# XDASM

Cross-Disassembler User's Guide

Version 2.1

Copyright 1990 - 1993 DataSync

**Data Sync Engineering**
40 Trinity Street
Newton, New Jersey 07860
(201) 383-1355 FAX (201) 383-9382

**Data Sync Engineering**
40 Trinity Street
Newton, New Jersey 07860
(201) 383-1355 FAX (201) 383-9382

# XDASM

(C) Copyright 1990-1993 by Data Sync Engineering.
All rights reserved.

No part of this publication may be reproduced without the prior written permission of Data Sync Engineering.

## SOFTWARE LICENSE AGREEMENT

On the following terms and conditions, Data Sync Engineering ("DSE") shall hereby grant and the purchaser ("Customer") shall hereby accept a nontransferable and nonexclusive license to use the below described software ("Licensed Software") and any manuals provided therewith:

### Licensed Software: XDASM Table Based Cross-disassembler

Customer shall be authorized to use the Licensed Software on a single computer system. A separate license shall be required for each system on which the Licensed Software shall be used. Customer shall not allow any third party to use or have access to the Licensed Software or any part or copies thereof without the prior written consent of DSE. This Agreement, the license hereunder and the Licensed Software may not be assigned, sublicensed or otherwise transferred, in whole or in part, without the prior written consent of DSE.

## DISCLAIMER

XDASM is sold as a software tool only. The Customer assumes the entire risk of its operation. DSE will in no event be held liable for direct, indirect or incidental damages including, but not limited to, any interruption of service, loss of business or anticipatory profits, or any other consequential damages.

**PC-DOS is a trademark of International Business Machines Corp.**
**MS-DOS is a trademark of Microsoft Corporation.**

**Data Sync Engineering**
P.O. Box 146
East Stroudsburg, PA 18301

TEL (717) 421-1977
FAX (717) 421-9095

## XDASM V2.1C ENHANCEMENTS

* Supports variable word-length formats in byte multiples up to 16 bytes.
* Reads Intel 8/16 or Motorola S 8/16 records.
* Includes CPU table compiler, XTC.EXE.
* Improved error checking of CPU table.
* Accepts full 8 character symbolic names in XRn files.
* Processor families supported;

**1802-6 3870 4004 6301/6303 64180/Z180 6502/65C02 65816 6800-6808 6805 6809 68HC11 8048 8051 8080/8085 8086 8096 89700 COP400 COP800 NEC78C10 NEC78C310 PIC16C5x TMS7000 TMS320C1x TMS320C2x Z8 Z80**

* New CPU table commands;

```
WORDSIZE
DWL      nbytes            -Sets word size, in number of bytes, up to sixteen (16).

IFGOTO
DWL      match,address     - Match to first word causes jump to second, zero
                             match ends command and contains "no-match jump".

IFZERO
DWL      address           - If pseudo-acc = zero, jump to specified addresss,
                             otherwise fall thru.

LOGICAL  ADD
DWL      16-bit value      - Adds the unsigned value to the pseudo-acc.

LOGICAL  SUB
DWL      16-bit value      - Subtracts the unsigned value from the pseudo-acc.
```

A registration form (**REGISTER.TXT**) has been supplied on your program disk. Please print, fill out and mail this form back to us.

Additional Tables, Updates, Tips and Techniques
On-line BBS (1200/2400 baud, No parity, 8 data bits, 1 stop bit)
(717) 424-6754
Your password: **"MORECPU"**

# XDASM CROSS-DISASSEMBLER User's Guide

## Table of Contents


**GENERAL**

1.0 General Description ........................... 1
1.1 System Requirements ........................... 1
1.2 Installation ........................... 1

**USING THE XDASM DISASSEMBLER**

2.0 Input File Types ........................... 2
2.1 Starting XDASM ........................... 3
2.2 XDASM Options ........................... 4
2.3 Tag Files ........................... 5
2.4 Operand Text Files ........................... 6
2.5 Conventional Assembler Format ........................... 7
2.6 Output File Description ........................... 8
2.7 Unresolved References ........................... 9
2.8 Cross Reference Lists ........................... 9
2.9 Altering The Disassembler Format ........................... 10
2.10 A Typical Disassembly Procedure ........................... 11

**WRITING CPU TABLES**

3.0 Operation Overview ........................... 15
3.1 Starting a CPU table ........................... 16
3.2 The Vector List Section ........................... 17
3.3 The Start-Up Initialization Section ........................... 18
3.4 Start-Up Initialization Commands ........................... 19
3.5 The Opcode Processing Sections ........................... 20
3.6 Opcode Processing Commands ........................... 21
3.7 Function Commands ........................... 30
3.8 Compiling The CPU Table ........................... 33

**REFERENCES**

4.0 Error Messages ........................... 35
4.1 Intel Hex Format ........................... 36
4.2 Motorola Hex Format ........................... 37
4.3 Processor File Descriptions ........................... 38
4.4 Manufacturer Directory ........................... 39
4.5 Simple EPROM reader circuit ........................... 41

## 1.0 GENERAL DESCRIPTION

XDASM is a table based cross-disassembler tool. "Cross" meaning that while running on one type of a computer, it disassembles code for another. In many ways, XDASM is similar to a debugger with the exception that it produces a source program code that is suitable for an assembler.

Features, such as, Instruction line HEXDUMP, Automatic Label assignment, and various Cross-reference lists greatly aid in the complicated task of reconstructing or debugging a program.

A separate control file, called a TAG file, allows the user to mark (or tag) specific program areas for a particular type of disassembly.

Another file, called a Format file, configures XDASM's Source Output and can be modified to match the requirements of your favorite assembler.

Some CPU tables utilize external text files which are used to substitute specific addresses with symbolic names. These text files are user modifiable and enable XDASM to be "tailored" or "adapted" to other members of that processor's family.

## 1.1 SYSTEM REQUIREMENTS

XDASM operates under MS-DOS 2.0 or later and uses about 22K of memory. An additional 390K of memory is allocated for cross-references and table storage. A typical system requirement would be about 512K bytes of memory.

XDASM produces a disassembled program with cross-references in three passes. Pass #1 reads the input file and tabulates labels and references. Pass #2 rereads the input file and begins writing each line of program code. Pass #3, which is optionally the reference generation pass, writes out the sorted cross-reference lists.

An 8K Binary file, typically 23K in HEX format, may produce a disassembled source file of approximately 173K. While it is possible to run XDASM on a floppy disk, a hard disk would be recommended for speed considerations.

## 1.2 INSTALLATION

To install, simply copy all the files from the distribution disk to your fixed disk. You may want to create a directory for **XDASM**. We recommend a directory named **XD**. You may also wish to delete unused processor tables from your FIXED DISK to save space.

## 2.0 INPUT FILE TYPES

XDASM can disassemble Intel Hex, Motorola Hex or Binary files.

### HEX

Hex files are translated versions of binary files. These translations are in a format of all ASCII characters. The entire file is broken down into groups of one-byte digits (which are ASCII characters). Each such group is given a separate load address (the place where it is to be loaded in memory) and a count (Number of characters in the group). In addition, there's a checksum in each group for detecting errors.

An Intel Hex format would look like:

```
:0300000020100FA
:10010000900118120108 80FEE493A3B4000122308C
```

The Motorola Hex format would look like:

### BINARY

Just as the eighth bit is used as a Parity check in communications, it is also used to enable graphic or special character symbols in video display adapters. Therefore, Binary files cannot be directly displayed. Programs that organize and display the contents of binary files are usually called DUMP or HEXDUMP programs.

The following is a view of a binary file using HEXDUMP:

```
0000  02 01 00 FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
       :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :
0100  90 01 18 12 01 08 80 FE E4 93 A3 B4 00 01 22 30 .............."0
0110  99 FD C2 99 F5 99 21 08 45 58 41 4D 50 4C 45 00 ......!.EXAMPLE.
```

## 2.1 STARTING XDASM

Starting XDASM requires specifying the input file, the CPU type and any of the disassembly options. The command form is:

**XDASM filename,type [/options]**

The FILENAME is the name of the input file that is to be disassembled. If the input file resides on a different drive or path, it must be specified as part of the filename.

The cpu TYPE, preceded by a comma, usually resides on the same drive and path as XDASM. There are two control files for each cpu type; the Disassembly Table, identified as "type.CPU", and the Disassembly Format, identified as "type.FMT. The Disassembly Format file is used to configure XDASM's output for the Assembler being used.

The disassembly OPTIONS are single character commands that are used to select various listing formats and disassembler functions. The option characters may be grouped together and must be preceded by the forward slash (/).

When started, XDASM searches for a file named "type.EQU". This is a user generated text file which usually contains Equate definitions plus any additional Directives for the assembler. If found, its contents are transferred, unmodified, to the start of the source output file.

For example, the user wants to disassemble an 8051 type program, called TEST.HEX, and would like to include all listing features...

