XA80

User Manual

Contents

[1. Introduction 3](#_Toc107574882)

[1.1. Document purpose 3](#_Toc107574883)

[1.2. Application scope 3](#_Toc107574884)

[2. XA80 Syntax 4](#_Toc107574885)

[3. Directives 4](#_Toc107574886)

[4. Instructions 5](#_Toc107574887)

[4.1. Labels 6](#_Toc107574888)

[4.2. Macro local labels 6](#_Toc107574889)

[4.3. Opcodes 7](#_Toc107574890)

[4.4. Operands 7](#_Toc107574891)

[4.5. Comments 7](#_Toc107574892)

[5. Expressions 8](#_Toc107574893)

[5.1. Literal values 8](#_Toc107574894)

[5.2. Symbols 9](#_Toc107574895)

[5.3. Operators and Expression Precedence 9](#_Toc107574896)

[5.4. Integer Functions 10](#_Toc107574897)

[5.5. String Functions 10](#_Toc107574898)

[6. Appendices 12](#_Toc107574899)

[6.1. Appendix - Command Line Usage 12](#_Toc107574900)

[6.2. Appendix – Opcodes 13](#_Toc107574901)

[7. Index 17](#_Toc107574902)

Document Release Status

|  |  |  |
| --- | --- | --- |
| Version | Date | Changes |
| 0.1 |  | Initial document release in draft form |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Contact

For contact about the content of this document, please contact Duncan Munro [duncan@duncanamps.com](mailto:duncan@duncanamps.com)

# 

# Introduction

## Document purpose

This document is the User Manual for XA80, **X** (Cross) **A**ssembler for x**80** processors. Its purpose is to provide a reference on how the application should be used, with examples where appropriate.

## Application scope

XA80 is intended to be used with the following 8/16 bit processors:

* 8080
* 8085
* Z80
* Z180

Being open source, the software naturally lends itself to being extensible should other processor types or families be required.

# 

# XA80 Syntax

Main elements are:

* Directives
* Instructions

Instructions are further divided into

* Labels
* Opcodes
* Operands
* Comments

# Directives

A full list of directives is shown below. Some items are synonyms to allow flexibility with a wide range of source code material.

The format is:

<optional\_label>  **<directive>**  <expression(s)> or…

<mandatory\_label> **<directive>** <expression(s)>

Mandatory label applies to the EQU and = directives, all others are optional. See section 4.1 for more details on how labels are used.

The list of directives is:

|  |  |  |  |
| --- | --- | --- | --- |
| Directive | Parameters | Description | Example |
| = | exprU16 or…  exprStr | Assigns a value to a mandatory label. Much like EQU, however the = construct can be used more than once while using EQU to redefine the value of a label would result in an error | START = 0FFFAH  NAME = ‘John’ |
| DB | listA8 | Defines a series of bytes in memory. The expression list is a comma separated list of 8 bit numeric expressions which can include single characters enclosed in quotes | DB 12, 0FAH, -12, ’H’, 0 |
| DC | listStr | Define Characters. Like DM however bit 7 of the last character in the string is set to 1. This is useful for lists of keywords etc. | DC “FOR”, “NEXT” |
| DEFB | Synonym for DB | | |
| DEFC | Synonym for DC | | |
| DEFM | Synonym for DM | | |
| DEFS | Synonym for DS | | |
| DEFW | Synonym for DW | | |
| DEFINE | symbol | Defines a symbol with a NULL value. The symbol must not exist already | DEFINE dump\_text |
| DM | listStr | Define Message. Stores the list of strings in memory | DM ‘Help’,’me’ |
| DS | exprU16 | Define Storage. Reserves an amount of memory given by expression | DS 100H ; Reserve 256 |
| DS | exprU16, exprA8 | Define Storage, second form. Fills an area of memory with the first expression for a number of bytes determined by the second expression. | DS 100H,0AAH ; Bits  DS 10H,’ ‘ ; Spaces |
| DW | listA16 | Defines a series of words (16 bit) in memory. These are stored in a little-endian form | DW -1, 0C800H, 8192 |
| ELSE |  | Marks the end of an IF block and the start of a ELSE block |  |
| END |  | Marks the end of the assembly, does not need to be present |  |
| ENDIF |  | Marks the end of an IF or IF / ELSE block |  |
| ENDM |  | Ends a MACRO block |  |
| EQU | exprU16 or…  exprStr | Assigns the expression to the label. A label is mandatory with this directive | START EQU 0800H |
| ERROR | exprStr | Raises the error message listed in the string expression and halts the assembly | ERROR ‘Wrong!!!’ |
| IF | exprA16 | Evaluates the expression and if it’s zero the following lines are not assembled. Used in conjunction with the ELSE and ENDIF directives. IF statements can be nested. The expression must be known on the first pass or the assembly will fail | IF mask\_active |
| IFDEF | symbol | Similar to IF but activates the following code if the symbol exists | IFDEF DEBUG |
| IFNDEF | symbol | Similar to IF but activates the following code if the symbol does not exist |  |
| INCLUDE | filename | Includes the filename into the source file. The INCLUDE directive can be nested. By default, include files are not listed. See INCLUDEF for details of how to list the included file | INCLUDE “foo.inc” |
| INCLUDEF | filename | Like INCLUDE, however listing is forced on (F=Force) |  |
| LIST |  | Turns the list file on (the default) |  |
| MACRO | name params | Defines a macro, concludes with a ENDM directive | MACRO foo from,to |
| MESSAGE | exprStr | Includes the message in the string expression into the assembly listing | MESSAGE “Complete” |
| NOLIST |  | Turns the assembly listing off |  |
| ORG | exprU16 | Sets the assembly origin to the expression. A default value of 0 is used if this directive has not been used at all | ORG 2000H  ORG $+2 ; Reserve spc |
| UNDEFINE | symbol | Removes a symbol from the symbol table | UNDEFINE foo |
| WARNING | exprStr | Issues the warning represented by the string expression | WARNING “No setup” |

