CALM
Common **A**ssembly **L**anguage for **M**icroprocessors

Peculiarities and Advantages

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

Table des matières

What is CALM ?

CALM is the abbreviation for Common Assembly Language forMicroprocessors. CALM is not a new programming language, but anotherconsistent, processor independant notation of assembly instructions.Nowadays each manufacturer defines a specific assembly language forhis microprocessor. Also the used terminology depends very much on themicroprocessor. What is still missing today is a consistent notationfor those instructions which are 100% identical.

CALM takes advantage of the fact, that many instructions - even ondifferent microprocessors - execute the same operation. Isn't itobvious in these cases to choose the same notation, independantly ofthe processor?

What defines CALM ?

CALM defines a consistent and processor independant syntax forinstructions and pseudo-instructions. Past experience and tests onover 20 microprocessors have shown that nearly all instructions of amicroprocessor could be expressed by the instructions and thenotation, which have been defined by CALM.

But CALM defines also a consistent assembly terminology. Also aconcept is presented, which shows, how an instruction is assembled.The user understands why an instruction in CALM has the givennotation. In addition a consistent notation is defined for operationcodes, addressing modes, address and data specifiers, condition codesand much more.

CALM proposes a unique notation of the instructions for all(micro)processors. CALM fulfills this goal in about 95% of theinstructions for a microprocessor. The remaining 5% representprocessor specific singularities, which could not be covered by acommon assembly language. But in many cases it is favorable, that justthese singularities are also distinguished by a different notation.

What are the advantages for the user ?

A common notation of the instructions and especially of theaddressing modes gives an objective view of the features of amicroprocessor. Therefore objective comparisons betweenmicroprocessors are also possible.

For the first time programmers of different microprocessors cancommunicate together. Up until now they failed because of thedifferent notation.

The changes of processors are also much easier, since the notationof the instructions does not change. Only the general architecture andsome singularities of the new processor must be learned.

CALM is not only appropriate for microprocessors, but also formini- processors, mainframes and microprogrammed units. CALM isextensible and for that not only limited to 8, 16 and 32 bitprocessors.And

what are the disadvantages ?

CALM defines only the software part of a processor. If one needsany hardware information (execution time, instruction code, pinconfiguration, electrical characteristics, information for theimplemented functions like timers, DMA-units, etc.), then thedocumentation of the manufacturer is necessary. Hence the user mustoccasionally know both notations of the instructions: the one of CALMand the one of the manufacturer.

Examples

The following page compares the notation of the instructions inCALM to one of the manufacturers for the microprocessors 8080, iAPX86and 68000.

```
; i8080: multiplication: RESULTAT.16 = MUL1.8 * MUL2.8
 ; modifie: A, B, D, E, H, L, F

MULT:
     MOVE   MUL1,A
     MOVE   A,E
     MOVE   #0,D
     MOVE   MUL2,A
     MOVE   #0,HL
     MOVE   #8,B
BOUCLE$:
     ADD    HL,HL
     RLC    A
     JUMP,CC SUITE$
     ADD    DE,HL
SUITE$:
     DEC    B
     JUMP,NE BOUCLE$
     MOVE   HL,RESULTAT
          RET         RET
```

```
MULT:
     LDA  MUL1
     MOV  E,A
     MVI  D,0
     LDA  MUL2
     LXI  H,0
     MVI  B,8
BOUCLE:
     DAD  H
RAL
     JNC  SUITE
     DAD  D
SUITE:
     DCR  B
     JNZ  BOUCLE
     SHLD RESULTAT
```

; iAPX86: traduit une chaîne de caractères EBCDIC en codes ASCII
; (<CR> termine); supposition: ES = DS, [DS] est équivalent à
; {DS}*16; modifie: AL, BX, CX, DI, SI, F

```
EBCDIC_ASCII:
     MOVE.16 #CONV_TAB,BX
     MOVE.16 #EBCDIC_CARACT,SI
     MOVE.16 #ASCII_CARACT,DI
     MOVE.16 [DS]+ASCII_LONGUEUR,CX
     AUTOINC
BOUCLE$:
     MOVE.8 [DS]+{SI!},AL
     MOVE.8 [DS]+{BX}+{AL},AL
     MOVE.8 AL,[ES]+{DI!}
     COMP.8 #16'D,AL
     LOOP,NE BOUCLE$
     RET.16
```

```
EBCDIC_ASCII:   PROC    NEAR
     MOV  BX,OFFSET CONV_TAB
     MOV  SI,OFFSET EBCDIC_CARACT
     MOV  DI,OFFSET ASCII_CARACT
     MOV  CX,SIZE ASCII_LONGUEUR
     CLD
BOUCLE:
     LODS EBCDIC_CARACT
     XLAT CONV_TAB
     STOS ASCII_CARACT
     CMP  AL,0DH
 LOOPNE BOUCLE
     RET   ; EQ: CR trouvé
```

