CALM
Common Assembly Language for Microprocessors

user manual

Patrick Fäh, La Colombière, CH-1783 Pensier, Suisse

Contents

CALM files

You should have the following files on your system (assuming thatyou have an assembler for the Z80 processor):

ASCALM.*	CALM assembler
MUFBIN.*	converts generated binary format (MUFOM)
Z80.PRO	processor module for the microprocessor Z800
TZ80.ASM	test file with Z80 instructions
ASCALMER.TXT	lfile with errors (used by the assembler)

Minimal configuration :

Atari ST:	680x0, 256 KByte free, 1 floppy drive
PC/MS-DOS >= 2.x:	iAPX86 comp. processor, 256 KByte, 1 floppy drive.
PC/MS-DOS:	only: set in CONFIG.SYS in minimum FILES=16.
DOS, TOS:	ASCALMER.TXT, *.PRO, and *.REF: actual PATH is used with SET CALM=path, one can define another path.Command line

Ligne de commande

To assemble a CALM file, enter on your system: ASCALM <source file> [/switch]

The switches are optional. The following switches are available:

/Apath1;path2; ASCALM defines additional paths for ASCALM, for the,files: ASCALMER.TXT, *.PRO and *.REF. Example:
DOS, TOS: PATH C:\ASCALM;C:\BIN;
command line: /AA:\PRODEF;
instruction: .PROC Z80
tries to open:
1) Z80.PRO
2) A:\PRODEF\Z80.PRO
3) C:\ASCALM\Z80.PRO
4) C:\BIN\Z80.PRO
Same for ASCALMER.TXT and all *.REF. Limit length for all paths is 80 characters. With .PROC A:Z80 ASCALM tries only A:Z80.PRO.

fichier/B .MUF generates a binary (MUFOM format)

fichier/C .ASC generates file.ASC: all macros and .IF/.ELSE/.ENDIF are replaced. Not possible if /E is used.

/Dnom=valeur def. a symbol with name and value (+ or -, decimal).

fichier/E .ERR copies all error messages also in this file.

/F does not remove unused symboles loaded by .REF.

/Ipath1;path2; Insert defines additional paths for .INS. Example:
command line: /IA:\SOURCE;B:\DEF\;C:\PROJET;
instruction: .INS IO_PART
tries to open: 1) A:\SOURCE\IO_PART.ASM
2) B:\DEF\IO_PART.ASM
3) C:\PROJET\IO_PART.ASM
4) IO_PART.ASM
Limit length for all paths: 80 characters. With .INS B:IO_PART ASCALM tries

	only B:IO_PART.ASM.
fichier/L .LST	generate a listing
/R	Read source files will not be modified
fichier/S .REF	contains all used symbols as assignments.
fichier/SA... .REF	as /S but add. are with ":"; also lists TRUE, etc.
fichier/Si... .REF	do not write some symbols in the .REF file: without values (i=0), addresses (1), == (2), or .SYSCALL (3).
/V	Verify generates no binary (inhibit /B)
/W	Wait waits for every error line
fichier/X .XRF	generates the cross-ref file appended to the listing

If you do not use a switch, the assembler generates only a binary file. The switch /B is only necessary when the binary .MUF should have another name or be located on another drive. The source file is any file generated by PFED or any similar editor.
Each line should not exceed 128 characters in length and is terminated by <CR><LF>.

Let us test the assembler and its module by assembling the corresponding test file generating a listing and a binary.

```
ASCALM TZ80/L
```

This command will take the source file TZ80.ASM and generate the binary TZ80.MUF and the listing TZ80.LST. To assemble a file in drive B: and to copy the binary and the listing on drive A: enter:

```
ASCALM B:TZ80 A:TZ80/L/B      ou
ASCALM B:TZ80 A:TZ80/L A:TZ80/B
```

Theory of operation

The assembler is started by the command: ASCALM <source>/L with the request of a listing. The assembler opens the source file and creates the binary (.MUF) and listing (.LST) file. If any errors are found in the program, the error line with the error indication is given in the listing, in the error file (if /E), on the screen, and in the source file (if not /R).

```
MOVE    B,ALPHA         ; source line of Z80 program
          ^ 31
```

When the assembler starts, he looks for the file ASCALMER.TXT. If this file is not found, the errors are shown with numbers. Otherwise the errors are shown in text form:

```
MOVE    B,ALPHA         ; source line of Z80 program
          ^ symbol value undefined
```

The following keys may be entered if a error message appears at the screen:

"D" don't wait after an error

"S" stop the assembly (always possible)

"W" wait after an error

If any fatal error occurs (file does not exist, disk full, etc.) the assembler halts and shows some information on the screen (ASCALM returns 4 to the system, else 0). Do the necessary corrections and restart the assembler again.

