1 to allow the interrupt handler to recognize a UCALL from an SCALL. The <expression> must also have a value in the range 0 to +127. A maximum of 128 different user functions may be accessed via the UCALL directive.

3.18 SKIP2 -- Skip Two Bytes

[<label>] SKIP2 [<comment>]

The SKIP2 directive is equivalent to a "FCB $8C" statement. The op-code of a "compare X immediate" instruction is generated. The execution of the byte will change the condition codes only (no other register affected) and cause two forward bytes to be skipped.

3.19 SKIP1 -- Skip One Byte

[<label>] SKIP1 [<comment>]

The SKIP1 directive uses the same concept as the SKIP2 directive. The op-code of a "branch never" instruction is generated (equivalent to FCB $21). No register is affected by the execution of this byte. One forward byte is skipped.

APPENDIX

A. CHARACTER SET

The character set recognized by the assembler is a subset of ASCII. The ASCII code is shown in the following figure. The following characters are recognized by the assembler :

1. The upper case letters A through Z.

2. The digits 0 through 9.

3. Four arithmetic operators : +, -, *, and /

4. The special two-character operators : !A, !>, !<, !X, !., !+, !R, and !L.

5. Parentheses in expressions : (,).

6. The special symbol characters : underscore (_), period (.), and dollar sign ($). Only the period may be used as the first character of a symbol.

7. The characters used as prefixes for constants and addressing modes :

#	Immediate addressing
$	Hexadecimal constant
&	Decimal constant
@	Octal constant
%	Binary constant
'	ASCII character constant

8. The characters used as suffixes for constants and addressing modes :

H	Hexadecimal constant
O	Octal constant
Q	Octal constant
B	Binary constant
,PCR	Indexed addressing
,S	Indexed addressing
,U	Indexed addressing
,X	Indexed addressing
,Y	Indexed addressing

9. Three separator characters : space, carriage return, and comma.

10. The character "*" to indicate comments. Comments may contain any printable characters from the ASCII set.

11. The special symbols "A", "B", and "D" to specify the accumulator in the operation code; the special symbol "*" to represent the value of the current program counter; the special symbols "PCR", "S", "U", "X", and "Y" to indicate indexed addressing in the operand

field; the special symbols "A", "B", "CC", "D", "DP", "PC", "S", "U", "X", and "Y" to indicate registers in the operand field of the TFR, EXG, PSHU, PULU, PSHS, and PULS instructions; and the special symbols "A", "B", and "D" to indicate offsets in the indexed mode.

12. The characters used to indicate indirect addressing: [and].

13. The character "<" preceding an expression to force direct addressing mode or 8-bit offset in indexed mode, and the character ">" preceding an expression to indicate extended addressing mode or 16-bit offset in indexed mode.

14. The characters used to indicate auto increment and auto decrement in the indexed mode : +, ++, -, --.

ASCII Character Set

		BITS 4 TO 6							
		0	1	2	3	4	5	6	7
	0	NUL	DLE	SP	0	@	P	\	p
	1	SOH	DC1	!	1	A	Q	a	q
B	2	STX	DC2	"	2	B	R	b	r
I	3	ETX	DC3	#	3	C	S	c	s
T	4	EOT	DC4	$	4	D	T	d	t
S	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
0	7	BEL	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
T	9	HT	EM	)	9	I	Y	i	y
O	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
3	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

B. SUMMARY OF INSTRUCTIONS

The following table lists the special symbols used in the description of M6809 Instructions.

Operation functions

=	Left side of equal sign is replaced by right side of equal sign.
[]	Evaluate contents first; groupings
()	The contents of
M()	The contents of memory specified by the parenthetical address.
+	Arithmetic addition
-	Arithmetic subtraction
*	Arithmetic multiplication
and	Boolean and
effad	M6809 effective address
or	Boolean inclusive or
xor	Boolean exclusive or
L>	Logical shift right by number of bits specified
L<	Logical shift left by number of bits specified
A>	Arithmetic shift right by number of bits specified
A<	Arithmetic shift left by number of bits specified
R>	Rotate right by number of bits specified
R<	Rotate left by number of bits specified

Operand sizes and register names

$nn	The hexadecimal number "nn"
n	A bit value of n (0 or 1)
nn	An eight-bit value of nn (00-FF)
nnnn	A sixteen-bit value of nnnn (0000-FFFF)
aa	Eight-bit address
aaaa	Sixteen-bit address
A	Accumulator A
B	Accumulator B
C	Carry condition code (bit 0 of CC)
CC	Condition code register
D	A, B accumulator pair
F	Fast interrupt condition code (bit 6 of CC)
H	Half carry condition code (bit 5 of CC)
I	Interrupt condition code (bit 4 of CC)
ii	Eight-bit immediate operand
iiii	Sixteen-bit immediate operand
N	Sign condition code (bit 3 of CC)
P	Program counter register
rl	Register list
rr	Eight-bit relative branch address
rrrr	Sixteen-bit relative branch address
S	Hardware stack pointer register
U	User stack pointer register
V	Overflow condition code (bit 1 of CC)

X	X index resister
xx	Eisht-bit indexed addressins offset
xxop	indexed operation depends on index mode (See B.4)
Y	Y index resister
Z	Zero condition code (bit 2 of CC)

Condition code symbols

T	Status bit tested and set if true, reset otherwise
0	Status bit reset by operation
1	Status bit set by operation
-	Status bit unaffected by operation
?	Programming Reference Manual contains details on setting of the status bit.

