XA80
Assembler User Manual

Contents

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

1. Introduction

1.1. 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.

Although XA80 is a multi-grammar assembler, all rules and examples presented in this document relate to the default XA80 grammar. Other grammars will have different rules, for example, around the naming of labels and expression processing.

Please refer to the XA80 Grammar Editor User Manual for more information on these topics.

1.2. 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.

2. XA80 Line Structure

Main elements are:

- Labels
- Directives / Instructions
- Operands
- Comments

2.1. 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 may suffixed by a colon ':' to indicate a label. Examples are:

```
Start:
_loop_pos_3:
KX0001:
```

Labels are case sensitive unless the command line switch -C has been used to switch this off. A label can exist in isolation with no other commands, in which case the value of the program counter is assigned to the label.

2.1.1. 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
```

2.1.2. Directives / Instructions

Each code generating line will have a directive or an instruction. A directive could, for example, be DB (define bytes) which fills memory with a byte pattern or an instruction such as PUSH HL to push the HL register pair onto the stack.

The format is:

<label> or…
<optional_label> **<directive>** <expression(s)> or…
<mandatory_label> **<directive>** <expression(s)> or…
<optional_label> **<opcode>** <operand(s)>

2.1.3. Directives

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

Mandatory label applies to the EQU, SET and = directives, all others are optional. See section **Error! Reference source not found.** 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'
ASMERROR	exprStr	Raises the error message listed in the string expression and halts the assembly	ERROR 'Wrong!!!'
ASMWARNING	exprStr	Issues the warning represented by the string expression	WARNING "No setup"
DB	list	Defines a series of bytes in memory. The expression list is a comma separated list of 8 bit numeric expressions which can include single or multiple characters enclosed in quotes	DB 12, 0FAH, -12, 'H',"ello", 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 DB		
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	Synonym for DB		
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
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	INCLUDE "foo.inc"

		directive can be nested. By default, include files are not listed.	
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”
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

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

2.1.4. Instructions

Instructions are the elements which create the executable code, for example PUSH, LD, MVI. More detail is given in section 2.2 on opcodes and section 2.3 on operands.

2.2. Opcodes

Opcodes can be any one of the 8080 / 8085 / Z80 / Z180 opcodes, for example LD or RRCA. A full list of opcodes can be found in section 4.2

2.3. Operands

There are three different styles of operand used with instructions 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

2.3.1. Simple operands

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

A	E	NC
AF	H	NZ
AF'	HL	P
B	(HL)	PE
BC	I	PO
(BC)	IX	PSW
C	(IX)	R
(C)	IY	SP
D	(IY)	(SP)
DE	L	Z
(DE)	M	

Not all operands are available on all processor types, for example PSW is available on 8080/8085, (IX) is available on Z80/Z180.

2.3.2. Complex operands

Complex operands represent the index with offset operands, specifically:

```
(IX+signed_displacement)
(IY+signed_displacement)
```

2.3.3. 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 3.

2.3.4. Operand indirection

Indirection is indicated by the ( ) characters, however it is converted internally into [ ] characters. Some legal examples:

```
          LD    HL,(SAVED)      ; Get saved HL back
          OUT   (C),A           ; Send byte to port
          LD    A,(IX+4)        ; Get parameter byte
```

2.4. 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	* comment	A * at the start of a line will process all following characters as a comment

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
             :    :
```

3. 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)
IIF(p==0,”Zero”,”Non-zero”)
```

3.1. 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, 123D or 0
4. Hexadecimal numbers, which can be prefixed by #, 0x or suffixed by H[1]. For example $33A, $ff78 or 33AH[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

3.2. 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.

[1] For hex literals, and B/H suffixes these are not case sensitive

[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

3.3. 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 % modulo / remainder & 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

3.4. Integer Functions

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

Function	Detail
HIGH(expression)	Returns the high byte of an expression (bits 8 to 15)
IIF(expression,true_exp,false_exp)	If the expression is non-zero, trueexp is returned otherwise falseexp is returned
LENGTH(string)	Returns the length of a string in characters
LOW(expression)	Returns the low byte of an expression (bits 0 to 7)

3.5. String Functions

A number of string functions are available within XA80:

Function	Detail
DATE()	Return the date as a string in the form YYYY-MM-DD
IIF(expression,true_exp,false_exp)	If integer expression is non-zero, the string expression true_exp is returned otherwise the string expression false_exp is returned
LEFT(string,count)	Take the leftmost count characters from a string
LOWER(string)	Take the lower case value of string
MID(string,start,count)	Take the middle of a string from start for count characters
STRING(number)	Convert a number to a string value
RIGHT(string,count)	Take the rightmost count characters from a string
TIME()	Return the time as a string in the form HH:MM:SS
UPPER(string)	Return the upper case value of a string
VERSION()	Version string for the assembler

4. Appendices

4.1. Appendix - Command Line Usage

From the program startup when invoking XA80 with no parameters:

```
XA80 Cross Assembler V0.1
Copyright (C)2020-2022 Duncan Munro

Usage: xa80 filename <options>

Options:
    -b <bn> --debug=<bn>     Set the debug name to <bn>
    -c <cn> --com=<cn>       Set the .com file name to <cn>
    -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>
    -p <pt> --processor=<pt> Use the nominated processor, default Z80
    -r      --redistribution Info on redistributing this software
    -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
    -w      --warranty       Display warranty information
    -x <hn> --hex=<hn>       Set the hex filename to <hn>

<bn>/<cn>/<en>/<hn>/<ln>/<mn>/<on> default to the filename with ext
changed to .d80/.log/.hex/.lst/.map/.o80 respectively. Not specifying
<bn>, <cn>, <en>, <hn>, <ln>, <mn> or <on> 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

Processor type <pt> can be Z80 or Z180

The include file directory and define list <id> can contain names or
symbols delimited by ; for example:
    --define=DEBUG;TAB_SIZE=4;CODE_NAME="Project ASM"
    --include=source/tables;source/help;/users/me/includes
```

An example of the above would be:

```
xa80 myfile.z80 --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

4.2. Appendix – Opcodes

The following opcodes are defined by the application:

Opcode	8080	8085	Z80	Z180
ACI	Y	Y		
ADC	Y	Y	Y	Y
ADD	Y	Y	Y	Y
ADI	Y	Y		
ANA	Y	Y		
AND			Y	Y
ANI	Y	Y		
BIT			Y	Y
CALL	Y	Y	Y	Y
CC	Y	Y		
CCF			Y	Y
CM	Y	Y		
CMA	Y	Y		
CMC	Y	Y		
CMP	Y	Y		
CNC	Y	Y		
CNZ	Y	Y		
CP	Y	Y	Y	Y
CPD			Y	Y
CPDR			Y	Y
CPE	Y	Y		
CPI	Y	Y	Y	Y
CPIR			Y	Y
CPL			Y	Y
CPO	Y	Y		
CZ	Y	Y		
DAA	Y	Y	Y	Y
DAD	Y	Y		
DCR	Y	Y		
DCX	Y	Y		
DEC			Y	Y
DI	Y	Y	Y	Y
DJNZ			Y	Y
EI	Y	Y	Y	Y
EX			Y	Y
EXX			Y	Y
HALT			Y	Y
HLT	Y	Y		
IM			Y	Y
IN	Y	Y	Y	Y
IN0				Y

Opcode	8080	8085	Z80	Z180
INC			Y	Y
IND			Y	Y
INDR			Y	Y
INI			Y	Y
INIR			Y	Y
INR	Y	Y		
INX	Y	Y		
JC	Y	Y		
JM	Y	Y		
JMP	Y	Y		
JNC	Y	Y		
JNZ	Y	Y		
JP	Y	Y	Y	Y
JPE	Y	Y		
JPO	Y	Y		
JR			Y	Y
JZ	Y	Y		
LD			Y	Y
LDA	Y	Y		
LDAX	Y	Y		
LDD			Y	Y
LDDR			Y	Y
LDI			Y	Y
LDIR			Y	Y
LHLD	Y	Y		
LXI	Y	Y		
MOV	Y	Y		
MULT				Y
MVI	Y	Y		
NEG			Y	Y
NOP	Y	Y	Y	Y
OR			Y	Y
ORA	Y	Y		
ORI	Y	Y		
OTD				Y
OTDM				Y
OTDMR				Y
OTDR			Y	Y
OTI				Y
OTIM				Y
OTIMR				Y
OTIR			Y	Y
OUT	Y	Y	Y	Y
OUT0				Y
OUTD			Y	

Opcode	8080	8085	Z80	Z180
OUTI			Y	
PCHL	Y	Y		
POP	Y	Y	Y	Y
PUSH	Y	Y	Y	Y
RAL	Y	Y		
RAR	Y	Y		
RC	Y	Y		
RES			Y	Y
RET	Y	Y	Y	Y
RETI			Y	Y
RETN			Y	Y
RIM		Y		
RL			Y	Y
RLA			Y	Y
RLC	Y	Y	Y	Y
RLCA			Y	Y
RLD			Y	Y
RM	Y	Y		
RNC	Y	Y		
RNZ	Y	Y		
RP	Y	Y		
RPE	Y	Y		
RPO	Y	Y		
RR			Y	Y
RRA			Y	Y
RRC	Y	Y	Y	Y
RRCA			Y	Y
RRD			Y	Y
RST	Y	Y	Y	Y
RZ	Y	Y		
SBB	Y	Y		
SBC			Y	Y
SBI	Y	Y		
SCF			Y	Y
SET			Y	Y
SHLD	Y	Y		
SIM		Y		
SLA			Y	Y
SLP				Y
SPHL	Y	Y		
SRA			Y	Y
SRL			Y	Y
STA	Y	Y		
STAX	Y	Y		
STC	Y	Y		

Opcode	8080	8085	Z80	Z180
SUB	Y	Y	Y	Y
SUI	Y	Y		
TST				Y
XCHG	Y	Y		
XOR			Y	Y
XRA	Y	Y		
XRI	Y	Y		
XTHL	Y	Y		

5. Index