The I/O files like CON:, AUX:, etc., are supported.

To show the listing on the screen, and without binary, you must enter: ASCALM TZ80 CON:/L/V

Important: The assembler makes a copy of the source file to insert the error messages. Therefore, make sure there is enough disk space! This copy (.AST) with the error messages will then replace the original file. With the option /R, the assembler will not modify the source file(s). This copy is also done for inserted files (.INS).

Characters in the file names: "0".."9","A".."Z","a".."z","_","?", "-",":","\",".".

Cross reference list

To obtain the cross reference list of a program, you must use the switch /X:
ASCALM TZ80 B:TZ80/X

The assembler generates the file B:TZ80.XRF. Be careful of the size of this file, if you use a lot of symbols in your program. The cross reference list is written (added if /L) to the listing TZ80.LST. The temporary file TZ80.XRF is deleted.

Object format

The CALM assembler always generates object files with the extension .MUF. This binary format is an ASCII format. Only absolute formats are generated. To convert the MUFOM format to for example .COM you must use the program MUFBIN:

MUFBIN <objet.MUF>/options

MUFBIN limits the binary output length to actually 64 Kbytes (exception: no limitation with /B/N, /H/N, /I/U, /M/U (/N: when data is consecutive).

he following options determine the output format (no default):

/B	.BIN binary, also /N or /Y must be indicated
i/E	.EXE for PC/MS-DOS, i (1..3) defines the memory structure
/H	.BIN hexadécimal, ASCII, il faut aussi indiquer /N ou /Y
/I	. HEX , fIntel's hex format
/M ou i/M	.FRS, Motorola's S format (default: S0, S3 and S7; 1/M: S0, S1 and S9; 2/M: S0, S2 and S8; 3/M: S0, S3 and S7).
/T	.TOS, for Atari ST

With the options /B and /H you can add a header in the front. These files are composed of a header of 256 bytes followed by the image of the binary. The header contains the following information:

offset.	content [determined by]
0	load address [lowest .LOC with generated code]
2	length [code]
4	start address [.START start]

The byte order for the three 16 bit numbers is low-high. If you don't want to add the header, enter /N. The generated binary file is compatible to the .COM file, if the start and the load address is hexadecimal 100. With /Y you can automatically add the header.

A	shifts the object by the specified value (hexadecimal_value/A).
/D	data size (used with /E; hexadecimal_value/D).
/F	performs an AND-operation between the indicated filter value and the addresses (hexadecimal_value/F).
/J	hh/J: val. for undef. byt., per def.: 00/J. Use FF/J for EPROM.
/L	fixes the number of data bytes in a line with /H, /I and /M (range: 1 to 250; default values: 39, 32, 32; value/L). The command /H 0/L will generate no <CR><LF>.
/O	changes the name of the output file (filename/O).
/S	stack size (used with /E; hexadecimal_value/S).
/U	undefined areas are not filled (used with /I or /M).
/V	shows all inform. (except data) contained in the MUFOM file.

/W word swap: exchanges LSB and MSB in a 16-bit word.

Examples:

- change the output file name (generating Intel format): MUFBIN input/I output/O I/O files (like CON:, AUX:) are possible for output.
- the 2 add. ranges 16'0 to 16'FFF and 16'F000 to 16'FFFF have to be placed in one physical 8 KByte EPROM. The command MUFBIN input/B/N 1FFF/F will gen. an 8 KByte output bin. file, which can be used directly to prog. the EPROM. Else the output file would be 64 KByte long.
- shift the object by the specified value: MUFBIN input/I 200/A If the file has been ass. with a .LOC 16'0, MUFBIN will gen. An Intel hex comp. output file which starts with 16'200. The object code itself is not mod. If also /F has been spec., the filter operation is performed first.