B.1 M6809 Instructions

In the following table, the "function" column for branch and long branch instructions only contains the test condition performed by the branch. The following function will be performed if the result of the test is true :

P=(P)+0002+rr (for branch)

P=(P)+0003+rrrr (for 1-byte long branch op-code)

P=(P)+0004+rrrr (for 2-byte long branch op-code)

If the result of the test is false, the following function will be performed :

P=(P)+0002 (for branch)

P=(P)+0003 (for 1-byte long branch op-code)

P=(P)+0004 (for 2-byte long branch op-code)

The functions for the instructions BSR, CWAI, DAA, EXG, JSR, LBSR, PSHS, PSHU, PULS, PULU, RTI, RTS, SEX, SWI, SWI2, SWI3, SYNC, and TFR are described in detail in the "M6809 Programming Reference Manual".

Mne-monic	Oper-and	Op-code	Function	Status F H I N Z V C
ABX	--	3A	X=(X)+(B)	- - - - - - -
ADCA	ii	89	A=(A)+ii+(C)	- T - T T T T
	aa	99	A=(A)+M(aa)+(C)	
	xxop	A9	A=(A)+xxop+(C)	
	aaaa	B9	A=(A)+M(aaaa)+(C)	
ADCB	ii	C9	B=(B)+ii+(C)	- T - T T T T
	aa	D9	B=(B)+M(aa)+(C)	
	xxop	E9	B=(B)+xxop+(C)	
	aaaa	F9	B=(B)+M(aaaa)+(C)	
ADDA	ii	8B	A=(A)+ii	- T - T T T T
	aa	9B	A=(A)+M(aa)	
	xxop	AB	A=(A)+xxop	
	aaaa	BB	A=(A)+M(aaaa)	
ADDB	ii	CB	B=(B)+ii	- T - T T T T
	aa	DB	B=(B)+M(aa)	
	xxop	EB	B=(B)+xxop	
	aaaa	FB	B=(B)+M(aaaa)	
ADDD	iiii	C3	D=(D)+iiii	- - - T T T T
	aa	D3	D=(D)+M(aa,aa+1)	
	xxop	E3	D=(D)+xxop	
	aaaa	F3	D=(D)+M(aaaa,aaaa+1)	
ANDA	ii	84	A=(A) and ii	- - - T T 0 -
	aa	94	A=(A) and M(aa)	
	xxop	A4	A=(A) and xxop	
	aaaa	B4	A=(A) and M(aaaa)	
ANDB	ii	C4	B=(B) and ii	- - - T T 0 -
	aa	D4	B=(B) and M(aa)	
	xxop	E4	B=(B) and xxop	
	aaaa	F4	B=(B) and M(aaaa)	
ANDCC	ii	1C	CC=(CC) and ii	? ? ? ? ? ? ?
ASL	aa	08	M(aa)=M(aa) A< 1	- ? - T T ? T
	xxop	68	xxop=xxop A< 1	
	aaaa	78	M(aaaa)=M(aaaa) A< 1	
ASLA	--	48	A=(A) A< 1	- ? - T T ? T
ASLB	--	58	B=(B) A< 1	- ? - T T ? T
ASR	aa	07	M(aa)=M(aa) A> 1	- ? - T T ? T
	xxop	67	xxop=xxop A> 1	
	aaaa	77	M(aaaa)=M(aaaa) A> 1	
ASRA	--	47	A=(A) A> 1	- ? - T T ? T
ASRB	--	57	B=(B) A> 1	- ? - T T ? T
BCC	rr	24	Test (C)=0	- - - - - - -
BCS	rr	25	Test (C)=1	- - - - - - -
BEQ	rr	27	Test (Z)=1	- - - - - - -
BGE	rr	2C	Test (N) xor (V)=0	- - - - - - -
BGT	rr	2E	Test (Z) or [(N) xor (V)]=0	- - - - - - -
BHI	rr	22	Test (C) xor (Z)=0	- - - - - - -
BHS	rr	24	Test (C)=0	- - - - - - -