The command line would then be:

**XDASM TEST.HEX,8051 /RC**

When XDASM is started, the following messages would appear:

```
XDASM - Table Based Cross-Disassembler   Version 2.1x   PC/MS-DOS
Copyright (C) 1990-1992 Data Sync Engineering  All rights reserved.

No Tag File
Starting PASS Number 1  -  Processing: 0110
Starting PASS Number 2  -  Processing: 0110
Starting PASS Number 3  -  Xref lists: 0118

     0 Unassigned Opcodes
     2 Unresolved References

Disassembly Complete --  Source File Created.
```

A new file called TEST.SRC has now been created.

## 2.2 XDASM OPTIONS

XDASM contains powerful disassembly options that provide a more complete detail on the program that is being disassembled. Option characters are preceded by a slash (/) and several characters may be grouped together (ie. XDASM TEST.HEX,8051 /LCTR).

| Option letter | Result |
|---|---|
| **B** | Tells XDASM to input a Binary coded file. |
| **C** | Append Comment Field Hexdump to each instruction line. |
| **L** | Converts output assembly code to lower case characters. |
| **M** | Mask 7-bits for ASCII Hexdump within comment fields. |
| **R** | Append Cross-reference lists to end of source file. |
| **T** | Include Tag File disassembly control. |
| **X** | Write Cross-Reference lists only, no assembly code. |

### COMMENT FIELD HEXDUMP

The Comment Field Hexdump is appended to each line containing a disassembled instruction. It shows the Program Counters value, the HEX bytes utilized by the current instruction and the displayable ASCII character equivalent of the Hex bytes. Non-displayable characters, such as control characters, are shown as periods ".".

### CROSS-REFERENCE LISTS

The Cross-reference lists produced by XDASM, map out all Program Code and Memory usages. The listing shows the Label or Memory address followed by all locations that refer to that address. The referencing locations are identified by the Program Counter value as shown in the comment field hexdump.

## 2.3 TAG FILES

The Tag file is used to add a more direct control over the disassembly process. It instructs XDASM as to what type of disassembly to switch to at what specified address. This file can be created with any ASCII text editor and is identified with the same filename, as the input file, but has the file extension of ".TAG".

Entries into the tag file consist of a four character hexadecimal start address followed by the "=" character then the command character. Only one command character is permitted on each line for each designated address. The carat (^) character is used to terminate the Tag list. Address entries are not required to be in sequential order and later address duplications will override previous.

**Note**: The tag file is used only when the "T" option is specified in the command line. The summary of commands are:

| Command | Result |
|---|---|
| **B** | Define BYTE disassembly (DB 000H). |
| **G** | GENERATE Label Equate for specified address. |
| **H** | HEX address offset (positive). |
| **I** | INSTRUCTION disassembly. |
| **S** | SKIP disassembly (ignores marked bytes). |
| **T** | TEXT disassembly (DB 'text'). |
| **>** | Define WORD, first byte = LSB of word. |
| **<** | Define WORD, first byte = MSB of word. |

## 2.4 OPERAND TEXT FILES

Some processor types, such as the 8051 and 6805, contain Special Function Registers mapped into the Page 0 address range (00H to FFH). They usually include parallel ports, serial ports, counter/timers, A/D converters, PWM's and so on. In this case, XDASM uses external text files to substitute the Hex addresses (000H or $00) with the symbolic name for that register. This provides an output listing more easily readable than if just Hex addresses were shown. These text files are identified with the same name as the processor type, but have the file extensions of ".XR1", ".XR2" and ".XR3".

Not all text files may be used for a particular processor type. For example, the 8051 uses two, one for its BYTE operands (8051.XR1) and the other for its BIT operands (8051.XR2), whereas, the 6805 only uses one for its Page 0 registers (6805.XR2).

These text files are easily modified using an ASCII text editor and allow the symbolic names to be matched to a member of that processor's family.

Entries within the text files begin with address "00H" and end with address "FFH" making a total of 256 lines. Each line contains ten characters, which includes the carriage-return and line-feed. A symbolic name, which can have up to eight characters, must be space filled using the equal "=" character. Any character(s) after the equal character are not inserted into the assembly file.

Whenever the text files are being edited, it is important that each line contains eight characters plus the carriage-return and line-feed. It is also recommended that upper-case characters be used since XDASM does have a lower-case conversion option. Unassigned entries should contain its Hex address text, such as 003H or $03. The following are some examples of text file entries:

| | 6805.XR2 | 8051.XR1 | 8051.XR2 |
|---|---|---|---|
| 00 - | PORTA=== | 000H==== | 000H==== |
| 01 - | PORTB=== | 001H==== | 001H==== |
| 02 - | PORTC=== | 002H==== | 002H==== |
| 03 - | $03===== | 003H==== | 003H==== |
| 04 - | DDRA==== | 004H==== | 004H==== |
| 05 - | DDRB==== | 005H==== | 005H==== |
| 06 - | DDRC==== | 006H==== | 006H==== |
| : | : | : | : |
| : | : | : | : |
| 80 - | $80===== | P0====== | P0.0==== |
| 81 - | $81===== | SP====== | P0.1==== |
| 82 - | $82===== | DPL===== | P0.2==== |
| 83 - | $83===== | DPH===== | P0.3==== |

## 2.5 CONVENTIONAL ASSEMBLER FORMAT

All program code is written out using the conventional assembler format. This format divides each line in the program into four fields: label, op code, operand and comments.

The LABEL field is used to assign a symbolic name or label to the location of an instruction, so that it can be referenced by other instructions in the program. For example, the instruction JMP L0100 will cause the program counter to be unconditionally loaded with the memory address 0100H which was assigned to the label L0100. The instruction at label L0100 will be the next instruction to be executed after the JUMP (JMP) instruction is executed. Most instructions will not be labeled, however, if an instruction is referenced, the label will begin in the leftmost column of the line and will begin with the character "L". The body of the label will contain the target address value. The label is ended with colon ":" and followed by a tab. If no label was assigned, the label field is skipped using a tab character.

The OP CODE field is mandatory for every line in the program that contains an instruction. The op code begins in field 2 and is separated from the label field by a tab character.

The OPERAND field is used to specify data or an address for instructions that require an operand. The operand begins in field 3 and is separated from the op-code by a tab character. XDASM provides three types of operand forms:

| | |
|---|---|
| **000H - 0FFH or $00 - $FF** | **Hexadecimal notation.** |
| **L0000 - LFFFF** | **Label names.** |
| **ACC, DPL, SBUF** | **Symbolic names.** |

The COMMENT field is used to add an explanatory note to a statement The contents of the comment field are ignored by the assembler. The text of the comment field will be preceded by the semicolon character ";". Comments may be used alone without being appended to a line that contains an instruction. XDASM uses the comment field in one of three forms:

1. To identify the address and contents of the current instruction line.
2. To deblock program segments for easier interpretation.
3. To append the various cross-reference lists.

## 2.6 OUTPUT FILE DESCRIPTION

The initial ORG assembly directive is inserted into the output file and always starts with address 0000, unless an ADDRESS offset was specified in the Tag file. Further ORG directives are automatically inserted whenever the Hex Load Address becomes non-sequential.

```
            ORG     00000H
;
            LJMP    L0100               ;0000  02 01 00      ...
;
            ORG     00100H

:03 0000 00 020100 FA
:10 0100 00 90011812010880FEE493A3B400012230 8C
    ----
      |
      +--> Load Address for current line
```

When the "C" option is specified in the command line, XDASM appends a Comment Field Hexdump to each line that contains a disassembled instruction. The description of that field is shown below:

```
L0100:  MOV     DPTR,#00118H        ;0100  90 01 18      ...
        LCALL   L0108               ;0103  12 01 08      ...
;
L0106:  SJMP    L0106               ;0106  80 FE         ..
;
L0108:  CLR     A                   ;0108  E4            .
        MOVC    A,@A+DPTR           ;0109  93            .
        INC     DPTR                ;010A  A3            .
        CJNE    A,#000H,L010F       ;010B  B4 00 01      ...
        RET                         ;010E  22            "
;
L010F:  JNB     TI,L010F            ;010F  30 99 FD      0..
        CLR     TI                  ;0112  C2 99         ..
        MOV     SBUF,A              ;0114  F5 99         ..
        AJMP    L0108               ;0116  21 08         !.
;
        ORL     A,058H              ;0118  45 58         EX
        AJMP    L024D               ;011A  41 4D         AM
;
        JNC     L016A               ;011C  50 4C         PL
        ORL     A,000H              ;011E  45 00         E.

Instruction address <--------------+
        HEX bytes utilized <--------------------+
                ASCII equivalent of HEX bytes <-----------+
```

## 2.7 UNRESOLVED REFERENCES

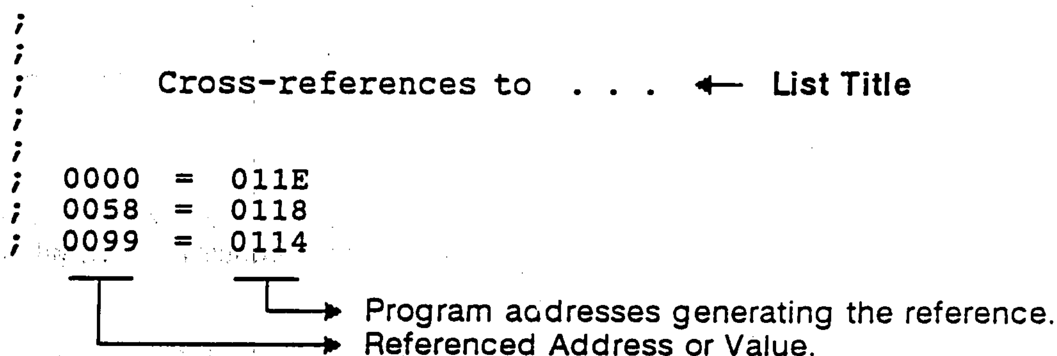
If a memory location reference was not found as an instruction address then XDASM automatically EQUATES the value to that label and shows it in the Unresolved Reference list.