Pseudo-instructions

Only the following pseudo-instructions are supported by the CALM Assembler: .ALIGN, .APC, .ASCII, .ASCIZ, .ASCIZE, .BASE, .BLK.n, .CHAP, .DATA.n, .ELSE, .END, .ENDIF, .ENDLIST, .ENDMACRO, .ERROR, .EVEN, .EXITMACRO, .FILL.n, .IF, .INS, .LAYOUT, .LAYOUTMACRO, .LIST, .LISTIF, .LOC, .LOCALMACRO, .MACRO, .MESSAGE, .ODD, .PAGE, .PROC, .PROCSET, .PROCVAL, .RANGE, .REF, .START, .STRING, .SYSCALL, .TITLE, .8, .16 und .32 .

Remarks to some pseudo-instructions (se also UPDATESE.*):

.ASCIZE
- corresponds to .ASCIZ followed by .EVEN (generates 0).

.IF/.ELSE/.ENDIF
- IF <expression> is true, when <expression> <> zero.
- IF..ENDIF may be nested up to 32 times.
- IF and the correspondant ENDIF must be in the same file.
- IF..ELSE..ELSE..ENDIF is possible.

.INS
- with .INS file,READONLY the inserted file is only read. Error messages will not appear in this file. They are reported in the main file (if possible) or the error file (if /E).
- default file extension: .ASM.
- only one nested level is allowed.

.LAYOUT
- With .LAYOUT the following parameters are possible:

HEX (addresses and data in hexadecimal representation)
LENGTH n (n lines per listing page, n=0: infinite)
Example: .LAYOUT HEX, LENGTH 60 ; values of the assembler

- Following values are always fixed:

HEX (OCT is not possible)
WIDTH 127 (line length)
TAB 8 (one tabulator corresponds to 8 spaces)

.LIST/.ENDLIST

- LIST <expression> is true when <expression> <> zero.
- LIST..ENDLIST may be nested up to 255 times.
- LIST and the correspondant ENDLIST must be in the same file.

.LISTIF <expression>

- .LISTIF shows all .IF/.ELSE/.ENDIF pseudo-instr. in the listing.
- .LISTIF is active when <expression> is <> zero or <expression> is not present.

.REF fichier

- file.REF is a text file and may contain assignments, .SYSCALLs, and comments. The file is read once and is never modified.
- the PATH is respected.

.SYSCALL.n nom (n = 8, 16 ou 32)

- defines a special macro: .MACRO name; .n name%1; .ENDMACRO. The SYSCALLs are allowed in .REF files. Example: INTDOS = 16'CD21;
.SYSCALL.16 INT; call: INT DOS; generates: .16 INTDOS.

The following pseudo-instructions are not supported: .ENDTEXT,.EXPORT, .IMPORT, .TEXT

Differences assembler - CALM standard

The CALM Assembler does not support the full CALM Standard. The differences:

symbole:

- name: 32 (local labels: 29) signs are significant. characters: "A".."Z", "a".."z", "_", "?" and "0".."9" (<> 1. position). Accent letters are converted to upper case letters.
- value: 32 bit with sign.

expression:

- word length: 32 bit with sign.
the maximum number of open operations is 15.
- shift amplitude (.SR., .SL. und .ASR.): shift amplitude is limited to 8 bits (-256..+255). A negative amplitude inverts the shift direction.

général:

- maximum length of a input assembly line is 127.
- the APC has a length of 32 bits.
- some pseudo-instructions are not supported.
- the \-commands are not supported.
- .PROC: only one times allowed

Assembler errors

Refer to ASCALMER.TXT. ASCALMEE.TXT contains the error messages in english. You may copy this file in ASCALMER.TXT.

Quelques remarques concernant les erreurs fatales:

101 .PROC error
(Something in the file.PRO is not correct. Try again with the /D switch to localise the instr., which causes this error.)

102 .PROC
too long (Not enough memory)

103 cannot load file
(The assembler can't find the specified file in .PROC or .INS.

104 cannot open source
(The assembler can't find the source file.)

105 error to create file
(The assembler can't create the *.MUF or/and *.LST file.)

106 no .PROC
The assembler tries to assemble an instruction, but no .PROC has been loaded: put .PROC in the front of your file.)