Mne-monic	Oper-and	Op-code	Function	Status F	H	I	N	Z	V	C
BITA	ii	85	(A) and ii	-	-	-	T	T	0	-
	aa	95	(A) and M(aa)							
	xxop	A5	(A) and xxop							
	aaaa	B5	(A) and M(aaaa)							
BITB	ii	C5	(B) and ii	-	-	-	T	T	0	-
	aa	D5	(B) and M(aa)							
	xxop	E5	(B) and xxop							
	aaaa	F5	(B) and M(aaaa)							
BLE	rr	2F	Test (Z) or [(N) xor (V)]=1	-	-	-	-	-	-	-
BLO	rr	25	Test (C)=1	-	-	-	-	-	-	-
BLS	rr	23	Test (C) or (Z)=1	-	-	-	-	-	-	-
BLT	rr	2D	Test (N) xor (V)=1	-	-	-	-	-	-	-
BMI	rr	2B	Test (N)=1	-	-	-	-	-	-	-
BNE	rr	26	Test (Z)=0	-	-	-	-	-	-	-
BPL	rr	2A	Test (N)=0	-	-	-	-	-	-	-
BRA	rr	20	Tests always true	-	-	-	-	-	-	-
BRN	rr	21	Tests always false	-	-	-	-	-	-	-
BSR	rr	8D	Subroutine call	-	-	-	-	-	-	-
BVC	rr	28	Test (V)=0	-	-	-	-	-	-	-
BVS	rr	29	Test (V)=1	-	-	-	0	1	0	0
CLR	aa	0F	M(aa)=00							
	xxop	6F	xxop=00							
	aaaa	7F	M(aaaa)=00							
CLRA	---	4F	A=00	-	-	-	0	1	0	0
CLRB	---	5F	B=00	-	-	-	0	1	0	0
CMPA	ii	81	(A)-ii	-	?	-	T	T	T	T
	aa	91	(A)-M(aa)							
	xxop	A1	(A)-xxop							
	aaaa	B1	(A)-M(aaaa)							
CMPB	ii	C1	(B)-ii	-	?	-	T	T	T	T
	aa	D1	(B)-M(aa)							
	xxop	E1	(B)-xxop							
	aaaa	F1	(B)-M(aaaa)							
CMPD	iiii	10,83	(D)-iiii	-	-	-	T	T	T	T
	aa	10,93	(D)-M(aa,aa+1)							
	xxop	10,A3	(D)-xxop							
	aaaa	10,B3	(D)-M(aaaa,aaaa+1)							
CMPS	iiii	11,8C	(S)-iiii	-	-	-	T	T	T	T
	aa	11,9C	(S)-M(aa,aa+1)							
	xxop	11,AC	(S)-xxop							
	aaaa	11,BC	(S)-M(aaaa,aaaa+1)							
CMPU	iiii	11,83	(U)-iiii	-	-	-	T	T	T	T
	aa	11,93	(U)-M(aa,aa+1)							
	xxop	11,A3	(U)-xxop							
	aaaa	11,B3	(U)-M(aaaa,aaaa+1)							
CMPX	iiii	8C	(X)-iiii	-	-	-	T	T	T	T
	aa	9C	(X)-M(aa,aa+1)							
	xxop	AC	(X)-xxop							
	aaaa	BC	(X)-M(aaaa,aaaa+1)							

Mne-monic	Oper-and	Op-code	Function	Status F	H	I	N	Z	V	C
CMPY	iiii	10,8C	(Y)-iiii	-	-	-	T	T	T	T
	aa	10,9C	(Y)-M(aa,aa+1)							
	xxop	10,AC	(Y)-xxop							
	aaaa	10,BC	(Y)-M(aaaa,aaaa+1)							
COM	aa	03	M(aa)=M(aa) xor $FF	-	-	-	T	T	0	1
	xxop	63	xxop=xxop xor $FF							
	aaaa	73	M(aaaa)=M(aaaa) xor $FF							
COMA	---	43	A=(A) xor $FF	-	-	-	T	T	0	1
COMB	---	53	B=(B) xor $FF	-	-	-	T	T	0	1
CWAI	ii	3C	(CC) and ii and wait for interrupt	?	?	?	?	?	?	?
DAA	---	19	Converts binary add of BCD into BCD	-	-	-	T	T	T	T
DEC	aa	0A	M(aa)=M(aa)-01	-	-	-	T	T	?	-
	xxop	6A	xxop=xxop-01							
	aaaa	7A	M(aaaa)=M(aaaa)-01							
DECA	---	4A	A=(A)-01	-	-	-	T	T	?	-
DECB	---	5A	B=(B)-01	-	-	-	T	T	?	-
EORA	ii	88	A=(A) xor ii	-	-	-	T	T	0	-
	aa	98	A=(A) xor M(aa)							
	xxop	A8	A=(A) xor xxop							
	aaaa	B8	A=(A) xor M(aaaa)							
EORB	ii	C8	B=(B) xor ii	-	-	-	T	T	0	-
	aa	D8	B=(B) xor M(aa)							
	xxop	E8	B=(B) xor xxop							
	aaaa	F8	B=(B) xor M(aaaa)							
EXG	r1	1E	Exchange 2 registers	?	?	?	?	?	?	?
INC	aa	0C	M(aa)=M(aa)+01	-	-	-	T	T	?	-
	xxop	6C	xxop=xxop+01							
	aaaa	7C	M(aaaa)=M(aaaa)+01							
INCA	---	4C	A=(A)+01	-	-	-	T	T	?	-
INCB	---	5C	B=(B)+01	-	-	-	T	T	?	-
JMP	aa	0E	P=aa	-	-	-	-	-	-	-
	xxop	6E	P=xxop							
	aaaa	7E	P=aaaa							
JSR	aa	9D	Subroutine call	-	-	-	-	-	-	-
	xxop	AD	" "							
	aaaa	BD	" "							
LBCC	rrrr	10,24	Test (C)=0	-	-	-	-	-	-	-
LBCS	rrrr	10,25	Test (C)=1	-	-	-	-	-	-	-
LBEQ	rrrr	10,27	Test (Z)=1	-	-	-	-	-	-	-
LBGE	rrrr	10,2C	Test (N) xor (V)=0	-	-	-	-	-	-	-
LBGT	rrrr	10,2E	Test (Z) or [(N) xor (V)]=0	-	-	-	-	-	-	-
LBHI	rrrr	10,22	Test (C) xor (Z)=0	-	-	-	-	-	-	-
LBHS	rrrr	10,24	Test (C)=0	-	-	-	-	-	-	-
LBLE	rrrr	10,2F	Test (Z) or[(N) xor (V)]=1	-	-	-	-	-	-	-
LBLO	rrrr	10,25	Test (C)=1	-	-	-	-	-	-	-
LBLS	rrrr	10,23	Test (C) or (Z)=1	-	-	-	-	-	-	-
LBLT	rrrr	10,2D	Test (N) xor (V)=1	-	-	-	-	-	-	-
LBMI	rrrr	10,2B	Test (N)=1	-	-	-	-	-	-	-