The case of the directive is not significant, for example SET has the same effect as Set.

# Instructions

Instructions take the form

[label] [opcode [operand1[,operand2]]] [comment]

## Labels

The label takes the form of an alphabetic character or underscore followed by zero or more trailing characters. The trailing characters may be an alphabetic character, digit or underscore. Finally, this is suffixed by a colon ‘:’ to indicate a label. Examples are:

Start:

\_loop\_pos\_3:

KX0001:

Labels are case sensitive unless the command line switch @@@@@ has been used to switch this off.

## Macro local labels

Labels used within a macro are always local to that macro.

|  |
| --- |
| MACRO LOOP\_TEST  LD HL,2000H  LD B,16  XOR A,A  loop: LD [HL],A  INC HL  DJNZ loop  ENDM |

At expansion time the label is preceded by a local prefix purely for that expansion, for example:

|  |
| --- |
| LD HL,2000H  LD B,16  XOR A,A  @0001@loop: LD [HL],A  INC HL  DJNZ @0001@loop |

## Opcodes

Opcodes can be any one of the Z80 / Z180 opcodes, for example LD or

RRCA. A full list of opcodes can be found in section 6.2

## Operands

There are three different styles of operand which are handled by the application, these being 0 operands, 1 operand or 2 operands. Examples are:

|  |  |
| --- | --- |
| Operands | Examples |
| 0 | CCF  NOP  RETNZ |
| 1 | RST 0 |
| 2 | LD [HL],B |

### Simple operands

Simple operands are short and fixed definitions which typically refer to processor registers or flag conditions, the list is:

|  |  |  |
| --- | --- | --- |
| A  AF  AF’  B  BC  [BC]  C  [C]  D  DE  [DE] | [DE]  E  F  H  HL  [HL]  I  IX  [IX]  IY  [IY] | L  M  NC  NZ  P  PE  PO  R  SP  [SP]  Z |

### Complex operands

Complex operands represent the index with offset operands, specifically:

[IX+signed\_displacement]

[IY+signed\_displacement]

### Expressions in operands

Expressions can form part or all of the operand. Some examples are:

CP A,’\_‘ ; Check if underscore

LD A, [IY+4] ; Get the byte parameter

LD HL, 2000H ; Point to start of buffer

LD C, (1 << 3) | 80H ; Set up initial value

There is a rich set of expression operators and function available, these are discussed in more detail in section 5.