107 cannot reset source
(The assembler resets the source before the second pass. Check your system.)

108 bad .PROC version
(Your file.PRO has a bad version and can't work correctly with the assembler.)

109 command line is empty
(You must give the file to assemble directly on the command line.)

110 new symbol in 2nd pass
(A new symbol appeared in the 2nd pass; if you are working in a network environnement try again; localise the line with the /D switch.)

111 over-/underflow in .PRO
(Something wrong with the interpretation of the .PRO file.)

112 symbol table overflow

113 end of file: missing .ENDMACRO

114 macro buffer overflow

115 too many nested .INS

116 stopped
(The assembler has been stopped by the key "S".)

117 .PROC/.REF: must precede code generation
(Put .PROC and .REF in the beginning of the source; after .TITLE.)

Inline with the CALM assembler

TurboPascal and Pascal/MT+ feature the INLINE statements as a very convenient way of inserting machine code instructions directly into the program text. Refer to the corresponding Pascal user manuals for more details about INLINE syntax and limitations.

The CALM assembler can be used to generate the corresponding machine code. The following example shows the steps to generate INLINE statements for the iAPX86 (PC/MS-DOS) and the Z80 (CP/M-80).

The function HEXNIBBLE tests and converts any character to an integer if the character is a valid hex number ('0'..'9','A'..'F','a'..'f'):

```
FUNCTION HEXNIBBLE(VAR H:INTEGER):BOOLEAN;
{in: ActCh, out: H (value), HEXNIBBLE (true or false)}
VAR C:CHAR;
  BEGIN
    C:=UpCase(ActCh); HEXNIBBLE:=TRUE;
    IF (C >= '0') AND (C <= '9')
    THEN BEGIN
      H:=ORD(C)-ORD('0');
    END
    ELSE
      IF (C >= 'A') AND (C <= 'F')
      THEN BEGIN
        H:=ORD(C)-ORD('A')+10;
      END
      ELSE BEGIN
        H:=0; HEXNIBBLE:=FALSE;
      END;
  END;
```

Now you must convert the Pascal procedure to assembler. Thefollowing two pages show the iAPX86 and Z80 versions of the functionHEXNIBBLE. For this, firstly you write the assembler instructions innormal assembler form in an assembler source file and assemble thisfile with a listing. Then, you delete the Pascal source lines betweenBEGIN and END in the function HEXNIBBLE. You insert the assemblerlisting file after BEGIN. You delete all dummy assembler listing linesand also the addresses (4 characters in front of each line). Now youput the generated listing bytes in the INLINE form: begin withINLINE(, add $ and / between the listing bytes, etc. Put theinstructions in Pascal comment form (with (* and *)). Finally, youmust replace all variable references by their correct name (RESULT,HEX).

It is important to know exactly the internal representation of thedifferent data types. Knowing this, you can significantly improve theassembler version (speed, code size). For example, the boolean valuehere is 1 for TRUE and 0 for FALSE. Also you must know how to accessthe different variables. The correspondent Pascal user manuals giveyou more information.

To test the Pascal/Assembler performance, the following smallprogram has been used:

```
PROGRAM THEX;
VAR {teste la version en assembleur et en Pascal de HEXNIBBLE}
  ACTCH: CHAR;
  RESULT: BOOLEAN;
  I, VALUE: INTEGER;
{$I P_HEX} { P_HEX: Pascal, A_HEX: assembleur }
BEGIN
  WRITELN('START');
  FOR I:=1 TO 1000 DO  {1000 x}
  BEGIN
    FOR ACTCH:=' ' TO '~' DO  {95 caractères}
     RESULT:=HEXNIBBLE(VALUE);
    END;
  WRITELN('END');
END.
```

The following results have been obtained:

	code size	seconds (for THEX)
Pascal version	192 octets	26,3
Assembler version	80 octets	20,6
Difference	-58 %	-22 %

(XT-compatible, clock 4.77 MHz, TurboPascal 3.0 for PC-DOS)

These numbers should give you a general idea. Very often, theassembler version is significantly faster (2..4) than the Pascalversion. In addition, the code size reduction is always impressive.The iAPX86 (PC/MS-DOS) version (Listing):