Mne-monic	Oper-and	Op-code	Function	Status F	H	I	N	Z	V	C
LBNE	rrrr	10,26	Test (Z)=0	-	-	-	-	-	-	-
LBPL	rrrr	10,2A	Test (N)=0	-	-	-	-	-	-	-
LBRA	rrrr	16	Tests always true	-	-	-	-	-	-	-
LBRN	rrrr	10,21	Tests always false	-	-	-	-	-	-	-
LBSR	rrrr	17	Subroutine call	-	-	-	-	-	-	-
LBVC	rrrr	10,28	Test (V)=0	-	-	-	-	-	-	-
LBVS	rrrr	10,29	Test (V)=1	-	-	-	-	-	-	-
LDA	ii	86	A=ii	-	-	-	T	T	0	-
	aa	96	A=M(aa)							
	xxop	A6	A=xxop							
	aaaa	B6	A=M(aaaa)							
LDB	ii	C6	B=ii	-	-	-	T	T	0	-
	aa	D6	B=M(aa)							
	xxop	E6	B=xxop							
	aaaa	F6	B=M(aaaa)							
LDD	iiii	CC	D=iiii	-	-	-	T	T	0	-
	aa	DC	D=M(aa,aa+1)							
	xxop	EC	D=xxop							
	aaaa	FC	D=M(aaaa,aaaa+1)							
LDS	iiii	10,CE	S=iiii	-	-	-	T	T	0	-
	aa	10,DE	S=M(aa,aa+1)							
	xxop	10,EE	S=xxop							
	aaaa	10,FE	S=M(aaaa,aaaa+1)							
LDU	iiii	CE	U=iiii	-	-	-	T	T	0	-
	aa	DE	U=M(aa,aa+1)							
	xxop	EE	U=xxop							
	aaaa	FE	U=M(aaaa,aaaa+1)							
LDX	iiii	8E	X=iiii	-	-	-	T	T	0	-
	aa	9E	X=M(aa,aa+1)							
	xxop	AE	X=xxop							
	aaaa	BE	X=M(aaaa,aaaa+1)							
LDY	iiii	10,8E	Y=iiii	-	-	-	T	T	0	-
	aa	10,9E	Y=M(aa,aa+1)							
	xxop	10,AE	Y=xxop							
	aaaa	10,BE	Y=M(aaaa,aaaa+1)							
LEAS	xxop	32	S=effad xxop	-	-	-	-	-	-	-
LEAU	xxop	33	U=effad xxop	-	-	-	-	-	-	-
LEAX	xxop	30	X=effad xxop	-	-	-	-	T	-	-
LEAY	xxop	31	Y=effad xxop	-	-	-	-	T	-	-
LSL	aa	08	M(aa)=M(aa) A< 1	-	?	-	T	T	?	T
	xxop	68	xxop=xxop A< 1							
	aaaa	78	M(aaaa)=M(aaaa) A< 1							
LSLA	--	48	A=(A) A< 1	-	?	-	T	T	?	T
LSLB	--	58	B=(B) A< 1	-	?	-	T	T	?	T
LSR	aa	04	M(aa)=M(aa) L> 1	-	-	-	0	T	-	T
	xxop	64	xxop=xxop L> 1							
	aaaa	74	M(aaaa)=M(aaaa) L> 1							