Unresolved References are generally caused by the disassembly of ASCII text characters, program tables or by references made to external I/O or memory. These Undefined References can be eliminated by using the Tag file to mark off these text or table sections or to manually generate an equate for the external address.

```
;
;        Unresolved Address Reference list
;
;
L016A:          EQU         0016AH
L024D:          EQU         0024DH
```

## 2.8 CROSS REFERENCE LISTS

When the "R" or "X" option is specified in the command line, XDASM appends up to four Cross-reference lists to the end of the source file. The first list are references to Labels (or memory addresses), the other three lists will vary according to the CPU type being disassembled. They may represent 8 or 16 bit constants, memory, I/O addresses or some other special registers. The LIST TITLE identifies the type of cross-reference.



## 2.9 ALTERING THE DISASSEMBLER FORMAT

Many types of assemblers require different variations of the Source file syntax. For example, one asssembler may use the directive "DB" to define a byte and the other may use "DFB". To overcome these variations, XDASM utilizes a Format file to configure the Directives and Delimiters used for its output source file. The format file also allows up to two lines of general text or assembly statements, to be inserted into the source file. Each line in the format file configures a different parameter or directive and may be altered using an ASCII text editor. Format files have the same name as the cpu type but have the extension of (.FMT). This enables XDASM to have different formats for each microprocessor. Listed below are the line descriptions and their default settings:

| Line # | Max Size | Description | Current Default |
|---|---|---|---|
| 1 | 128 | General Assembler Directive Line #1 | <Cross-16 CPU> |
| 2 | 128 | General Assembler Directive Line #2 | <Cross-16 HOF> |
| 3 | 8 | Define Byte Directive for Hex bytes | DFB |
| 4 | 8 | Define Byte Directive for ASCII Text | DFB |
| 5 | 8 | Define Word Directive (first byte = MSB) | DWH |
| 6 | 8 | Define Word Directive (first byte = LSB) | DWL |
| 7 | 8 | Origin Directive | ORG |
| 8 | 8 | Equate Directive | EQU |
| 9 | 8 | End Directive | END |
| 10 | 3 | Output Source File Extension | SRC |
| 11 | 1 | Start Of Label character | L |
| 12 | 1 | End Of Label character | : |
| 13 | 1 | End Of Equate Label character | : |
| 14 | 1 | Comment Field Delimiter character | ; |
| 15 | 1 | Text Delimiter characters | " |
| 16 | 1 | Hexadecimal format; 0 = 0xxH, 1 = $xx | 0 |
| 17 | *** | The Carat (^) signifies the end of the format list | ^ |

Each line entry is terminated with a carriage-return character. If the carriage-return were the only character on a line, then that directive or delimiter will not be written. For example, some assemblers do not require colon (:) characters at the end of label names, so line # 12 and possibly # 13 will contain only the carriage-return character.

**Note**: XDASM's format lines 1 and 2 initially contain the default statements used by CROSS-16/32, the companion Cross-assembler. They may be replaced with any other statement or assembler directive. XDASM inserts these lines at the start of the assembly source file.

## 2.10 A TYPICAL DISASSEMBLY PROCEDURE

**STEP 1** - First we will disassemble TEST.HEX using the Hexdump and Cross-reference options.

**XDASM TEST.HEX,8051 /RC**

```
XDASM - Table Based Cross-Disassembler   Version 2.1x   PC/MS-DOS
Copyright (C) 1990-1992 Data Sync Engineering  All rights reserved.

No Tag File
Starting PASS Number 1  -  Processing: 0110
Starting PASS Number 2  -  Processing: 0110
Starting PASS Number 3  -  Xref lists: 0118

    0 Unassigned Opcodes
    2 Unresolved References

Disassembly Complete -- Source File Created.
```

Now examine the disassembled listing (TEST.SRC).

```
;=================================================================;
; Disassembled Using XDASM -- (C)1990-92 DataSync Engineering ;
;=================================================================;
;
;       Unresolved Address Reference list
;
;
L016A:          EQU         0016AH
L024D:          EQU         0024DH
;
;
        ORG     00000H
;
        LJMP    L0100               ;0000  02 01 00        ...
;
;
        ORG     00100H
;
;
L0100:  MOV     DPTR,#00118H        ;0100  90 01 18        ...
        LCALL   L0108               ;0103  12 01 08        ...
;
L0106:  SJMP    L0106               ;0106  80 FE           ..
;
```

```
L0108:  CLR     A                       ;0108  E4          .
        MOVC    A,@A+DPTR               ;0109  93          .
        INC     DPTR                    ;010A  A3          .
        CJNE    A,#000H,L010F           ;010B  B4 00 01    ...
        RET                             ;010E  22          "
;
L010F:  JNB     TI,L010F                ;010F  30 99 FD    0..
        CLR     TI                      ;0112  C2 99       ..
        MOV     SBUF,A                  ;0114  F5 99       ..
        AJMP    L0108                   ;0116  21 08       !.
;
L0118:  ORL     A,058H                  ;0118  45 58       EX
        AJMP    L024D                   ;011A  41 4D       AM
;
        JNC     L016A                   ;011C  50 4C       PL
        ORL     A,000H                  ;011E  45 00       E.
;
;       Cross-references to LABELS
;
; L0100  =  0000
; L0106  =  0106
; L0108  =  0103  0116
; L010F  =  010B  010F
; L016A  =  011C
; L024D  =  011A
;
;       Cross-references to Data Memory locations
;
;  0000  =  011E
;  0058  =  0118
;  0099  =  0114
;
;       Cross-references to BIT addressable locations
;
;  0099  =  010F  0112
;
;       Immediate  Byte references
;
;  0000  =  010B
;  0118  =  0100

        END
```

As you can see, XDASM automatically inserted Labels for memory location references and Directive statements for Assembler control. A code segmentation method is also performed by inserting blank comment lines after certain instructions, such as RET and JMP, and before a line that contains a Label.

As we examine the listing, we notice that the area between addresses 0118 to 011E contain ASCII text characters. It is also the same area that is generating the Unresolved Address References.

**STEP 2** - Using a text editor, we will create a Tag file called TEST.TAG which will contain the following lines:

```
0118=T
011F=B
^
```

Our Tag commands above simply specify, that, starting at address 0118 switch to "Define Text" disassembly. Then at address 011F switch to "Define Byte" disassembly. The carat (^) character signifies the end of the Tag file command list.

**Note**: Text can be segmented by inserting additional Define Text commands.

**STEP 3** - The final step is to again disassemble the TEST.HEX program using the "T" (Tag File) option.

For clarity, we will also use the "C" (Hexdump) and "L" (Lower-case) options.

**XDASM TEST.HEX,8051 /TCL**

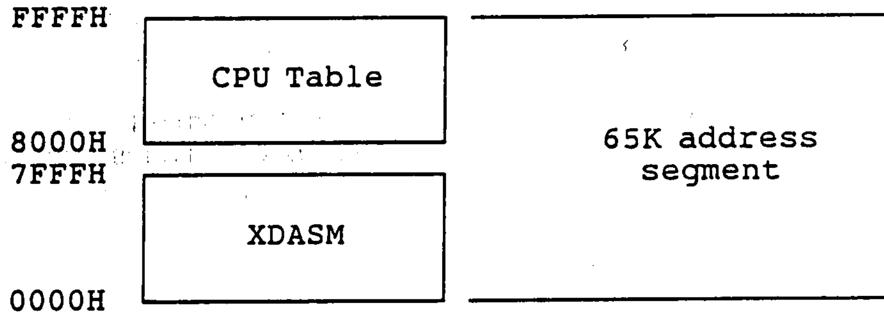
```
XDASM - Table Based Cross-Disassembler   Version 2.1x   PC/MS-DOS
Copyright (C) 1990-1992 Data Sync Engineering  All rights reserved.

Tag File Processed
Starting PASS Number 1  -  Processing: 0110
Starting PASS Number 2  -  Processing: 0110

     0 Unassigned Opcodes
     0 Unresolved References

Disassembly Complete --. Source File Created.
```