Version iAPX86 (PC/MS-DOS), Listing:

```
0000                            .TITLE  HEXNIBBLE
0000                            .PROC   IAPX86
2710     00002710               .LOC    10000
2710                    RESULT:                 ; address > 8 bit:
2710                    HEX:                ; assembler gets 16 bit
0000     00000000               .LOC    0
0000                    HEXNIBBLE:
0000   8A861027                 MOVE.8  [SS]+{BP}+RESULT,AL
0004   31C9                     XOR.16  CX,CX   ; CL = FALSE (=0),
0006   2C30                     SUB.8   #"0",AL ; CH = VALUE
0008   7211                     JUMP,LO END$
000A   3C09                     COMP.8  #9,AL
000C   760A                     JUMP,LS OK$
000E   2C07                     SUB.8   #"A"-"0"-10,AL
0010   3C0A                     COMP.8  #10,AL
0012   7207                     JUMP,LO END$
0014   3C0F                     COMP.8  #15,AL
0016   7703                     JUMP,HI END$
0018   88C5             OK$:    MOVE.8  AL,CH   ; nibble
001A   41                       INC.16  CX      ; CL = TRUE (=1)
001B   88AE1027         END$:   MOVE.8  CH,[SS]+{BP}+RESULT
001F   888E1027                 MOVE.8  CL,[SS]+{BP}+HEX
```

Procédure:

```
FUNCTION HEXNIBBLE(VAR H:INTEGER):BOOLEAN;
{in: ActCh, out: H (value), HEXNIBBLE (true or false)}
VAR RESULT:INTEGER; HEX:BOOLEAN;
BEGIN
  RESULT:=ORD(UpCase(ActCh));
  INLINE(
  $8A/$86/RESULT/ (*        MOVE.8  [SS]+{BP}+RESULT,AL          *)
  $31/$C9/        (*        XOR.16  CX,CX;CL=FALSE (=0), CH=VALUE*)
  $2C/$30/        (*        SUB.8   #"0",AL                      *)
  $72/$11/        (*        JUMP,LO END$                         *)
  $3C/$09/        (*        COMP.8  #9,AL                        *)
  $76/$0A/        (*        JUMP,LS OK$                          *)
  $2C/$07/        (*        SUB.8   #"A"-"0"-10,AL               *)
  $3C/$0A/        (*        COMP.8  #10,AL                       *)
  $72/$07/        (*        JUMP,LO END$                         *)
  $3C/$0F/        (*        COMP.8  #15,AL                       *)
  $77/$03/        (*        JUMP,HI END$                         *)
  $88/$C5/        (*OK$:    MOVE.8  AL,CH   ; nibble             *)
  $41/            (*        INC.16  CX      ; CL = TRUE (=1)     *)
  $88/$AE/RESULT/ (*END$:   MOVE.8  CH,[SS]+{BP}+RESULT          *)
  $88/$8E/HEX);   (*        MOVE.8  CL,[SS]+{BP}+HEX             *)
  HEXNIBBLE:=HEX; H:=RESULT;
END;
```

Version Z80 (CP/M-80), Listing:

```
0000                            .TITLE  HEXNIBBLE
0000                            .PROC   Z80
2710     00002710               .LOC    10000
2710                    RESULT:
2710                    HEX:
0000     00000000               .LOC    0
0000                    HEXNIBBLE:
0000   3A1027                   MOVE    RESULT,A
0003   010000                   MOVE    #0,BC   ; C = FALSE (=0),
0006   D630                     SUB     #"0",A  ; B = VALUE
0008   3810                     JUMP,LO R8^END$
000A   FE0A                     COMP    #9+1,A
000C   380A                     JUMP,LO R8^OK$
000E   D607                     SUB     #"A"-"0"-10,A
0010   FE0A                     COMP    #10,A
0012   3806                     JUMP,LO R8^END$
0014   FE10                     COMP    #15+1,A
0016   3002                     JUMP,HS R8^END$
0018   47               OK$:    MOVE    A,B     ; nibble value
0019   0C                       INC     C       ; TRUE (=1)
001A   78               END$:   MOVE    B,A
001B   321027                   MOVE    A,RESULT
001E   79                       MOVE    C,A
001F   321027                   MOVE    A,HEX
```