Mne-monic	Oper-and	Op-code	Function	Status F	H	I	N	Z	V	C
LSRA	--	44	A=(A) L> 1	-	-	-	0	T	-	T
LSRB	--	54	B=(B) L> 1	-	-	-	0	T	-	T
MUL	--	3D	D=(A)*(B)	-	-	-	-	T	-	T
NEG	aa	00	M(aa)=00-M(aa)	-	?	-	T	T	?	T
	xxop	60	xxop=00-xxop							
	aaaa	70	M(aaaa)=00-M(aaaa)							
NEGA	--	40	A=00-(A)	-	?	-	T	T	?	T
NEGB	--	50	B=00-(B)	-	?	-	T	T	?	T
NOP	--	12	P=(P)+0001	-	-	-	-	-	-	-
ORA	ii	8A	A=(A) or ii	-	-	-	T	T	0	-
	aa	9A	A=(A) or M(aa)							
	xxop	AA	A=(A) or xxop							
	aaaa	BA	A=(A) or M(aaaa)							
ORB	ii	CA	B=(B) or ii	-	-	-	T	T	0	-
	aa	DA	B=(B) or M(aa)							
	xxop	EA	B=(B) or xxop							
	aaaa	FA	B=(B) or M(aaaa)							
ORCC	ii	1A	CC=(CC) or ii	?	?	?	?	?	?	?
PSHS	rl	34	Push registers on M(S)	-	-	-	-	-	-	-
PSHU	rl	36	Push registers on M(U)	-	-	-	-	-	-	-
PULS	rl	35	Pull registers from M(S)	?	?	?	?	?	?	?
PULU	rl	37	Pull registers from M(U)	?	?	?	?	?	?	?
ROL	aa	09	M(aa)=M(aa) R< 1	-	-	-	T	T	?	T
	xxop	69	xxop=xxop R< 1							
	aaaa	79	M(aaaa)=M(aaaa) R< 1							
ROLA	--	49	A=(A) R< 1	-	-	-	T	T	?	T
ROLB	--	59	B=(B) R< 1	-	-	-	T	T	?	T
ROR	aa	06	M(aa)=M(aa) R> 1	-	-	-	T	T	?	T
	xxop	66	xxop=xxop R> 1							
	aaaa	76	M(aaaa)=M(aaaa) R> 1							
RORA	--	46	A=(A) R> 1	-	-	-	T	T	?	T
RORB	--	56	B=(B) R> 1	-	-	-	T	T	?	T
RTI	--	3B	Return from interrupt	?	?	?	?	?	?	?
RTS	--	39	Return from subroutine	-	-	-	-	-	-	-
SBCA	ii	82	A=(A)-ii-(C)	-	-	-	T	T	T	T
	aa	92	A=(A)-M(aa)-(C)							
	xxop	A2	A=(A)-xxop-(C)							
	aaaa	B2	A=(A)-M(aaaa)-(C)							
SBCB	ii	C2	B=(B)-ii-(C)	-	-	-	T	T	T	T
	aa	D2	B=(B)-M(aa)-(C)							
	xxop	E2	B=(B)-xxop-(C)							
	aaaa	F2	B=(B)-M(aaaa)-(C)							
SEX	--	1D	Sign extension of B into A	-	-	-	T	T	0	-
STA	aa	97	M(aa)=(A)	-	-	-	T	T	0	-
	xxop	A7	xxop=(A)							
	aaaa	B7	M(aaaa)=(A)							

Mne-monic	Oper-and	Op-code	Function	Status F H I N Z V C
STB	aa	D7	M(aa)=(B)	- - - T T 0 -
	xxop	E7	xxop=(B)	
	aaaa	F7	M(aaaa)=(B)	
STD	aa	DD	M(aa,aa+1)=(D)	- - - T T 0 -
	xxop	ED	xxop=(D)	
	aaaa	FD	M(aaaa,aaaa+1)=(D)	
STS	aa	10,DF	M(aa,aa+1)=(S)	- - - T T 0 -
	xxop	10,EF	xxop=(S)	
	aaaa	10,FF	M(aaaa,aaaa+1)=(S)	
STU	aa	DF	M(aa,aa+1)=(U)	- - - T T 0 -
	xxop	EF	xxop=(U)	
	aaaa	FF	M(aaaa,aaaa+1)=(U)	
STX	aa	9F	M(aa,aa+1)=(X)	- - - T T 0 -
	xxop	AF	xxop=(X)	
	aaaa	BF	M(aaaa,aaaa+1)=(X)	
STY	aa	10,9F	M(aa,aa+1)=(Y)	- - - T T 0 -
	xxop	10,AF	xxop=(Y)	
	aaaa	10,BF	M(aaaa,aaaa+1)=(Y)	
SUBA	ii	80	A=(A)-ii	- ? - T T T T
	aa	90	A=(A)-M(aa)	
	xxop	A0	A=(A)-xxop	
	aaaa	B0	A=(A)-M(aaaa)	
SUBB	ii	C0	B=(B)-ii	- ? - T T T T
	aa	D0	B=(B)-M(aa)	
	xxop	E0	B=(B)-xxop	
	aaaa	F0	B=(B)-M(aaaa)	
SUBD	iiii	83	D=(D)-iiii	- ? - T T T T
	aa	93	D=(D)-M(aa,aa+1)	
	xxop	A3	D=(D)-xxop	
	aaaa	B3	D=(D)-M(aaaa,aaaa+1)	
SWI	---	3F	Software interrupt	- - - - - - -
SWI2	---	10,3F	Software interrupt	- - - - - - -
SWI3	---	11,3F	Software interrupt	- - - - - - -
SYNC	---	13	Synchronize	- - - - - - -
TFR	r1	1F	Transfer register	? ? ? ? ? ? ?
TST	aa	0D	M(aa)-00	- - - T T 0 -
	xxop	6D	xxop-00	
	aaaa	7D	M(aaaa)-00	
TSTA	---	4D	(A)-00	- - - T T 0 -
TSTB	---	5D	(B)-00	- - - T T 0 -

B.2 M6809 Indexed Addressing Modes

The value of the post-byte (the first byte following the op-code) for instructions using the indexed addressing mode is determined by the format of the operand. Two formats exist: simple indexing and complex indexing. Simple indexing is used when the operand is of the form:

<exp>,R

where <exp> is an expression in the range -16 to +15 but not equal to zero, and R is one of the index registers "S", "U", "X", or "Y". All other indexed addressing modes use the complex indexing format. The two post-byte formats are described below:

Simple Indexing -- Post-byte

7	6	5	4	3	2	1	0
0	RR			OFFSET			

where RR=00 if X register
01 if Y register
10 if U register
11 if S register

OFFSET=5-bit 2's complement

Complex Indexing -- Post-Byte

7	6	5	4	3	2	1	0
1	RR		I	TTTT			

where RR=00 if X or PCR
01 if Y
10 if U
11 if S

I=0 if not indirect
1 if indirect

TTTT=0000 Single auto-increment (R+)
0001 Double auto-increment (R++)
0010 Single auto-decrement (-R)
0011 Double auto-decrement (--R)
0100 0 offset value or no offset
0101 B accumulator is offset (B,R)
0110 A accumulator is offset (A,R)
1000 8-bit offset
1001 16-bit offset
1011 D accumulator is offset (D,R)

1100 8-bit offset with PCR
1101 16-bit offset with PCR
1111 Extended indirect

B.3 M6800 Instructions and M6809 Equivalents

Not all M6800 instructions have exact equivalents recognized by the M6809 Assembler. Some translate into instructions that generate more bytes by the M6809 assembler. However, all op-code mnemonics recognized by the M6800 Macro Assembler are recognized by the M6809 Absolute Assembler and are translated into equivalent M6809 code where possible. Some translations are not equivalent but the same function is still performed. In addition, some "M6800-like" mnemonics are recognized by the M6809 Assembler and translated.

M6800/M6801 Mnemonic	Type of instruction	M6809 Equivalent
ABA	6800	PSHS B
		ADDA S+
ASLD	6801	ASLB
		ROLA
CBA	6800	PSHS B
		CMPA S+
CLC	6800	ANDCC #$FE
CLF	6800-like	ANDCC #$FB
CLI	6800	ANDCC #$EF
CLIF	6800-like	ANDCC #$AF
CLC	6800	ANDCC #$FD
CPX	6800	CMPX
CPY	6800-like	CMPY
DES	6800	LEAS -1,S
DEX	6800	LEAX -1,X
DEY	6800-like	LEAY -1,Y
INS	6800	LEAS 1,S
INX	6800	LEAX 1,X
INY	6800-like	LEAY 1,Y
LDAA; LDA A	6800	LDA
LDAB; LDA B	6800	LDB
LDAD	6801	LDD
LSLD	6801	ASLB
		ROLA
LSRD	6801	LSRA
		RORB
ORAA; ORA A	6800	ORA
ORAB; ORA B	6800	ORB
PSHA; PSH A	6800	PSHS A
PSHB; PSH B	6800	PSHS B
PSHX	6801	PSHS X
PULA; PUL A	6800	PULS A
PULB; PUL B	6800	PULS B
PULX	6801	PULS X
SBA	6800	PSHS B
		SUBA S+

M6800/M6801 Mnemonic	Type of instruction	M6809 Equivalent
SEC	6800	ORCC #$01
SEF	6800-like	ORCC #$40
SEI	6800	ORCC #$10
SEIF	6800-like	ORCC #$50
SEV	6800	ORCC #$02
STAA; STA A	6800	STA
STAB; STA B	6800	STB
STAD	6801	STD
TAB	6800	TFR A,B TSTA
TBA	6800	TFR B,A TSTA
TAP	6800	TFR A,CC
TPA	6800	TFR CC,A
TSX	6800	TFR S,X
TXS	6800	TFR X,S
WAI	6800	CWAI #$FF

APPENDIX

C. DIRECTIVE SUMMARY

A complete description of all directives appears in CHAPTER 3.

ASSEMBLY CONTROL

END	Program end
NAM	Assign program name
ORG	Origin program counter
SETDP	Set direct page pseudo register

SYMBOL DEFINITION

EQU	Assign permanent value
REG	Register list definition

DATA DEFINITION / STORAGE ALLOCATION

BSZ	Block storage of zeroes; single bytes
FCB	Form constant byte
FCC	Form constant character string
FDB	Form constant double byte
RMB	Reserve memory byte

PROGRAM CONTROL

SCALL	System function call
UCALL	User function call
SKIP1	Skip one byte
SKIP2	Skip two bytes

LISTING CONTROL

OPT CRE	Print cross reference table
OPT G	Print generated lines of FCB, FCC, and FDB directives
OPT NOG	Don't print generated lines of FCB, FCC, and FDB directives

OPT L	Print source listing from this point
OPT NOL	Inhibit printing of source listing from this point
OPT LLEN=n	Change line length
OPT M	Create object output to memory
OPT O	Create object output file
OPT NOO	Do not create object output file
OPT P=n	Change page length
OPT NOP	Inhibit pasing and printing of headings
OPT S	Print symbol symbol table
OPT W	Print warnings
OPT NOW	Don't print warnings
PAGE	Print subsequent statements on top of next page
SPC	Skip lines
TTL	Initialize heading for source listing

D. ASSEMBLER MESSAGES

A description of all error and warning messages follows. The format of an error is:

****ERROR XXX-- YYYYY

Where XXX is the error message number and YYYYY is the line number of the previously encountered error. If YYYYY = 00000, this indicates that there is no previous error. The format of the warning messages is similar.