The final TEST.SRC result should now look like:

```
;==================================================================;
; Disassembled Using XDASM  -  (C)1990-92 DataSync Engineering ;
;==================================================================;
;
;
;
;       Unresolved Address Reference list
;
;
;
;
        org     00000h
;
        ljmp    L0100                   ;0000  02 01 00         ...
;
;
        org     00100h
;
;
L0100:  mov     dptr,#00118h            ;0100  90 01 18         ...
        lcall   L0108                   ;0103  12 01 08         ...
;
L0106:  sjmp    L0106                   ;0106  80 FE            ..
;
L0108:  clr     a                       ;0108  E4               .
        movc    a,@a+dptr               ;0109  93               .
        inc     dptr                    ;010A  A3               .
        cjne    a,#000h,L010F           ;010B  B4 00 01         ...
        ret                             ;010E  22               "
;
;
L010F:  jnb     ti,L010F                ;010F  30 99 FD         0..
        clr     ti                      ;0112  C2 99            ..
        mov     sbuf,a                  ;0114  F5 99            ..
        ajmp    L0108                   ;0116  21 08            !.
;
;
L0118:  dfb     "EXAMPLE"
        dfb     000h                    ;011F  00               .
;
;
        end
```

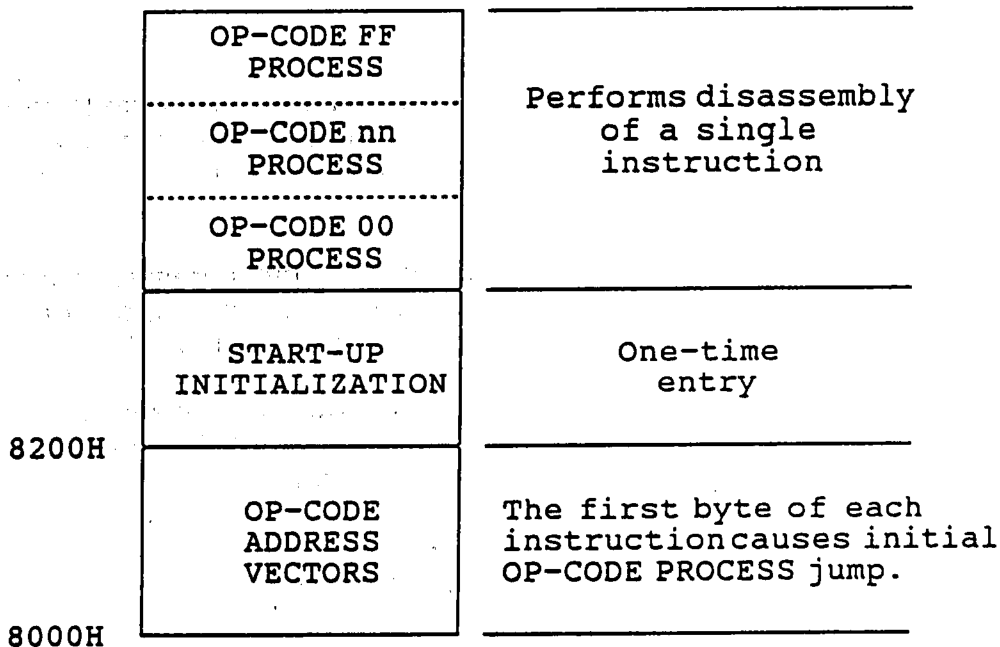
## 3.0 OPERATION OVERVIEW

The XDASM cross-disassembler basically consists of two parts, the XDASM interpreter and the CPU table application. These two programs reside within a 65K segment of memory and interact with each other using high-level commands. When started, XDASM loads the upper address section of memory with the CPU table specified in the command line.




XDASM manages all sorting, storing and disk file operations. It acts as a command interpreter when control is transferred to the CPU table program.

The CPU table contains three sections; the initial vector list, the start-up initialization and the opcode processing blocks. Each opcode processing block performs the disassembly of a single instruction line.



The very first section in the CPU table contains the initial jump vectors. These vectors are address pointers to the opcode processing blocks within the table. The first opcode byte of every instruction generates the index into the vector list. XDASM then jumps to that address specified. If an opcode is made up of more than one byte, then additional vector lists may be used at each target address jump. The 6809 microprocessor is a prime example for using multiple vector lists.

The Start-up Initialization section is located at address 8200H. Its primary function is for opening and naming Cross-reference lists, loading the appropriate text overlay files and setting disassembly parameters. XDASM executes the Start-up Initialization routine before starting the disassembly process.

The final section contains the individual opcode processing blocks. They are responsible for reading all required bytes that make up the instruction, writing the mnemonic text and logging references.

## 3.1 STARTING A CPU TABLE

Writing a CPU table follows a similar convention as writing assembly language code. That is, Labels, Instructions and optional Comments are used on each line. However, instead of processor related instructions, XDASM uses high-level commands which are then compiled to produce a binary image. An XDASM Table Compiler, XTC.EXE, allows this high-level method of programming to be utilized.

Before starting a CPU table, it is recommended that an outline of the target processor be developed. This outline should contain all the opcode mnemonics and at least the first opcode byte of each instruction. This format was used in developing the tables supplied on your XDASM program disk. The outline files are identified with the file extension of ".MNE".

Examine your target processor's instruction set. In most cases, it may be an enhanced variation of previously related processor. The new table can then be developed with minor editing or additions to an already existing table.

The actual commented source code for the 8051 table has been provided as a template example. You can regenerate the source codes for the other tables by simply disassembling them using the following command:

**XDASM type.CPU,TABLE /B**

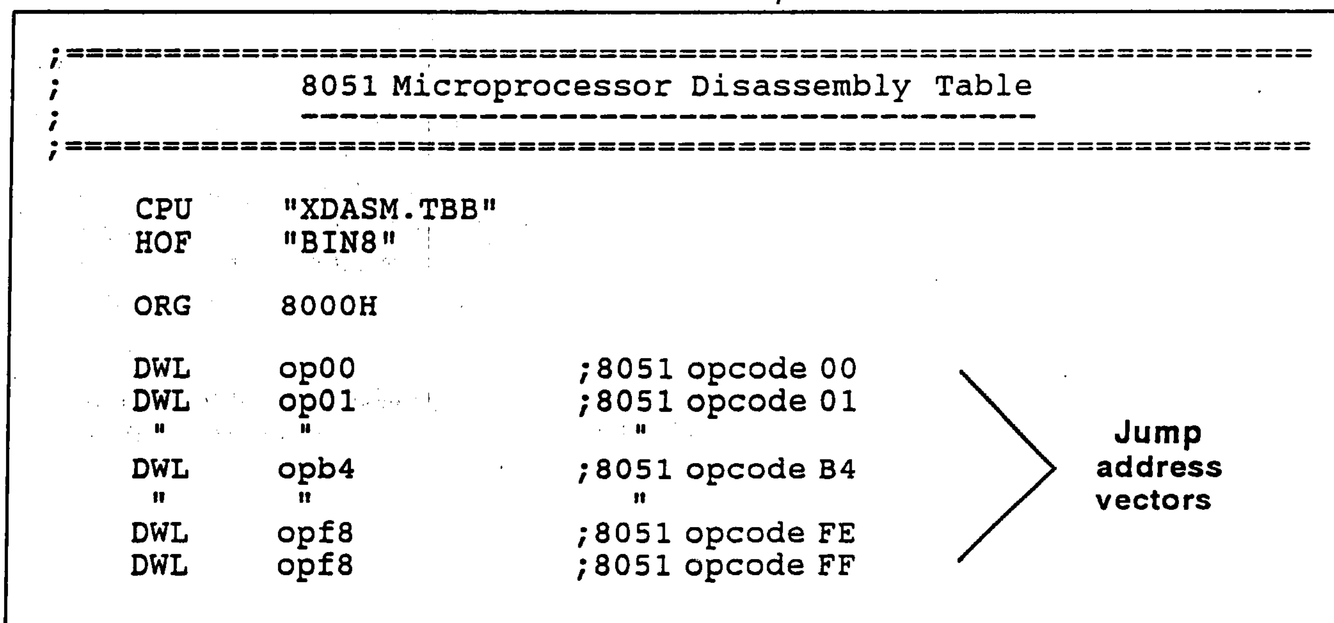
*A note about table disassembly*

Theoretically, when a table is disassembled, it should not produce any Unresolved References. But, since binary mode disassembly is being used. some left over bytes beyond the end of the table may be disassembled and may create some unresolved references. By examining the created source, you can locate this unused code at the end of the listing. It's usually the opcode processing blocks that do not contain a label.

Also, keep in mind that the disassembled tables will not have descriptive comments. You may need to refer to the 8051 source table for a better understanding. Our BBS system contains all of the CPU table sources for downloading.