Procédure:

```
FUNCTION HEXNIBBLE(VAR H:INTEGER):BOOLEAN;
{in: ActCh, out: H (value), HEXNIBBLE (true or false)}
VAR RESULT:INTEGER; HEX:BOOLEAN;
BEGIN
  RESULT:=ORD(UpCase(ActCh));
  INLINE(
  $3A/RESULT/     (*        MOVE    RESULT,A                      *)
  $01/$00/$00/    (*        MOVE    #0,BC  ;C=FALSE (=0), B=VALUE*)
  $D6/$30/        (*        SUB     #"0",A                        *)
  $38/$10/        (*        JUMP,LO R8^END$                       *)
  $FE/$0A/        (*        COMP    #9+1,A                        *)
  $38/$0A/        (*        JUMP,LO R8^OK$                        *)
  $D6/$07/        (*        SUB     #"A"-"0"-10,A                 *)
  $FE/$0A/        (*        COMP    #10,A                         *)
  $38/$06/        (*        JUMP,LO R8^END$                       *)
  $FE/$10/        (*        COMP    #15+1,A                       *)
  $30/$02/        (*        JUMP,HS R8^END$                       *)
  $47/            (*OK$:    MOVE    A,B     ; nibble value       *)
  $0C/            (*        INC     C       ; TRUE (=1)          *)
  $78/            (*END$:   MOVE    B,A                           *)
  $32/RESULT/     (*        MOVE    A,RESULT                      *)
  $79/            (*        MOVE    C,A                           *)
  $32/HEX);       (*        MOVE    A,HEX                         *)
  HEXNIBBLE:=HEX; H:=RESULT;
END;
```

Generating .EXE programs with the CALM assembler

The PC/MS-DOS operating system uses two types of programs: .COM and.EXE. The .COM programs are residues from CP/M-80. Program, data andstack segments reside in a total memory space of 64K. Programexecution starts at 16'100. Therefore, all four segment registers ofthe iAPX86, CS, DS, ES and SS, have the same value, and they all pointto the beginning of a 64 K segment. However, when a .COM program isstarted, all the free memory is used - not only the 64K.

The .EXE programs are more complicated. They have a header, whichcontains information on the program length, the starting values of theprogram counter (CS:IP) and the stack pointer (SS:SP), etc. A .EXEprogram may have separate program, data and stack segments. So thelimit of 64 K is no longer valid. An .EXE program only occupies thememory space needed.

It is possible to generate .EXE programs with the CALM assembler.To do this, the programmer must know where the segments (program,data, stack) are and how to initialize and access them. However, thecode segment length is limited to 64 K (without tricks). The stack anddata segments may also have 64 K each (but any length with somemanipulations).

Writing .EXE programs with the CALM assembler needs some care. Itis not possible to load a segment register (DS or ES) with a constant(for example with a label via AX), because the program would belocated at another (unknown) address. Relocation is done by the user.He correctly initializes the segment registers and uses them as "basepointers". Note, that this programming style is possible with anyassembler. Many .EXE files in the PC/MS-DOS system do not needrelocation before starting up (because they have no relocation entriesin their .EXE header).

To generate .EXE programs, you need the CALM assembler (ASCALM andMUFBIN) and the processor module iAPX86 (or iAPX186, iAPX286). TheCALM assembler is also used to generate .COM programs. In .COMprograms, you will never modify the segment registers. The programwill start with .LOC 16'100. In .EXE programs, you MUST initialize thedata segment. Before execution, the system initializes the segments CS(code) and SS (stack). The segments DS and ES (data) point to the PSP(program segment prefix). The program code always starts in theassembler source file with .LOC 0. In addition, you can specify anystart address with .START. You can assemble the source file andconvert the generated object with MUFBIN file i/E to the specific.EXE binary file. However, the stack and data sizes are not known andthe user must enter these values. MUFBIN supports actually three .EXEmemory structures. These three possibilities depends on yourprogramming choice. You will find three corresponding examples(TESTEXE1/2/3.ASM) joined to the module iAPX86. In these examples youwill also find more explanations.