D.1 Error Messages

173 Invalid use of direct mode indicator

The direct mode indicator, "<", was specified in the indirect addressing mode (e.g., LDA <[VAR]).

174 Invalid auto-increment/decrement format

Single auto increment or decrement was specified in the indirect mode (e.g., LDA [X+]) or more than two minus or plus signs detected (e.g., LDA ---X).

175 Invalid index register format

One of the accumulators "A", "B", or "D" was specified as the offset in the indexed mode, but was not followed by one of the index registers "S", "U", "X", or "Y" (e.g., LDA A,PCR).

176 Invalid expression for PSH/PUL

The immediate expression following one of the instructions PSHS, PULS, PSHU, or PULU contained symbols defined with other than the REG directive (section 3.11), contained an operator other than "!+", or contained no symbols following the "#" (e.g., PSHU #$FF; PSHS #REG1*REG2).

177 Incompatible register for PSH/PUL instruction

The register list for the PSHS/PULS instructions cannot contain the register "S", and the register list for the PSHU/PULU instructions cannot contain the register "U". The register list specified with the REG directive cannot contain both "S" and "U". In this case with the REG directive, the value assigned to the symbol will be the first "U" or "S" encountered (e.g., PSHS S).

178 Invalid register operand specification

Undefined register name encountered in register list, not exactly two register names in register list specification for TFR or EXG instructions; or no register list specified for PSH/PUL instructions. Valid register names are: A, B, CC, D, DP, PC, S, U, X, and Y. (e.g., TFR A,B,X; PULU Q)

179 Incompatible register pair

The register pair of an EXG instruction was not the same size (i.e., two 16-bit registers or two 8-bit registers), or the register pair specification of a TFR instruction indicated a transfer from an 8-bit register to a 16-bit register. The 8-bit registers are A, B, CC, and DP.The 16-bit registers are D, PC, S, U, X, and Y. (e.g., EXG X,A; TFR B,PC)

202 Label or op-code error

The label or op-code symbol does not begin with an alphabetic character or a period.

205 Label error

The statement label field is not terminated with a blank. This usually occurs if an invalid character is used in the label.

207 Undefined op-code

The symbol in the op-code field is not a valid op-code mnemonic, or directive.

208 Branch out of range

The operand resulted in an offset greater than 129 bytes forward or 126 bytes backward from the first byte of the branch instruction. This error also occurs when branching to an undefined symbol.

209 Illegal addressing mode

The specified addressing mode in the operand field is not valid with this instruction type.

210 Byte overflow -- operand too large

The operand's value exceeded 1 byte (8 bits). The most significant eight bits of the sixteen bit expression must be all zeroes or all ones for a one byte field.

211 Undefined symbol

The symbol never appears in a label field.

212 Directive operand error

A syntax error was detected in the operand field of a directive.

214 FCB directive syntax error

The structure of the FCB directive is syntactically incorrect.

215 FDB directive operand error

The structure of the FDB directive is syntactically incorrect.

216 Directive operand error

The directive's operand field is missing, terminated by an invalid terminator, or an expression in the operand field contains an invalid operator.

217 Option error

An option in the operand field of the OPT directive was undefined.

219 No END statement

The END directive was not found at the end of the last source file. The END directive is automatically supplied.

220 Phasing error

The value of the program counter during pass 1 and pass 2 for the same instruction is different.

221 Symbol table overflow

The symbol table has overflowed. This is a fatal error and terminates the assembly process during pass 1.

222 Reserved symbol used

One of the reserved symbols (A, B, D, CC, DP, X, Y, S, U, PC, and PCR) appeared in the label field or in the operand field of a statement. These symbols can only be

used in the operation field to modify the root mnemonic (A, B, or D) or in the operand field to specify indexed addressing (e.g., ,X).

223 The directive must or must not have a label

Depending on the directive used, the label field must be blank or must contain a valid symbol.

226 Illegal parenthesis

The parentheses in an expression do not balance.

227 Too many digits in a numeric constant

An overflow in the numeric evaluation of a constant was detected. Also used if a sequence number is missing on a line in a file that has sequence numbers.

229 Invalid starting execution address

The starting execution address specified as the expression on the END statement is not within the range of the XDOS-loadable object file. This can happen since RMB's at the beginning or end of the program are not included in the range of the program.

233 Symbol name too large

A symbol of more than 6 characters was encountered.

234 Multiple defined symbol

A reference was made to a multiple defined symbol.

235 Memory error

The OPT M option was used and the object code was going to be written into non-existant memory or into contiguous memory belonging to the assembler.

236 Program counter overflow

The program counter overflowed its maximum value of $FFFF.

237 Invalid terminator for sequence number

The character following a user-supplied sequence number was not a blank.

244 Illegal page or listing line length

A page or listing line length was not within the allowed range.

247 Invalid terminator for an operand

The character following the legal part of an operand is not a valid terminator (usually a carriage return or space). This error could occur if invalid indirect pairing: i.e. an operand has a "[" but no "]".

D.2 Warning messages

1 Long branch not required

A long branch instruction was used to branch to an address within the range -126 to +129. Although the long branch instruction could be changed to a short branch, it could result in other out-of-range short branches.

2 Extended addressing should be used

Direct addressing was forced by using the "<" indicator. However, the direct page pseudo register assigned by the SETDP directive (section 3.13) indicated that the extended mode should have been used.