; 68000: division: D4 = D5D4 / D3, reste dans D5, CS si erreur

```
DIV64:
     TEST.32
     JUMP,ZS R8^DIV_ZERO$
     PUSH.32 D0
     MOVE.32 #32-1,D0
DIV BOUCLE$:
     SETX
     RLX.32  D4
     RLX.32  D5
     JUMP,CS R8^DIV_DEPASSEMENT$
     SUB.32  D3,D5
     JUMP,HS R8^DIV_OK$
     ADD.32  D3,D5
     TCLR.32 D4:#0
DIV_OK$:
     DJ.16,NMO D0,DIV_BOUCLE$
     POP.32  D0
     CLRC
     RET
DIV_DEPASSEMENT$
     POP.32  D0
DIV_ZERO$:
     SETC
     RET
```

```
DIV64     ; modifie: D3, D4, D5, F
     TST.L  D3
     BEQ.S  ZERO
     MOVE.L D0,-(A7)
     MOVEQ  #32-1,D0
BOUCLE
     ORI    #$10,CCR
     ROLX.L D4
     ROLX.L D5
     BCS.S  DEPASSEMENT
     SUB.L  D3,D5
     BCC.S  OK
     ADD.L  D3,D5
     BCLR   #0,D4
OK
     DBRA   D0,BOUCLE
     MOVE.L (A7)+,D0
     ANDI   #$FE,CCR
     RTS
DEPASSEMENT
     MOVE.L (A7)+,D0
     ZERO
     ORI    #$1,CCR
RTS
```

The CALM assembler consists of several programs and files:

The assembly programs

ASCALM: The real assembler. It works like all other traditionalassemblers. The only difference: The programs must be written in theproducer independent assembly language CALM. Advantage: This assemblylanguage does not differ from one processor to another. With thecorresponding modules, the assembler generates machine code withoutlinker for nearly all microprocessors (see price list).

Assemblerfeatures: labels (32 significant characters), local labels, expressionwith 32 bit precision, conditional assembly (.IF/.ELSE/.ENDIF),conditional listing (.LIST/.ENDLIST), inserting of files of any size(.INS), actual system time and date can be accessed by constants,inserting error messages (language selectable) directly in the sourcefile, cross reference generator, macros, and much more.

Macrofeatures: up to eight parameters and a data specifier may be specifiedin a macro call. The length of a parameter is only limited by theinput line length. Parameters may be predefined. In a macro call, onecan distinguish if any passed parameters have been predefined or havebeen defined at the macro call. Macros may call other macros (allowednested level: 10). In addition, the passed parameters can be analyzedby built-in functions (compare, copy, test characters, etc.). Withthese functions, powerful macros may be written (i.e. translate theinstructions from the producer notation to the CALM notation). Machinecode output format: MUFOM (refer to MUFBIN).

MUFBIN: Converts the object files from the MUFOM format to theformats: binary (.BIN/.COM), hex, Intel hex (.HEX), Motorola S format(.FRS), PC/MS-DOS (.EXE) and Atari ST (.TOS/.TTP/.PRG).

Debugger: Debugger for 8086 (**DBGCALM**, PC/MS-DOS) or 68000 (**DEBUG68**,Atari ST/Smaky 100). With disassembler (CALM notation).

The utility programs

CALMMENU	presents a simple menu.
FORMCALM	formats a CALM assembly source file.
LSTTOASM	transforms a listing to a source file.
PFED	a program editor (with macros!).
PROCSET	changes in *.PRO modules the default value.
SPACETAB	replaces the spaces by tabulators in any source file.
TABSPACE	replaces the tabulators by spaces in any source file.
TESTLIST	verifies a listing file.

The files for a processor

*.DOK	CALM reference card for the processor *, i.e., Z80.DOK.
*.PRO	the module for the processor *, i.e., Z80.PRO.
B*.TXT	description of the processor module, i.e. BZ80.TXT.
C*.TXT	instruction comparison (producer notation -> CALM notation).
D*.EXE	disassembler CALM for the processor *, i.e. DZ80.EXE.
I*.TXT	list of machine codes with the CALM notation (disassembling).
ST*.ASM	list of instructions in the CALM notation (sorted alphabetically).
S_*.ASM	program examples (or E*.ASM or *.ASM).
T*.ASM	test file (list of instructions).
xxx_CALM	translator (producer notation -> CALM)
CALM_xxx	translator (CALM -> producer notation)

Note: for some processors, not all files above are available.