## 3.2 THE VECTOR LIST SECTION

*8051 CPU example*



```
;=================================================================
;               8051 Microprocessor Disassembly Table
;               --------------------------------------
;
;=================================================================

        CPU     "XDASM.TBB"
        HOF     "BIN8"

        ORG     8000H

        DWL     op00            ;8051 opcode 00
        DWL     op01            ;8051 opcode 01
         "        "                    "
        DWL     opb4            ;8051 opcode B4
         "        "                    "
        DWL     opf8            ;8051 opcode FE
        DWL     opf8            ;8051 opcode FF
```



**Note**: The vector list is shown condensed to conserve space, but normally, it would contain 256 address entries.

## 3.3 THE START-UP INITIALIZATION SECTION

*8051 CPU example*

```
;               Initialization Subroutine Entry

        org     8200h


        title1
        dfb     "Cross-references to Data Memory",0
        title2
        dfb     "Cross-references to Bit Addressable Memory",0
        title3
        dfb     "Cross-references to Immediate values",0
        load    ov1
        dfb     "8051.XR1",0            ;Byte Operand text overlay
        load    ov2
        dfb     "8051.XR2",0            ;Bit Operand text overlay
        return                          ;back to XDASM
```

XDASM executes the initialization routine before any disassembly can begin. In the example above, three additional cross-reference lists are defined and two text substitution overlay files are loaded. XDASM will always provide the cross-references to LABEL addresses. Cross-reference list entries are performed using the **LOG XREFn** commands within the opcode processing routines. Each reference entry stores the contents of the pseudo accumulator and the memory address of the current instruction.

The text substitution buffer uses the lower 8-bits of the pseudo accumulator as an index to a text string. The commands, **WRITE OVn**, cause the indexed string to be written. The ASCII text overlay files, identified as type.XRn, can be easily modified using a text editor.

## 3.4 START-UP INITIALIZATION COMMANDS

```
ORIGIN
DWL        offset
```

Adds the 16-bit offset to the current instruction address from the source file. All disassembly will be relative to the new offset address.

```
TITLEn                                  (n = 1, 2 or 3)
DFB        "text",0
```

Opens a cross-reference buffer and inserts the Title message "text" *(see section 2.8)*. Text messages are null terminated.

XDASM will always provide cross-references to label addresses and will maintain up to three more additional lists. An entry into a list is performed using the **LOG XREFn** command *(see usage in section 3.5)*. The lists are kept in numerical order and are written to the output source file when the **R** or **X** option is specified in the command line.

```
LOAD       OVn                          (n = 1, 2 or 3)
DFB        "filename",0
```

Reads the specified text file into the symbolic overlay buffer.

XDASM contains three overlay buffers which are used to replace an operand hex notation with a symbolic name. The "filename" is usually the CPU type such as "8051.XR1". The command, **WRITE OVn**, causes the text to be written into the output source file.

*Read section 2.4 for text file details.*

## 3.5 THE OPCODE PROCESSING SECTIONS

*8051 CPU example*

```
; CJNE A,#imm8,rel8

opb4:
        read    byte            ;byte2 = immediate byte constant
        read    byte            ;byte3 = rel address
        write   mnemonic        ;send op-code text
        dfb     "CJNE",tab,"A,#",0
        get     byte2
        log     xref3           ;log 8-bit Immediate value
        write   hexbyte
        write   comma
        get     byte3           ;2nd operand = relative jump offset
        get     relative
        dwl     0000000011111111b
        write   label           ;rel jump to address L####
        exit
```

If an instruction contains additional operand bytes, they must be read in at the very start of the opcode process routine. This enables XDASM to properly handle incomplete instructions that are caused by an "end of file" or a "new load address origin".

## 3.6 OPCODE PROCESSING COMMANDS

```
READ        BYTE
```

Reads in the next byte from the input file. Stores the value into its associated position, ie. if 2nd byte, then value is stored in BYTE2, if 3rd then BYTE3 and so on, up to maximum of eight bytes. If the "end-of-file" or a "load address change" is encountered, all accumulated bytes are written as a Define Byte statement.

```
WRITE       MNEMONIC
DFB         "text",0
```

Writes a carriage-return and line-feed, and if address was referenced, writes the Label name (Lnnnn:). The tab character is then written followed by the "text" message. Text messages are null terminated.

```
STORE       MNEMONIC
DFB         "text",0
```

Stores the "text" message into a pseudo buffer for later recall. Text messages are null terminated.

```
RECALL      MNEMONIC
```

Same as "WRITE MNEMONIC" except, text message is retrieved from pseudo buffer.

```
APPEND      MNEMONIC
DFB         "text",0
```

Writes the "text" message. Text messages are null terminated.

```
WRITE       (
```

Writes the opening parentheses character.

```
WRITE       )
```

Writes the closing parentheses character.

```
WRITE       COMMA
```

Writes a comma character.

```
WRITE       [
```

Writes the opening bracket character.

```
WRITE       ]
```

Writes the closing bracket character.

```
WRITE       #
```

Writes the number sign character.

```
WRITE       TAB
```

Writes the tab character.

```
WRITE       NIBBLE
```

Writes a single hexadecimal character, 0 to F, defined by the lower 4-bits of the pseudo accumulator.

```
WRITE   TEXT
DFB     "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16",0
```

Writes the text string indexed by the lower 4-bits of the pseudo accumulator. Up to sixteen, 7-character, text message entries can be defined. Each individual text string is position dependant and is separated by a comma. The complete text array is null terminated.

```
WRITE       SIGNED
DWL         mask
```

Writes the signed binary number in the pseudo accumulator, using the mask bit length, as Decimal ASCII characters preceded by a plus "+" or minus "-" character. Mask bit lengths are minimum 3-bits (-4 to +3) up to 16-bits (-32768 to +32767) maximum.

| OPNG |
|---|

Causes XDASM to write the opcode byte as "Define Byte" statement. Used when an unassigned opcode is encountered.

| WRITE | HEXBYTE |
|---|---|

Writes the low byte value of the pseudo accumulator as an ASCII Hex notation (ie. 0nnH or $nn).

| WRITE | HEXWORD |
|---|---|

Writes the value of the 16-bit pseudo accumulator as an ASCII Hex notation (ie. 0nnnnH or $nnnn).

| WRITE | LABEL |
|---|---|

Writes the value of the 16-bit pseudo accumulator as a Label operand, (ie. Lnnnn ). Logs the cross-reference to this Label.

| WRITE | TEXT1 |
|---|---|

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Text = | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | RA | RB | RC | RD | RE | RF |

| WRITE | TEXT2 |
|---|---|

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Text = | B | C | D | E | H | L | (HL) | A |

WRITE TEXT3

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Text = | NZ | Z | NC | C | PO | PE | P | M |

WRITE TEXT4

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Text = | BC | DE | HL | SP |

WRITE TEXT5

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 |
|---|---|---|
| Text = | IX | IY |

WRITE TEXT6

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Text = | X | Y | U | S |

| WRITE | TEXT7 |
|---|---|

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Text = | B | C | D | E | H | L | M | A |

| WRITE | TEXT8 |
|---|---|

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Text = | B | D | H | SP |

| WRITE | TEXT9 |
|---|---|

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Text = | B | D | H | PSW |

| WRITE | TEXT10 |
|---|---|

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Text = | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | R10 | R11 | R12 | R13 | R14 | R15 |

| WRITE | NUMBER |
|---|---|

Using the pseudo accumulator as an index, writes the character(s) from the stored text array.

| Index= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Text = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

| WRITE | OVn | (n = 1, 2 or 3) |
|---|---|---|

Using the pseudo accumulator as an 8-bit index, writes the mnemonic text substitution from the specified overlay buffer.

| LOG | XREFn | (n = 1, 2 or 3) |
|---|---|---|

Stores the 16-bit value of the pseudo accumulator together with the current instructions address into the specified cross-reference buffer.

```
EXIT
```

Returns control back to XDASM. The next opcode byte is fetched.

```
EXIT1
```

Returns control back to XDASM. A blank comment line is written then the next opcode byte is fetched.

```
EXIT2
```

Returns control back to XDASM. A blank comment line is written, another comment line containing dash characters, then the next opcode byte is fetched.

```
GOTO
DWL        address
```

Causes a jump to the designated 16-bit address.

```
SEARCH
DWL        mask,match,jump

<additional match statements>

DWL        0,0,nomatch                  ;end of search list
```

The 16-bit pseudo accumulator is ANDed with the 16-bit "**mask**" value, compared to the 16-bit "**match**" value and if equal, jumps to the 16-bit "**jump**" address. An unlimited number of match statements can be used, the last statement with zero mask, zero match, contains the 16-bit "**nomatch**" jump address.

```
TRANSLATE
DWL        mask,from,to

<additional translate statements>

DWL        0                            ;end of translate list
```

The 16-bit pseudo accumulator is ANDed with the 16-bit "**mask**" value, compared to the 16-bit "**from**" value and if equal, the "**to**" value is loaded into the pseudo accumulator, otherwise, the zero "mask" marks the end and the pseudo accumulator is left unchanged.

```
VECTOR
DWL          address          ;jump address if value = 0000H
DWL          address          ;jump address if value = 0001H
 "              "
DWL          address          ;jump address if value = 00FEH
DWL          address          ;jump address if value = 00FFH
```

Using the lower 8-bits of the pseudo accumulator as the index, causes a vectored jump to the specified "**address**". The vector list contains 256 16-bit addresses.

```
HIBYTE
```

Writes the high byte of the pseudo accumulator as two ASCII hex characters (00 to FF).

```
LOBYTE
```

Writes the low byte of the pseudo accumulator as two ASCII hexcharacters (00 to FF).

```
HIHEX
```

Writes the high byte of the pseudo accumulator in ASCII hex notation (0nnH or $nn).

```
LOHEX
```

(Same as WRITE HEXBYTE)

Writes the low byte of the pseudo accumulator in ASCII hex notation (0nnH or $nn).

```
WORD
```

Writes the pseudo accumulator as four ASCII hex characters (0000 to FFFF).

```
TASK
DWL         value
```

Sets the general purpose Task Register to the defined value.

```
GET     TASK
```

Loads the value of the general purpose Task Register into the pseudo accumulator.