The command line to start MUFBIN is:

MUFBIN input_file{.MUF} i/E {stack_size/S} {data_size/D}Text within braces are optional. /i chooses one of the three .EXEmemory structures. /S and /D fix the stack and data sizes (ifrequired). Example (case 2):

```
MUFBIN TESTEXE2 2/E 80/S 104/D
```

Using local labels

Local labels (for example LOOP$) are not really different fromglobal labels (for example START). However, local labels can not befound in the cross reference list because their significance is onlylocal. In addition, local labels are valid (may be used) only betweentwo global labels. For these reasons, local labels are preferably usedin subroutines where they are the marker points for loops, conditionsand exit. Example:

```
TEXTHL:
PRINTC$ =       2               ; local assignment
          PUSH    HL              ; in HL ^.ASCIZ
LOOP$:
          MOVE    {HL},A
          OR      A,A
          JUMP,EQ R8^END$         ; <NULL> reached ?
          PUSH    HL
          MOVE    A,E
          MOVE    #PRINTC$,C      ; show character on the screen
          PUSH    HL
          CALL    BDOS
          POP     HL
          INC     HL
          JUMP    LOOP$
END$:
          POP     HL
          RET
```

LOOP$ and END$ may also be used in other subroutines. Therefore,one must not always invent new names like LOOP1, LOOP2, etc. Inaddition, only TEXTHL (the name and the entry point of the subroutine)appears in the cross reference list.JUMP instructions optimally coded

There exist two addressing possibilities for jump instructions inmany 8 bit microprocessors:

instructions optimally coded

There exist two addressing possibilities for jump instructions inmany 8 bit microprocessors:

1) JUMP R8^étiquette (adr. relatif, APC-128..APC+127, 2 octets)
2) JUMP 16^étiquette (adr. absolu, 0..16'FFFF, 3 octets)

If the address specifiers R8^ and 16^ are not present, the CALMassembler chooses automatically either case 1) or 2). But when is case 1) generated?
Case 1) has two advantages compared with case 2):relative addressing (address independent) and shorter machine code (2instead of 3 bytes).

Let us consider the following situation:

```
          ...
BEFORE:
          ...
          JUMP   BEFORE       ; (a)
          ...
          JUMP   AFTER        ; (b)
          ...
AFTER:
```

We see, that JUMP BEFORE (a) jumps to a label, which is definedbefore the jump instruction, and that JUMP AFTER (b) jumps to alabel which is defined after the jump instruction.

In the case of (a), the CALM assembler may generate the relativeaddressing. For this to happen, the offset, from the APC of the JUMPBEFORE instruction to the label BEFORE, must be less than 129. If theoffset is greater, the CALM assembler will choose the absoluteaddressing.

The CALM assembler chooses in (b) always the absolute addressing,even when AFTER is less than 127 Bytes from (b) away. This is becauseof the following reason: The assembler reads a source program twice(two passes) to generate the machine program. In the first pass, theCALM assembler meets in (b) the unknown label AFTER. As the CALMassembler does not know the value, the worst case is taken (that meansthe label AFTER is far away) and the CALM assembler generates anabsolute jump instruction (3 bytes). The first pass is only used toevaluate the correct addresses. In the second pass, the CALM assemblermeets in (b) again the label AFTER, now known. In some cases, theoffset is really less than 128, or in other words, a relative 8 bitaddressing would have been possible. But now this is impossible: Ifthe CALM assembler were to choose here a relative jump instruction (2bytes), then all the following addresses would be moved by one byte!Therefore, the CALM assembler must choose in (b) the same instructionas in the first pass, that means, an absolute jump instruction.

To force case 1) or case 2), one must insert the address specifierR8^ or 16^. The same thing is also valid for other instructionsoptimally coded.Extensions of the CALM assembler

Extensions of the CALM assembler

The CALM assembler is a macro assembler, that is, also macros aretreated.

Multiple definitions in assignments and labels are allowed if thesymbols have the same name and the same value (CR = 13; CR = 16'D).

Local assignments (A$ = 10) are allowed and have the samepossibilities as the local symbols (A$:).

Multiple assignments (A == 10) are possible and the value may bemodified (A == 20). Corresponds to SET on other assemblers. Usualassignments (A = 10) can not be mixed with multiple assignments (A == 20).

The base number may be indicated by letters:

```
16'nnnn  H'nnnn  X'nnnn  H'AF
10'nnnn  D'nnnn          D'100
 8'nnnn  O'nnnn  Q'nnnn  Q'377
 2'nnnn  B'nnnn          B'110
```

End of document.