DANSK DATA ELEKTRONIK ApS

# ID-7000 ONEPASS ASSEMBLER
for the
ID-7000 MICROPROCESSOR SYSTEM
Users Manual

first edition,august 1976

written by Ole Lading.

1.Introduction.

The ID-7000 Onepass Assembler is a program used to translate user programs written in symbolic machine language (assembler language) for the 8080 microprocessor into binary machine code.

The assembling is performed in one pass. This means that the source text is read by the assembler only once. The assembler loads the user program directly into RAM-memory. This feature, combined with the fact that the assembler may be ROM-resident, gives a great advantage compared to traditional 2 (or 3) pass assemblers, especially when only a low speed input media is available.

The ID-7000 Onepass Assembler uses the standard ID-7000 input/output system (see ID-7000 Utility Routines manual). The assembler uses 3 logical devices:

1. READER device	for source input
2. LIST device	for list output
3. CONSOLE device	for initial communication and error messages.

The logical devices are connected to the physical devices by means of the I/O status byte in location 0 in memory. In this way different standard- and user defined I/O devices may be used by the assembler. For example, interactive asembling is possible by using the TTY as READER device.

The assembler generates no binary output, but loads the program directly into RAM-memory. If binary output is wanted, the binary dump facilities of the DEBUG/MONITOR program may be used.

The assembler uses 4kbyte of ROM memory from location C000 to CFFF (hex). Furthermore the assembler needs some RAM-memory for data storage. Locations 40 to FF (hex) are reserved for this purpose.

The assembler is planned and programmed by Claus Tøndering. It is debugged and has been subject to minor changes by Claus E. Christoffersen and Ole Lading.

2. Running the assembler.

The assmbler is started from the DEBUG/MONITOR program by executing an XJ command. The startaddress of the assembler is C000 (hex). The assembler should be run with the interrupt system disabled. Prior to starting the assembler by an C000<OXJ command, the source text should be placed in the wanted READER device.

When started, the assembler generates the following text on the TTY-CONSOLE device:

```
        ID-7000 ONEPASS ASSEMBLER
        DEFAULT SYMBOL TABLE AND I/O-UNITS ? (Y/N)
```

If default symboltable placement (location 100-1FF hex) and default I/O-units (TTY as READER,CONSOLE and LIST device) is wanted, the user writes a Y on the TTY-CONSOLE device and the assembler starts reading the source text from the TTY-READER or keyboard. In this case, the assembler can handle a maximum of 10 symbols. If any other character is written, the assembler responds:

```
        WRITE HEX  FIRST ADDRESS, LAST ADDRESS FOR SYMBOL TABLE  CR
```

The user must now write two hex numbers (max. 4digits) separated by a komma and terminated by a CR (carriage return). The symbol table must contain at least 50 (hex) locations for the predefined symbols. Beyond this every user specified symbol needs 8 locations in the symbol table.

After this the assembler responds:

```
        WRITE HEX IOSB CR
```

The user must now write a hex number (max. 2 digits) on the TTY-CONSOLE device. This number is then used as input/output status byte by the assembler, and the assembler starts reading the source text from the selected READER device.

3. The source language.

3.1 Syntax: The syntax of the source language is given below in BNF-notation. Readers not familiar with this notation should read the sample program in appendix 1 together with this section. The Character set used is the ASCII/ ISO character set.

```
(program)               ::=(statement)|(program)(statement)
(statement)             ::=(label)(spacempt)(opcode)(param)(comment)(crlf)
(spacempt)              ::=(spaces)|(empty)
(spaces)                ::=(space)|(spaces)(space)
(space)                 ::=ASCII space character
(empty)                 ::=
(label)                 ::=(empty)|(symbol):
(symbol)                ::=(letter)(letdig)
(letter)                ::=a|b ..... |z|A|B|.....|Z
(letdig)                ::=(empty)|(letter) 0 1 2 ..... 9
(opcode)                ::=(machine-instruction)|(assembler-directive)|(empty)
(machine-instruction)   ::=MVI|MOV|JMP|...|HLT
                        INTEL 8080 machine instruction mnemonics
(assembler-directive)   ::=ORG|END|EQU|DB|DW|DS
(param)                 ::=(empty)|(spaces)(firstop)(secondop)
(firstop)               ::=(op)
(secondop)              ::=(empty)|,(op)
(op)                    ::=(symbol)|(dec)|(hex)|(ascii)
(dec)                   ::= (digits)
(digits)                ::=(digit)|(digits)(digit)
(digit)                 ::=0|1|2|.....|9
(hex)                   ::=(digit)(hexdig)
(hexdig)                ::=(digit)|a|b|c|d|e|f|A|B|C|D|E|F
(ascii)                 ::='(string of ascii characters different from')'
(comment)               ::=(empty)|;(string of ascii characters)
(crlf)                  ::= CR LF |LF CR
```

Only the first five characters in a symbol are regarded, the rest is skipped. In opcodes only four characters are regarded. It should be noticed that the following symbols are predefined and reserved:

A, B, C, D, E, H, L, M, SP, and PSW

These symbols are names of the registers of the 8080.