## 3.7 FUNCTION COMMANDS

The Function commands are used to extract, calculate and manipulate the required portions of an instruction which make up the operand. All function results are returned in the pseudo accumulator. The following sections describe the command functions.

**! Important !** The additional operand bytes must be read in before they can be accessed using the "**GET BYTEn**" command.

| | | |
|---|---|---|
| GET | BYTEn | (n = 1 to 16) |
| GET | BYTEn+n | (n = 1 to 16) |

| Function command | | Pseudo accumulator after execution |
|---|---|---|
| *8-bit functions* | | |
| GET | BYTE1 | 00000000 : BYTE1 |
| GET | BYTE2 | 00000000 : BYTE2 |
| GET | BYTE3 | 00000000 : BYTE3 |
| GET | BYTE4 | 00000000 : BYTE4 |
| GET | BYTE5 | 00000000 : BYTE5 |
| GET | BYTE6 | 00000000 : BYTE6 |
| GET | BYTE7 | 00000000 : BYTE7 |
| GET | BYTE8 | 00000000 : BYTE8 |
| *16-bit functions* | | |
| GET | BYTE1+2 | BYTE1 : BYTE2 |
| GET | BYTE2+3 | BYTE2 : BYTE3 |
| GET | BYTE3+4 | BYTE3 : BYTE4 |
| GET | BYTE4+5 | BYTE4 : BYTE5 |
| GET | BYTE5+6 | BYTE5 : BYTE6 |
| GET | BYTE6+7 | BYTE6 : BYTE7 |
| GET | BYTE7+8 | BYTE7 : BYTE8 |
| *16-bit functions using inverted byte format* | | |
| GET | BYTE2+1 | BYTE2 : BYTE1 |
| GET | BYTE3+2 | BYTE3 : BYTE2 |
| GET | BYTE4+3 | BYTE4 : BYTE3 |
| GET | BYTE5+4 | BYTE5 : BYTE4 |
| GET | BYTE6+5 | BYTE6 : BYTE5 |
| GET | BYTE7+6 | BYTE7 : BYTE6 |
| GET | BYTE8+7 | BYTE8 : BYTE7 |

| GET | PAGE |
|---|---|

Replaces the high order 8-bit value of the pseudo accumulator with the high order 8-bit contents of the current instructions address.

*Example*
```
Pseudo acc before execution:  5678h    (0101  0110  0111  1000)
Instruction address (PC):     1234h    (0001  0010  0011  0100)
Pseudo acc after execution:   1278h    (0001  0010  0111  1000)
```

| GET | BITS |
|---|---|
| DWL | mask |

Extracts and right-justifies the selected bits of the pseudo accumulator as defined by the 16-bit mask value.

*Example*
```
Pseudo acc before execution:  5678h    (0101  0110  0111  1000)
Value of 16-bit mask:         003ch    (0000  0000  0011  1100)
Pseudo acc after execution:   000eh    (0000  0000  0000  1110)
```

| GET | RELATIVE |
|---|---|
| DWL | mask |

Calculates the absolute address, relative to the current instruction, using the signed contents of the pseudo accumulator as defined by the 16-bit mask value. The absolute address is returned in the pseudo accumulator.

*Example*
```
If mask= 001fh,  then relative range would be -16 to +15.
If mask= 00ffh,  then relative range would be -128 to +127.
If mask= 0ffffh, then relative range would be -32768 to +32767.
```

| LOGICAL | AND |
|---|---|
| DWL | argument |

Logically AND's the value of the pseudo accumulator with the value defined in the 16-bit argument. The result is returned in the pseudo accumulator.

*Example*
```
Pseudo acc before execution:   5678H    (0101  0110  0111  1000)
Contents of 16-bit argument:   0F00H    (0000  1111  0000  0000)
Pseudo acc after execution:    0600H    (0000  0110  0000  0000)
```

```
LOGICAL    OR
DWL        argument
```

Logically OR's the value of the pseudo accumulator with the value defined in the 16-bit argument. The result is returned in the pseudo accumulator.

*Example*

```
Pseudo acc before execution:  5678H   (0101 0110 0111 1000)
Contents of 16-bit argument:  0081H   (0000 0000 1000 0001)
Pseudo acc after execution:   56F9H   (0101 0110 1111 1001)
```

```
LOGICAL    XOR
DWL        argument
```

Logically Exclusive OR's the value of the pseudo accumulator with the value defined in the 16-bit argument. The result is returned in the pseudo accumulator.

*Example*

```
Pseudo acc before execution:  5678H   (0101 0110 0111 1000)
Contents of 16-bit argument:  1234H   (0001 0010 0011 0100)
Pseudo acc after execution:   444CH   (0100 0100 0100 1100)
```

```
SAVE
```

Saves a copy of the pseudo accumulator contents in a temporary buffer.

```
COMBINE
```

Combines (logically or's) the contents of the temporary buffer with the current contents of the pseudo accumulator.

*Miscellaneous Functions*

| Function command | Pseudo accumulator after execution |
|---|---|
| GET PC | 16-bit address of current instruction |
| GET (8P8) | 11-bit absolute address (8048 / 8051) |

## 3.8 COMPILING THE CPU TABLE

After the CPU program code is written, it must be converted into a binary image for XDASM. The XTC.EXE compiler is used to create the CPU binary.

**NOTE:** The following assembler directives must be present in the CPU table program before it is compiled by XTC.

```
CPU     "XDASM.TBB"
HOF     "BIN8"
ORG     8000H
```

The "CPU" directive statement instructs XTC to load and use the "XDASM.TBB" table list for its compilation. This table contains the converted instructions for the high-level commands.

The "HOF" directive statement instructs XTC to produce a binary image for its output file.

The "ORG" directive statement defines the memory address of where the CPU table code will reside.

To start XTC, enter the following command line ("type" is the name of your CPU program):

```
XTC type.ASM -L type.LST -H type.CPU
```

After a successful completion, two files are created; "type.LST" which contains the assembled program listing and "type.CPU" which is the binary image that XDASM uses for the CPU table.

*< This page intentionally left blank >*

## 4.0 ERROR MESSAGES

**ERROR -- No Input File Specified or No CPU Type Specified**
Name was missing in the command line.
Command Line syntax: XDASM <filename>[.ext],<cpu> [/options]

**ERROR -- File Not Found: <filename>**
File was not found on current disk or directory.
Check for proper filename, drive or path.

**ERROR -- File is Empty: <filename>**
File did not contain any data.
Check file contents.

**ERROR -- Bad Format in File: <filename>**
File structure did not conform to XDASM's specifications.
Check file contents.

**Unexpected DOS Error in File: <filename>**
DOS reported an error when creating or closing the file.
Possible Write Protected disk or unfamiliar DOS structure.
Consult your DOS User's Guide.

**ERROR -- Insufficient Disk Space**
Disk or Directory Full.
Insert a new disk or delete unused files.

**Checksum Error in Hex File: <filename>**
Checksum did not compare or unrecognized hex file format.
Try another HEX file.

**Warning - Cross-reference overflow**
A Cross-reference list will be truncated.
Disassembly will continue.

## 4.1 INTEL HEX FORMAT

**DATA RECORDS ---**

| BYTE # | |
|---|---|
| 1 | Colon (:), signifies start of a record. |
| 2 & 3 | Number of data bytes in this record. |
| 4 & 5 | Load address for this record, High Byte. |
| 6 & 7 | Load address for this record, Low Byte. |
| 8 & 9 | Record type, must be "00". |
| 10 to X | Data bytes, two ASCII hex characters each. |
| X+1 & X+2 | Checksum, two ASCII hex characters. |
| X+3 & X+4 | CR & LF, (carriage return & line-feed). |

**END RECORD ---**

| BYTE # | |
|---|---|
| 1 | Colon (:), signifies start of a record. |
| 2 & 3 | Record length, must be "00". |
| 4 to 7 | Start address, "0000" = end record. |
| 8 & 9 | Record type |
| 10 & 11 | Checksum, two ASCII hex characters. |
| 12 & 13 | CR & LF, (carriage return & line-feed). |

The CHECKSUM is the two's complement of the 8-bit sum of the Record Length, the two byte Load Address, the Record Type, and all the Data bytes.