### Operand indirection

Indirection is indicated by the [ ] characters; the old style format of ( ) brackets is not allowed, some legal examples:

|  |
| --- |
| LD HL,[SAVED] ; Get saved HL back  OUT [C],A ; Send byte to port  LD A,[IX+4] ; Get parameter byte |

## Comments

Comments allow descriptive text to be added without influencing the operation of the assembler. There are two different types of comments available:

|  |  |  |
| --- | --- | --- |
| Style | Format | Description |
| 1a | optional\_text ; comment | Any text from a ; onwards will be treated as a comment. Text prior to the ; will be treated as valid information and will be processed by the assembler |
| 1b | optional text // comment | Any text from the // onwards will be treated as a comment |
| 2 | optional text {comment} optional text | Any text within the { } characters will be treated as a comment. Text to the left and right will be processed as normal |

The following code example shows how comments can be used:

|  |
| --- |
| //  // ASSEMBLY FILE TEST  //  BIT\_MASK EQU 01101001B ; Use this to get correct flags  FACTOR EQU (10 + 3 {record offset!}) \* 2  ; Code starts here  START:  XOR A,A ; Zero A  : : |

# Expressions

Expressions can be integer expressions or string expressions

Expressions are formed from literal values, symbols, operators and functions. Examples are:

A > B

1 << bit\_5

2 + 3 \* 4

LOW(address)

15 \* (1 + 2)

Pos(“-“,title)

IIF(i>5,1,0)

build()

Left(title,3)

IIFS(p==0,”Zero”,”Non-zero”)

## Literal values

Literal values can be:

1. Binary numbers, prefixed by %, 0b or suffixed with B. For example %01101001 or 0110B
2. Octal numbers, suffixed with letter O or Q. For example 123O or 777q
3. Decimal numbers – for example 123 or 0
4. Hexadecimal numbers, which can be prefixed by #, 0x or suffixed by H[[1]](#footnote-1). For example $33A, $ff78 or 33AH[[2]](#footnote-2)
5. String values enclosed in single or double quotes, for example “MyString”
6. ASCII values of characters in single or double quotes, for example ‘A’ returns the hex value 65

## Symbols

Symbols are constant values or variables used within the assembly. They can be associated with:

* A null value
* An integer value
* A string value

A null value is produced by the DEFINE directive where a symbol is declared but has not specific value associated with it. It can only be used with IFDEF or IFNDEF directives.

## Operators and Expression Precedence

Expressions are evaluated using the following precedence:

|  |  |
| --- | --- |
| Precedence | Element |
| 1 | ( bracketed expression ) |
| 2 | String to integer functions |
| 3 | Symbols  Special symbols  Numeric functions  + unary plus  - unary minus  ~ Not operator |
| 4 | \* multiplication  / division  & bitwise and  ^ bitwise xor  << shift left  >> shift right |
| 5 | + addition  - subtraction  | bitwise or |
| 6 | == comparison operators  !=  <  >  <=  >= |
| 7 | && boolean and  ^^ boolean xor |
| 8 | || boolean or |
| 9 | ! boolean not |
| 10 | = assignment operator |

## Integer Functions

These are functions returning an integer value. They may be dealing with strings.

|  |  |
| --- | --- |
| Function | Detail |
| ASC(string) | ASCII value of the first character in a string. Produces an error if the string is empty |
| HIGH(expression) | Returns the high byte of an expression |
| IIF(condition,trueexpr,falseexpr) | Immediate If, the expression returned depends if the condition is true or false |
| LOW(expression) | Returns the low byte of an expression |
| POS(substr,string) | Returns the position of a substring within another string in the range 1..n or zero if the substring is not found |
| VALUE(string) | Returns a value from a string for example “123” or “%011011”. Binary and hex strings are allowed |

## String Functions

These are functions returning a string value.

|  |  |
| --- | --- |
| Function | Detail |
| BUILD () | Current build string, for example 138 |
| CHR (value) | Returns the character corresponding to an ASCII value, for example CHR(65) will return “A” |
| DATE () | Returns todays date in the form YYYY-MM-DD, for example 2020-04-26 |
| HEX(value[,digits]) | Returns a value as a hex string without the preceding $ character. The correct number of digits will be used, for example 826 will convert to “33A” but it is possible to force the number of digits. As an example of that, HEX(826,4) will yield “033A”. If value cannot be represented in the number of digits, an error is produced |
| IIFS(condition,trueexpr,falseexpr) | Immediate If for strings, the string expression returned (truexpr, falseexpr) depends if the logical condition is true or false |
| LEFT (string,count) | Returns the leftmost count characters from a string. If count is greater than the length of the string, then the whole string will be returned. If count is < 1 then an error is produced |
| LOWER (string) | Returns a string in lowercase form |
| MID (string,start,count) | Returns the mid part of a string from start character for count characters. If start < 1 or count < 1 then an error is produced. If start+count is greater than the length of the string, the rightmost part from start characters is returned |
| RIGHT (string,count) | Returns the rightmost count characters from a string. If count is greater than the length of the string, then the whole string will be returned. If count is < 1 then an error is produced |
| STRING(value) | Returns the value as a string representation |
| TIME () | Returns the current time in the form HH:MM:SS, for example 08:05:33 |
| UPPER (string) | Returns a string in uppercase form |
| VERSION () | Current version string, for example 1.0.3.22 |