3.2 Assembler directives: Beyond the 79 INTEL 8080 machine instructions, six assembler instructions (assembler directives) may be used as opcodes in a statement. The function of these instructions is discussed in this section.

3.2.1 ORG-directive. This directive is used to assign a value to the instruction counter (IC) of the assembler:

```
(label) ORG (dec)          or:
(label) ORG (hex)
```

The instruction counter points the current load address, and is counted up during assembling. The ORG-directive initiates the instruction counter to the value following the command. A possible label obtains the same value.

3.2.2 EQU-directive. This directive is used to give a symbol a specific value:

```
(symbol): EQU (dec)        or:
(symbol): EQU (hex)        or:
(symbol): EQU (ascii)
```

The symbol obtains the value specified by (dec),(hex) or (ascii). The value should be contained in a 16 bit word, otherwise an error message is given.

3.2.3 END-dirctive. The END-directive terminates the assembling:

```
(label) END (dec)          or:
(label) END (hex)          or:
(label) END (symbol)       or:
(label) END
```

In all cases the assembling is stopped, and a list of possible undefined symbols is given on the selected CONSOLE device (see section 4). If a parameter ((dec),(hex) or (symbol)) is present in the END statement, program execution is started in the specified address if the assembling has terminated without errors. Otherwise control is transferred to the DEBUG/MONITOR program. A label in the END statement has no effect. The (symbol) in the END-directive must contain 5 characters.

3.2.4 DB-directive. This directive defines byte(s) of data:

```
(label) DB (dec)            or:
(label) DB (hex)            or:
(label) DB (ascii)
```

In the first two cases a single byte containing the value given by (dec) or (hex) is allocated. Only the 8 least significant bits in the value are used. In the third case a set of consecutive memory locations containing the specified ASCII string is allocated. A possible (label) in the DB-statement obtains the value of the current instruction counter, i.e. the address of the (first) stored byte.

3.2.5 DW-directive. This dimctive defines two bytes of data:

```
(label) DW (dec)            or:
(label) DW (hex)            or:
(label) DW (ascii)          or:
(label) DW (symbol)
```

Two consecutive memory locations containing the specified 16 bit value are allocated. As shown in the last examble, the parameter may be a (possible still undefined) symbol. A label in the DB-statement obtains the value of the current instruction counter, i.e. the address of the first of the two bytes.

3.2.6 DS-directive. This directive reserves a set of consecutive memory locations as specified by (dec) or (hex) with undefined contents. A possible (label) in the DS-statement obtains the value corresponding to the startaddress of the reserved memory area.

```
(label) DS (dec)            or:
(label) DS (hex)
```

3.3 Restrictions in use of symbols. When a symbol is used as operand ((op)) in the parameter-field in a statement, the following rule must be observed:

A forward reference is only allowed if the machine instruction expects a 16 bit operand in the respective parameter-field.

ex: The following program segment is valid:

```
JMP alfa
LXI H,beta
...
...
alfa: MOV A,B
...
...
beta: DW OOFF
```

The JMP and the LXI instruction expect a 16 bit operand.

ex: The following program segment is not valid:

```
MVI A,beta
...
...
beta:EQU 7F
```

Here the MVI instruction expects an 8 bit operand. The program segment should be replaced by:

```
beta: EQU 7F
...
...
MVI A,beta
```

If an 8 bit parameter is undefined when used, an error message is given. (See section 4).

4. Error messages.

Whenever the assembler detects an error in the source language,an error-message is written on the selected CONSOLE device. When an error is detected the rest of the current statement is skipped and the assembler starts translation of the next statement. An appropiate amount of NOP's are placed in memory to make possible a later correction by use of the DEBUG/MONITOR program if a new assembling is not performed. Some errors cause an immediate jump to the DEBUG/MONITOR-program. Section 4.1 to 4.15 describes the different error messages.

4.1 NO INPUT: This message is given, when the high speed reader is specified as READER device, and this unit is not loaded before the assembler i started. The message is also given when an END-statement is not present on the paper tape. If TTY-reader is specified as READER device, this message is not given.A manual start of the papertape reader, or input of the source text from the keyboard,is then possible.

4.2 SYNTAX ERROR: This message is given, when the syntax of the source language is illegal.

4.3 SYMBOL TABLE OVERFLOW: This error message is given when the specified area for symbol table is used. After generating this error message, the assembler transfers control to the DEBUG/MONITOR program.

4.4 NO LABEL AT EQU-DIRECTIVE: This error message is given when a statement containing an EQU directive has no label.

4.5 TOO MANY PARAMETERS: This error message is given when a statement contents too many parameters to the specified opcode.

4.6 MISSING PARAMETERS: This error message is given when a statement contains too few parameters to the specified opcode.

4.7 UNKNOWN OPCODE: This error message is given when a statement contains an opcode which is not a known INTEL 8080 mnemonic or an assembler directive.