## 4.2 MOTOROLA HEX FORMAT

**DATA RECORDS ---**

| BYTE # | | |
|---|---|---|
| | 1 & 2 | "S1" signifies start of a data record. |
| | 3 & 4 | Number of data bytes in this record. |
| | 5 & 6 | Load address for this record, High Byte. |
| | 7 & 8 | Load address for this record, Low Byte. |
| | 9 to X | Data bytes, two ASCII hex characters each. |
| | X+1 & X+2 | Checksum, two ASCII hex characters. |
| | X+3 & X+4 | CR & LF, (carriage return & line-feed). |

**END RECORD ---**

| BYTE # | | |
|---|---|---|
| | 1 & 2 | "S9" signifies end of file. |
| | 3 & 4 | Number of data bytes in this record. |
| | 5 to 8 | Start address |
| | 9 & 10 | Checksum, two ASCII hex characters. |
| | 11 & 12 | CR & LF, (carriage return & line-feed). |

The CHECKSUM is the one's complement of the 8-bit sum of the Record Length, the two byte Load Address, and all the Data bytes.

## 4.3 PROCESSOR FILE DESCRIPTIONS

Each processor type has three or more associated disk files;

```
type.MNE
type.CPU
type.FMT
type.EQU                              (type = processor name)
type.XR1
type.XR2
type.XR3
```

The "type.MNE" is an ASCII text file and is not used by XDASM. This file contains the mnemonic instruction list, the file usage list and the cross-references that will be generated for this processor type. The user can list or print this file for general reference.

The "type.CPU" is the control file used for directing the disassembly process. This is a binary file and does not contain any modifiable information.

The "type.FMT" is an ASCII file used by XDASM when generating the output source file. It contains the directive formats relevant to a specific assembler. This file can be easily modified using a non-document mode text editor.

The "type.EQU" is an optional ASCII text file that is transferred to the source output file when XDASM is started. This file usually contains Equate Definitions, Assembler Directives or any other text the user wishes written. Since it is transferred unmodified, the appropriate assembler format must be used. This file need not be present for XDASM to operate.

The "type.XRn" files are ASCII text substitution files used by XDASM to replace 8-bit hex notations with symbolic text. These files are generally used by processors that contain special function registers or peripheral addresses mapped into a 256-byte page. The usage of these files are indicated in the "type.MNE" listing. Initially, these files contain all hex notation entries.

## 4.4 MANUFACTURER DIRECTORY

| | | |
|---|---|---|
| California Micro Devices<br>2000 West 14th Street<br>Tempe, AZ 85281<br>(602) 968-4431<br><br>6502<br>65C02 | Hitachi America Ltd.<br>Semiconductor & IC Div.<br>2000 Sierra Point Pkwy<br>Brisbane, CA 94005<br>(415) 244-7124<br><br>64180<br>6301<br>6800<br>6801<br>6805<br>6809 | Motorola Inc.<br>Microproc. Product Group<br>6501 William Cannon Dr. W.<br>Austin, TX 78735<br><br>6800<br>6801<br>6805<br>6809<br>68HC11 |
| Fujitsu Microelectronics Inc.<br>3330 Scott Blvd<br>Santa Clara, CA 95054<br>(408) 727-1700<br><br>6800<br>6809<br>8048<br>8051 | Intel Corp.<br>P.O. Box 58130<br>Santa Clara, CA 95052<br>(800) 548-4725<br><br>8048<br>8051<br>8085<br>8096 | National Semiconductor Corp<br>2900 Semiconductor Drive<br>Santa Clara, CA 95052<br>(408) 721-5121<br><br>8048<br>COP400<br>COP800 |
| Harris Semiconductor<br>724 Route 202<br>Somerville,NJ 08876<br>(201) 685-6849<br><br>1802<br>6805<br>8051<br>8085 | Mitsubishi Elect. America Inc.<br>1050 East Arques Ave.<br>Sunnyvale, CA 94086<br>(408) 730-5900<br><br>8085 | NCR Microelect. Products<br>1635 Aeroplaza Dr.<br>Colorado Springs, CO 80916<br>(719) 596-5795<br><br>6502<br>65C02 |

| | | |
|---|---|---|
| NEC Electronics<br>Corporate Headquarters<br>401 Ellis St.<br>Mountain View, CA 94039<br>(415) 960-6000<br><br>8048<br>8085<br>Z80 | Rockwell International<br>Digital Comm. Division<br>4311 Jamboree Road<br>Newport Beach, CA 92658<br>(714) 833-6840<br><br>65C02 | Texas Instruments<br>P.O. Box 809066<br>Dallas, TX 75380<br>(800) 336-5236<br><br>TMS370<br>TMS7000<br>TMS9900 |
| OKI Semiconductor Inc.<br>650 N. Mary Ave.<br>Sunnyvale, CA 94086<br>(408) 720-1900<br><br>8048<br>8051<br>8085 | SGS-Thomson Microelect Inc.<br>1000 East Bell Road<br>Phoenix, AZ 85002<br>(602) 867-6259<br><br>6800<br>6801<br>6805<br>6809<br>SUPER8<br>Z8<br>Z80 | Toshiba America Inc.<br>9775 Toledo Way<br>Irvine, CA 92718<br>(714) 455-2000<br><br>6801<br>6805<br>68HC11<br>8048<br>8085<br>Z80 |
| Philips Components- Signetics<br>811 East Arques Ave.<br>P.O. Box 3409<br>Sunnyvale, CA 94088<br>(408) 991-3445<br><br>8048<br>8051<br>8096 | Siemens Components Inc.<br>2191 Laurelwood Road<br>Santa Clara, CA 95054<br>(408) 980-4592<br><br>8048<br>8051<br>8085 | Zilog<br>210 Hacienda Ave.<br>Campbell, CA 95008<br>(408) 370-8000<br><br>SUPER8<br>Z8<br>Z80<br>Z180 |

# READ EPROMS USING THE PC'S PARALLEL PORT

7805 +5V REG.
OUT IN
COM
.1 uF
.1 uF
+
8 - 12
V.D.C.
−

10K ohm resistor network SIP
10 9 8 7 6 5 4 3 2 1

To LPT port

14 AUTO
9 PD7
8 PD6
7 PD5
6 PD4
5 PD3
4 PD2
3 PD1
2 PD0
1 STROBE
17 SEL OUT (L=D0-D3, H=D4-D7)
11 BUSY (D3 / D7)
10 ACK (D2 / D6)
12 PAPER (D1 / D5)
13 SEL IN (D0 / D4)
18 GND
19 GND
20 GND
21 GND

**DB-25P**
25-PIN CONNECTOR

74LS373
11 LE
+5V 20
18 D7 Q7 19
17 D6 Q6 16
14 D5 Q5 15
13 D4 Q4 12
8 D3 Q3 9
7 D2 Q2 6
4 D1 Q1 5
3 D0 Q0 2
1 OE
10 GND

B
A
OPTION

74LS157
15 OE
+5V 16
1 SEL
4B 13
3B 10
2B 6
1B 3
12 4Y
9 3Y
7 2Y
4 1Y
4A 14
3A 11
2A 5
1A 2
8 GND

28 +5V
1 A15 / VP
27 A14 / PGM
2 A12
23 A11 / VP
21 A10
24 A9
25 A8
26 A13 / VC
3 A7
4 A6
5 A5
6 A4
7 A3
8 A2
9 A1
10 A0
20 CS
22 OE
19 D7
18 D6
17 D5
16 D4
15 D3
13 D2
12 D1
11 D0
14 GND

**EPROM**
**28-PIN IC SOCKET**

**"OPTION A"** for 2716 & 2732 / **"OPTION B"** for 2764, 27128, 27256 & 27512

Use with BASIC program "READROM.BAS"