# 

# Appendices

## Appendix - Command Line Usage

From the program startup when invoking XA80 with no parameters:

6502 Macro Assembler V0.1

Copyright (C)2020-2022 Duncan Munro

Usage: asm65 filename <options>

Options:

-b <bn> --debug=<bn> Set the debug filename to <bn>

-d <id> --define=<id> Define one or more symbols

-e <en> --errorlog=<en> Set error log to <en>

-h --help Display this message

-I <id> --include=<id> Set the include directory to <id>

-l <ln> --listing=<ln> Set the listing name to <ln>

-m <mn> --map=<mn> Set the map filename to <mn>

-o <on> --object=<on> Set the object name to <on>

-t <n> --tab=<n> Tab size for input file (default 4)

-v <n> --verbose=<n> Verbose output while assembling

-V --version Display version and other status info

-x <hn> --hex=<hn> Set the hex filename to <hn>

<bn> / <en>/<ln>/<mn>/<on>/<hn> default to the filename with ext changed to

.log/.hex/.lst/.obj/.hex respectively. Not specifying <en>, <ln>, <mn>

or <hn> will stop that output.

verbose <n> options:

0 Normal output levels (the default)

1 Verbose output

2 "War and Peace", lots more output

3 Debug level output

The include file directory and define list <id> can contain names or

symbols delimited by ; for example:

--define=DEBUG;ALLOW\_SPACES

--include=source/tables;source/help;/users/me/includes

An example of the above would be:

asm65 myfile --listing=myfile --map=myfile --object=newprog --verbose=1

A full list of the parameters and their usage is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Short form | Long form | Action | Notes |
| -b <bn> | --debug=<bn> | Set the debug filename to <bn> | Not specifying a filename will cause the software to use the name sourcename.dbg. If the option is not used at all, no debug file is created |
| -c | --casesensitive | Make the processing of labels case sensitive | Labels are stored as uppercase by default. Using this flag allows case sensitive assembly to be used. Directives and opcodes are never case sensitive, irrespective of this flag |
| -d <list> | --define=<list> | Define a list of symbols | Valid use of <list> would be INC\_MONITOR or FLAG2;FLOPPY\_DRV;MASK\_ALL |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Appendix – Opcodes

The following opcodes are defined by the application:

| **Opcode** | **Z80** | **Z180** |
| --- | --- | --- |
| ACI |  |  |
| ADC | Y | Y |
| ADD | Y | Y |
| ADI |  |  |
| ANA |  |  |
| AND | Y | Y |
| ANI |  |  |
| BIT | Y | Y |
| CALL | Y | Y |
| CC |  |  |
| CCF | Y | Y |
| CM |  |  |
| CMA |  |  |
| CMC |  |  |
| CMP |  |  |
| CNC |  |  |
| CNZ |  |  |
| CP | Y | Y |
| CPD | Y | Y |
| CPDR | Y | Y |
| CPE |  |  |
| CPI | Y | Y |
| CPIR | Y | Y |
| CPL | Y | Y |
| CPO |  |  |
| CZ |  |  |
| DAA | Y | Y |
| DAD |  |  |
| DCR |  |  |
| DCX |  |  |
| DEC | Y | Y |
| DI | Y | Y |
| DJNZ | Y | Y |
| EI | Y | Y |
| EX | Y | Y |
| EXX | Y | Y |
| HALT | Y | Y |
| HLT |  |  |
| IM | Y | Y |
| IN | Y | Y |
| IN0 |  | Y |
| INC | Y | Y |
| IND | Y | Y |
| INDR | Y | Y |
| INI | Y | Y |
| INIR | Y | Y |
| INR |  |  |
| INX |  |  |
| JC |  |  |
| JM |  |  |
| JMP |  |  |
| JNC |  |  |
| JNZ |  |  |
| JP | Y | Y |
| JPE |  |  |
| JPO |  |  |
| JR | Y | Y |
| JZ |  |  |
| LD | Y | Y |
| LDA |  |  |
| LDAX |  |  |
| LDD | Y | Y |
| LDDR | Y | Y |
| LDI | Y | Y |
| LDIR | Y | Y |
| LHLD |  |  |
| LXI |  |  |
| MOV |  |  |
| MULT |  | Y |
| MVI |  |  |
| NEG | Y | Y |
| NOP | Y | Y |
| OR | Y | Y |
| ORA |  |  |
| ORI |  |  |
| OTD |  | Y |
| OTDM |  | Y |
| OTDMR |  | Y |
| OTDR | Y | Y |
| OTI |  | Y |
| OTIM |  | Y |
| OTIMR |  | Y |
| OTIR | Y | Y |
| OUT | Y | Y |
| OUT0 |  | Y |
| OUTD | Y |  |
| OUTI | Y |  |
| PCHL |  |  |
| POP | Y | Y |
| PUSH | Y | Y |
| RAL |  |  |
| RAR |  |  |
| RC |  |  |
| RES | Y | Y |
| RET | Y | Y |
| RETI | Y | Y |
| RETN | Y | Y |
| RIM |  |  |
| RL | Y | Y |
| RLA | Y | Y |
| RLC | Y | Y |
| RLCA | Y | Y |
| RLD | Y | Y |
| RM |  |  |
| RNC |  |  |
| RNZ |  |  |
| RP |  |  |
| RPE |  |  |
| RPO |  |  |
| RR | Y | Y |
| RRA | Y | Y |
| RRC | Y | Y |
| RRCA | Y | Y |
| RRD | Y | Y |
| RST | Y | Y |
| RZ |  |  |
| SBB |  |  |
| SBC | Y | Y |
| SBI |  |  |
| SCF | Y | Y |
| SET | Y | Y |
| SHLD |  |  |
| SIM |  |  |
| SLA | Y | Y |
| SLP |  | Y |
| SPHL |  |  |
| SRA | Y | Y |
| SRL | Y | Y |
| STA |  |  |
| STAX |  |  |
| STC |  |  |
| SUB | Y | Y |
| SUI |  |  |
| TST |  | Y |
| XCHG |  |  |
| XOR | Y | Y |
| XRA |  |  |
| XRI |  |  |
| XTHL |  |  |

# Index

.DEFINE, 15

.DEFMACRO, 8

.ELSE, 8

.ENDIF, 8

.ENDM, 8

.IF, 8

.IFDEF, 15

.IFNDEF, 15

.INCLUDE, 8

Boolean, 16

Comment, 11, 12

Comments, 5

Comparison operators, 16

Directive

.DB, 7

.DEFMACRO, 8

.DW, 8

.ELSE, 8

.ENDIF, 8

.ENDM, 8

.ERROR, 8

.IF, 8

.IFDEF, 8

.IFNDEF, 8

.INCLUDE, 8

.LIST, 8

.MESSAGE, 9

.NOLIST, 9

.ORG, 9

.WARNING, 9

Directives, 5, 7

Expression, 7, 8, 9, 15, 16, 17

Functions, 15, 16, 17

ASC, 16

BUILD, 17

CHR, 17

DATE, 17

HEX, 17

HIGH, 16

IIF, 15, 17

LEFT, 15, 17

LOW, 15, 17

LOWER, 17

MID, 17

POS, 15, 17

RIGHT, 17

STRING, 17

TIME, 18

UPPER, 18

VALUE, 17

VERSION, 18

Instructions, 5

Label, 11

Labels, 5

Literal, 15

Local prefix, 11

Macro, 8, 11

Null, 15

Opcode, 11

Opcodes, 5

Operand, 11

Operands, 5, 12

ORG, 15

Precedence, 16

String, 15, 16, 17

Symbol, 15, 16

Unary minus, 16

Unary plus, 16

1. For hex literals, and B/H suffixes these are not case sensitive [↑](#footnote-ref-1)
2. Hex literals using the H suffix must start with a digit. This is to avoid confusion with labels as FABH could be a hex literal or a label. In this instance, use 0FABH to make it clear to the assembler that this is a literal value [↑](#footnote-ref-2)