4.8 NUMBER OVERFLOW: This error message is given when a (dec) or (hex) constant is too big, i.e. can not be contained in a 16 bit integer.

4.9 ASCII STRING OVERFLOW: This error message is given when an ascii constant does not contain exactly two bytes. An exception to this rule is ascii constants in DB-statements. In this case the ascii string may have any length.

4.10 DOUBLE DEFINED SYMBOL: This error message is given when a symbol is defined twice.

4.11 FORWARD REFERENCE NOT 16 BIT. This error message is given when a not defined symbol is used as parameter where an 8 bit parameter is expected by the opcode (see section 3.3).

4.12 ILLEGAL RECORD LENGTH: This error message is given when a statement in the source text is too long. Only 50 characters in a statement are processed. The rest is skipped. After this error message, the assembler starts processing the first 50 characters in the normal way.

4.13 ILLEGAL ARGUMENT IN ASSEMBLER DIRECTIVE: This error message is given when an illegal argument is found in an assembler directive. (See section 3.2).

4.14 START ADDRESS NOT FOUND: This message is given when an END statement contains a symbol in the parameter field, and this symbol is undefined. After this error message, control is transferred to the DEBUG/MONITOR program.

4.15 ..... IS AN UNDEFINED SYMBOL: When the assembling is terminated by an END-statement, the assembler writes a list of undefined symbols in this format.If an undefined symbol is present, the assembler transfers control to the DEBUG/MONITOR program, although otherwise specified by the END-statement.

5. Physical Record Format.

When the TTY-reader is specified as READER device, this reader is started at the beginning of each record by an ASCII XON character transmitted to the device. By the end of the record, it is stopped by an XOFF character. In this way it is not possible to stop the TTY--reader immediately. The records must be separeted by at least 4 blind characters following the CR-LF (LF-CR) sequence to secure correct operation. As record separation characters may be used NUL, DELETE and space. When the source text is generated by the ID-7000 TEXT-EDITOR, the records are automatically separated by NUL characters.

When high speed reader is specified as READER device, records must be separated by at leat one blind character.

The record may contend both small and capital letters. In symbols and opcodes no destinction is made between small and capital letters. The notations AlFa and alfa describes the same symbol. In ASCII strings the correct values of the letters (small or capital) are stored.

The characters NUL and DELETE are blind characters to the assembler, and may be present anywhere in the source text.

APPENDIX Ï

```
C000<0XJ


ID-7000 ONEPASS ASSEMBLER
DEFAULT SYMBOL TABLE AND I/O-UNITS ? (Y/N) Y
0001 1700                                   ;SAMPLE ID-7000 PROGRAM.
0002 1700                                   ;THE PROGRAM WRITES A
0003 1700                                   ;TEXT ON THE TTY-CONSOLE
0004 1700                                   ;DEVICE FROM A TEXTBUFFER
0005 1700
0006 1700
0007 1700                                   ;THE WRITING TERMINATES
0008 1700                                   ;WHEN AN X.FF IS READ FROM
0009 1700                                   ;THE TEXTBUFFER.
0010 1700
0011 1700                                   ;THE PROGRAM USES THE
0012 1700                                   ;ID-7000 UTILITY-ROUTINES
0013 1700                                   ;FOR I/O
0014 1700
0015 1700         ORG     700               ;PROGRAM START ADDR.
0016 0700
0017 0700 ISET:   EQU     0EEFB             ;ADDRESS FOR SET IOSB RT
0018 0700 CO:     EQU     0EEAA             ;CONSOLE OUTPUT ROUTINE
0019 0700 STACK:  EQU     800               ;STARTADDR.+1 FOR STACK
0020 0700 IOSB:   EQU     0                 ;IOSB FOR TTY
0021 0700
0022 0700
0023 0700 START:  LXI     SP,STACK
0024 0703         MVI     C,IOSB
0025 0705         CALL    ISET              ;LOAD IOSB
0026 0708         LXI     H,BUFF            ;HL=BUFFER START ADDR.
0027 070B WRITE:  MOV     A,M               ;GET CHARACTER
0028 070C         CPI     0FF               ;X.FF?
0029 070E         CZ      0F008             ;YES,RETURN TO DEBUG
0030 0711         MOV     C,A
0031 0712         CALL    CO                ;WRITE CHAR ON CONSOLE
0032 0715         INX     H                 ;HL:=HL+1
0033 0716         JMP     WRITE             ;
0034 0719
0035 0719 BUFF:   DB      'CONGRATULATIONS! YOU HAVE '
0036 0733         DB      'ASSEMBLED YOUR '
0037 0742         DW      0D0A      ;CR LF
0038 0744         DW      0         ;2 NUL CHAR'S
0039 0746         DB      'FIRST ID-7000 PROGRAM'
0040 075B         DB      0FF
0041 075C         END     START             ;END VECTOR TO START
CONGRATULATIONS! YOU HAVE ASSEMBLED YOUR
FIRST ID-7000 PROGRAM

>
```