Data Sync Engineering 11/90

```
10 REM      ******************************************************
20 REM      * EPROM Reader Program, DATA SYNC ENGINEERING, 11/90 *
30 REM      ******************************************************
40 REM set LPT address for: LPT1/378H= 888, LPT2/278H= 632 or LPT3/3BCH= 956
50 LPT=888
60 REM
70 CLS : PRINT : PRINT : PRINT
80     PRINT "---- EPROM READER by Data Sync Engineering ----" : PRINT
90    PRINT "1.    2716      2  KB     Option A jumper"
100    PRINT "2.    2732      4  KB     Option A jumper"
110    PRINT "3.    2764      8  KB     Option B jumper"
120    PRINT "4.    27128     16 KB     Option B jumper"
130    PRINT "5.    27256     32 KB     Option B jumper"
140    PRINT "6.    27512     65 KB     Option B jumper" : PRINT
150    PRINT "7.    RETURN TO DOS" : PRINT : PRINT
160    INPUT "  Choose EPROM Type (1-7)";TYP
170       IF TYP<1 OR TYP >7 THEN GOTO 70
180       IF TYP=7 THEN OUT LPT+2,0 : SYSTEM
190       IF TYP=1 THEN MAX=2048 : MASK%=8
200       IF TYP=2 THEN MAX=4096 : MASK%=0
210       IF TYP=3 THEN MAX=8192 : MASK%=192
220       IF TYP=4 THEN MAX=16384 : MASK%=192
230       IF TYP=5 THEN MAX=32768! : MASK%=128
240       IF TYP=6 THEN MAX=65536! : MASK%=0
250 OPEN "READROM.HEX" FOR OUTPUT AS #1
260    ADR = 0 : LOCATE 18,1 : PRINT "Reading address: ";
270    ADRH% = INT(ADR/256) : ADRL% = ADR-ADRH%*256 : SUM%=16+ADRH%+ADRL%
280       IF ADR=0 THEN PRINT #1, ":10000000"; : GOTO 330
290       IF ADR<16 THEN PRINT #1, ":10000";HEX$(ADR);"00"; : GOTO 330
300       IF ADR<256 THEN PRINT #1, ":1000";HEX$(ADR);"00"; : GOTO 330
310       IF ADR<4096 THEN PRINT #1, ":100";HEX$(ADR);"00"; : GOTO 330
320    PRINT #1, ":10";HEX$(ADR);"00";
330    LOCATE 18,18 : PRINT ADR
340    FOR ADR = ADR TO ADR+15 : GOSUB 430
350       IF BYT%<16 THEN PRINT #1, "0";
360    PRINT #1, HEX$(BYT%); : SUM%=SUM%+BYT% : NEXT ADR
370    HBYTE% = INT(SUM%/256) : LBYTE% = SUM%-HBYTE%*256 : BYT%=256-LBYTE%
380       IF BYT%=256 THEN BYT% = 0
390       IF BYT%<16 THEN PRINT #1, "0";
400    PRINT #1, HEX$(BYT%)
410       IF ADR < MAX THEN GOTO 270
420    PRINT #1, ":00000001FF" : CLOSE #1 : GOTO 70
430 HBYTE% = INT(ADR/256) : LBYTE% = ADR-HBYTE%*256 : HBYTE% = HBYTE% OR MASK%
440    OUT LPT,HBYTE% : OUT LPT+2,8 : OUT LPT+2,10 : OUT LPT,LBYTE%
450    OUT LPT+2,11 : BYT% = INP(LPT+1) : BYT% = BYT% AND 240
460       IF BYT% < 128 THEN BYT% = BYT% + 128 : GOTO 480
470    BYT% = BYT% AND 112
480    BYT% = BYT%/16
490    OUT LPT+2,3 : TBYT% = INP(LPT+1) : TBYT% = TBYT% AND 240
500       IF TBYT% < 128 THEN TBYT% = TBYT% + 128 : GOTO 520
510    TBYT% = TBYT% AND 112
520    OUT LPT+2,10 : BYT% = BYT% OR TBYT% : RETURN
```

# XDASM Cross-Disassembler

XDASM is a powerful, MS-DOS based Cross-Disassembler which is used to reconstruct or debug source level code for various processor types. Its unique table-driven structure and output format adaptibility, makes XDASM the most universal program disassembler available.

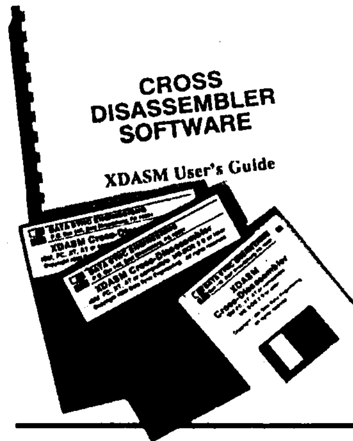
XDASM's disassembly process can be directly controlled by a user generated TAG file. The output source file is created from an Intel/Motorola Hex or Binary coded input file of up to 65K bytes.

XDASM is very easy to use and produces an "Assembler-ready" source file that can be immediately re-assembled with little or no editing.

Optional comment fields are attached to each disassembled line which show the current Program Counter, Instruction HEX bytes and ASCII character translations.

XDASM automatically inserts Origin, Define Byte and Equate directives when required. Labels are assigned to every instruction address that is referenced. De-blocking is used in the output listing to allow easier program interpretation.

Cross-Reference lists can be optionally appended to the output file to provide further program detail. The sorted reference lists are maintained as comment fields which are ignored by the assembler. Based on the processor type, up to four cross-reference lists may be generated. Label name address references are always provided. Other reference lists may include, Immediate Values, I/O Addresses and Special Address Registers. XDASM is capable of producing up to 130K of sorted references.




* PC/MS-DOS based
* Table-driven disassembly
* Hex or Binary input files
* Creates assembly source
* Direct disassembly control
* Manufacturers Mnemonics
* Assigns Label names
* Inserts assembler directives
* Deblocks output listing
* Generates Cross-References

## Command Line

XDASM filename,type /options

filename = Hex/Binary input file
type = Processor name

Options:

| | |
|---|---|
| B | Binary file input |
| C | Include line comments |
| L | Lower-case output |
| M | Mask 7-bit ASCII |
| R | Append cross-references |
| T | Use TAG file control |
| X | Cross-references only |

## Tables Included

* 1802 1805 1806
* 64180 Z180
* 6502 65C02
* 6800 6802 6808
* 6801 6803
* 6301 6303
* 6805
* 6809
* 68HC11
* 8048
* 8051
* 8085 8080
* 8096
* COP400
* COP800
* Z8
* Z80
* . . . call for others

## Output Format Control

Format file = type.FMT

(Current default setting)

| | |
|---|---|
| ; | Insert directive, line 1 |
| ; | Insert directive, line 2 |
| DFB | Define Hex byte directive |
| DFB | Define Text directive |
| DWH | Define Word, MSB first |
| DWL | Define Word, LSB first |
| ORG | Origin directive |
| EQU | Equate directive |
| END | End directive |
| SRC | Output file extension |
| L | Start of label character |
| : | End-of-label character |
| : | End-of-equate character |
| ; | Comment field delimiter |
| " | Text string delimiters |
| 0 | Hex notation format |

## Tag File Control

Tag file = filename.TAG

Disassembly commands:

| | |
|---|---|
| aaaa=B | Define Byte |
| aaaa=H | Hex load addr offset |
| aaaa=I | Instruction |
| aaaa=S | Skip byte |
| aaaa=T | Text byte |
| aaaa=> | Define Word, LSB |
| aaaa=< | Define Word, MSB |

(aaaa = starting address)

## Condensed Sample Listing

```
;=============================================================;
; Disassembled Using XDASM   --   (C)1990 Data Sync Engineering ;
;=============================================================;
;
;        Unresolved Address Reference list
;
L00CD:   EQU   000CDH
L00EF:   EQU   000EFH
L00F5:   EQU   000F5H
;
         ORG   00000H
;
L0000:   LD    HL,08000H          ;0000 21 00 80       !..
;
L0003:   DEC   HL                 ;0003 2B             +
         LD    A,H                ;0004 7C             |
         OR    L                  ;0005 B5             .
         JP    NZ,L0003           ;0006 C2 03 00       ...
;
;        Cross-references to LABEL Addresses
;
; L0000= 1D18
; L0003= 0006
; L0009= 01E1  01FC  02A3  02BE  0506  0524  06E4  06FF  085B  0AA3
;        0B0A  0B51  0CC5  0DC1  1172
```

### Automatic Directives

ORG - Origin directives are used to specify the memory location of where the program resides. XDASM inserts origin directives in accordance with the input files specifications. Any time the Hex load address changes, XDASM will insert a new origin statement. An address offset can be added by using the "H" command in the TAG file.

DFB - Define Byte directives are inserted whenever an Unassigned Opcode or an Incomplete Instruction is encountered from the input file. The "B" command in the TAG file will also cause Define Byte statements.

EQU - If an address is referenced and is not found within the program, XDASM will equate a label to that address and will show it in the Unresolved Address Reference list. The key to a successful disassembly is to eliminate all unresolved references. If the address is valid, it can be manually equated by using the "G" command in the TAG file. This removes that address from the unresolved list.

### Line Comments

Line comments are enabled by using the "C" option in the command line. Each disassembled line will contain a comment field showing the Instruction Address, the HEX bytes that made up the instruction and the ASCII character equivalent of the Hex bytes. This is very useful for distinguishing between code and text. In some cases the high bit may be set as a flag or to disguise text. The "M" option will mask the high bit for ASCII character display.

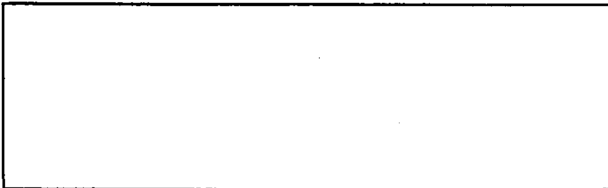
### The TAG File

The TAG file is a separate ASCII character file that may be generated using a standard text editor. It is used to control the disassembly process by switching to various modes at specified addresses. Byte, Word, Text, Skip and Instruction modes are easily selected by entering the start address followed by a command character.

### System Requirements

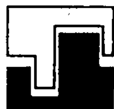
MS-DOS version 2.0 or later.
512 Kilobytes RAM.
5.25" / 3.50" floppy drive.

**Your local sales representative is:**



*** Ask about the companion Cross-Assembler.**

MS-DOS is a registered trademark of Microsoft Corporation.



**DATA SYNC ENGINEERING**
P.O. Box 146, East Stroudsburg, PA 18301 **(717) 421-1977**