3 Duplicate register specification

The same register name was used more than once in a register list. Register "D" specified with either register "A" or "B" gives this warning.

4 Possible SETDP expression error

The most significant byte of the expression in a SETDP directive was not zero. The direct page pseudo register is assigned the value of the least significant byte anyway.

6 Possible transfer error

The TFR instruction was used with a transfer from a 16-bit register to an 8-bit register. The result of such a transfer is to move the least significant byte of the 16-bit register to the 8-bit register.

APPENDIX

E. ASSEMBLER OUTPUT FORMAT

All numeric information printed on the source listing is in hexadecimal, unless otherwise noted.

E.1 Listing Format

The M6809 assembler will automatically print user-supplied sequence numbers in the left margin if they appear in the source file.

Seq #	COLUMN No Seq #	CONTENTS
1-5	-----	User-supplied sequence numbers (decimal)
7-11	1-5	Source line number; a five-digit decimal counter maintained by the assembler
14-17	8-11	Current program counter
19-20	13-14	First byte of machine operation code
21-22	15-16	Second byte of op-code (if any)
		For non-branch, non-indexed instructions :
24-25	18-19	First byte of operand
26-27	20-21	Second byte of operand (if any)
		For non-branch, indexed instructions :
24-25	18-19	Index post-byte
27-28	20-21	First byte of operand
29-30	23-24	Second byte of operand (if any)
		For branch instructions :
24-25	18-19	First byte of relative branch offset
26-27	20-21	Second byte of offset (if any)
29-32	23-26	Absolute address of destination
		For M6800 equivalent instructions :
24-25	18-19	Second byte of translated instruction
27-28	21-22	Third byte of instruction (if any)
29-30	23-24	Fourth byte of instruction (if any)
		For directives like BSZ, EQU, ORG, etc :
29-32	23-26	Value of expression

34-39	28-33	Label field
41-46	35-40	Operation field
47-54	41-48	Operand field; longer operands extend into the comment field
56-132	50-132	Comment field

E.2 Cross Reference Format

COLUMN	CONTENTS
4-7	Hexadecimal value of symbol
9-14	Symbol name
16-?	Assembler-maintained source line numbers of symbol reference. The asterisk appears after the line number of a symbol's definition. If the symbol was undefined, the

APPENDIX

F. USING THE M6809 ABSOLUTE ASSEMBLER

The following section describes how to invoke the M6809 Absolute Assembler from an XDOS diskette. After the assembler has been invoked, it will display the following message :

M6809 ABSOLUTE ASSEMBLER 01.00
COPYRIGHT BY MOTOROLA 1979

to indicate the current version and revision number.

The M6809 Absolute Assembler is invoked from the XDOS command line as are other XDOS commands. The format of the command line is :

ASM <name 1>[,<name 2>,...,<name n>] [;<options>]

where <name i> are the names of source files. Each source file name in the list of source files is in the standard XDOS file name format :

<filename> [.<suffix>] [:<logical unit number>]

The default values of "SA" and ."0" are used if suffix and logical unit number are not explicitly entered. Up to twenty file names can be accomodated by the assembler. If multiple source files are specified, only the last source file should contain the END directive. If an END directive is found in a file prior to the last one, the assembly will exclude any files after the END directive.

The <options> may be one or more of the options listed in the following table. All options except those that control the destination of the source listing, the destination of the object file, and the printing of error messages on the printer if no listing is desired, can be specified from within the source program with the OPT directive. Certain options are automatically used as a default condition. These conditions can be reversed or overridden by preceding the option letter by a minus sign (-). The following options are recognized by the assembler :

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
G	-G	Printing of generated code from FCB, FCC, and FDB directives
L	-L	Print source listing on line printer
L=#CN	-L	Print source listing on console
L=<name>,	-L	Print source listing into dikette file <name> (default suffix is "AL",default logical unit is zero)
M	-M	Print error messages only on line printer

N=ddd,	N=72	Set printed line length to "ddd" (decimal)
O	O	Create object file with name <name 1> and suffix "LO" (absolute XDOS-loadable) on same drive as <name 1> of command line
O=<name>,	O	Create object file with name <name>
P=dd,	P=58	Set number of printed lines per page to "dd" (decimal)
S	-S	Print symbol table
W	W	Print warning messages
X	-X	Print cross reference table

Certain options (L=, N=, O=, P=) require a terminating comma only if other options follow. Options are normally specified without any intervening blanks or separators. The options "L" and "M" are mutually exclusive.

Each symbol in the symbol table requires ten bytes of memory. However, if the cross reference option is specified, the symbol table requirements differ. In this case, an additional ten bytes are required by each symbol for every four references to that symbol.

Like other XDOS commands, the ASM command is sensitive to the CTL-P and CTL-W keys of the system console.

M6809 ABSOLUTE ASSEMBLER PROBLEM REPORT
****

Date :

Name :

Address

Phone number :

Host system :

M6809 Assembler version number :

Problem description :

Return this form to :

Motorola Semiconductors
Microsystems Software Engineering
16, chemin de la voie-creuse
1211 - GENEVE 20
Switzerland

Include a listing of the source program along with the source file on mini-diskette.
Also include any other information that may be appropriate to the solution of the problem.