Object code without linker

The CALM assembler generates object files without a linker in theso-called MUFOM format. This allows the user to directly obtain anexecutable program after the transformation of the MUFOM format to thebinary format.

This is sufficient in many cases. The combination assembler-linker,which generates relocatable object modules and links them, very oftenneeds much more time than an assembler, which assembles each time thewhole source and directly generates the machine code. However, someefficient hardware is required for this (like hard disks and emulateddisks in memory).

The type of programming depends on the requirements. The programsfor simple 8 bit microprocessors (like 6502, 6800, 8080, Z80) andsingle-chip microcomputers are relatively small. Furthermore, theseprocessors are based on operating systems which load and start theuser program always at the same address.

The requirements for more efficient microprocessors (like 6809,iAPX86, 68000, NS32000) and operating systems are higher: the programsmust be loaded and executed at any memory address and the programsmust be separated in program, data and stack segments. Bothrequirements can be satisfied without problems, as these processorshave the needed addressing modes (relative addressing, indirectaddressing with any offset value, etc.).

The programmer can choose the desired addressing mode in the CALMassembler. If he wants to generate position independant programs(which can be loaded and executed at any address without relocation),then he should only use relative addressing. He can access the dataand stack segments only with the indirect addressing. The reward forthese limitations: the generated objects can be loaded and executed atany memory address without complicated and time-consuming relocation.

One has also to bear in mind that the programming field haschanged. Nobody today addresses more than one MByte of memory spacefor the program and the data with absolute addressing.

Therefore this programming concept requires a certain disciplinefrom the programmer as he can no longer use all addressing modes. Whenthe programmer does not have this discipline, an assembler with alinker is necessary.

Object code in the MUFOM format

The CALM assembler generates an object in the so-called MUFOMformat. This format has some advantages when compared to the formats.HEX (Intel) and S format (Motorola): In addition to thecharacteristics of the two "standard" formats like checksum, dataaddresses, start address, ASCII codes, alterable by an editor, etc.,the following information is given: version of the assembler and theprocessor description, indications of the processor architecture, andthe character strings of the pseudo-instructions .TITLE and .CHAPappear also in this format. All this information is uncoded (ASCIIcodes) and therefore can be read by the operation system command TYPE.

In addition, the MUFOM format is also usable for linkable objects.The MUFOM format is processor independant. Actually, the CALMassembler uses only the MUFOM commands for non-linkable objects.

Processor documentation

The delivered CALM documentation is normally not enough tounderstand a processor in all its details. This is particularly truefor microprocessors and single-chip microcomputers with built-infunctions like RAM, DMA and I/O. Hence, the correspondingdocumentation of the producer for the concerned processor is at leastnecessary. Therefore, the CALM documentation also contains comparisonlists, for example producer notation to CALM notation.

CALM reference card

On a CALM reference card, all the instructions of a microprocessorare clearly arranged in the producer independent assembler notationCALM. The benefit of this card is to give an overview.

Utility of a CALM reference card

CALM reference cards are primarily useful in daily programmingwork: Which addressing modes are allowed with AND? Which flags aremodified with COMP? etc.

But even if you have no interest for the CALM assembler, a CALMreference card may be useful to you: For example, if:

you want to better understand your own microprocessor with a different notation

you want to obtain a producer independent description of all instructions of a microprocessor

you want to compare microprocessors and want to be independant of producer informations

you look for a new, better microprocessor

you need to rate the performance of a microprocessor

you would like to indicate if a microprocessor has a specific instruction/operation code/addressing mode/data type

you want to get to know CALM first

Structure of a CALM reference card

All CALM reference cards are arranged in the same way. On the oneside, this gives an homogeneous appearance and, on the other side,direct comparisons are possible.

In addition, the operation codes of the producer are given with theCALM operation codes.Extent of supply of a CALM reference card documentation

CALM reference card documentation consists of:

CALM reference card

comparison producer notation -> CALM notation of the instructions

an example (in CALM and producer notation)

alphabetically sorted list of all operation codes (CALM notation)

alphabetically sorted list of all instruction codes with the correspondent instructions in the CALM notation

Example of a CALM reference card

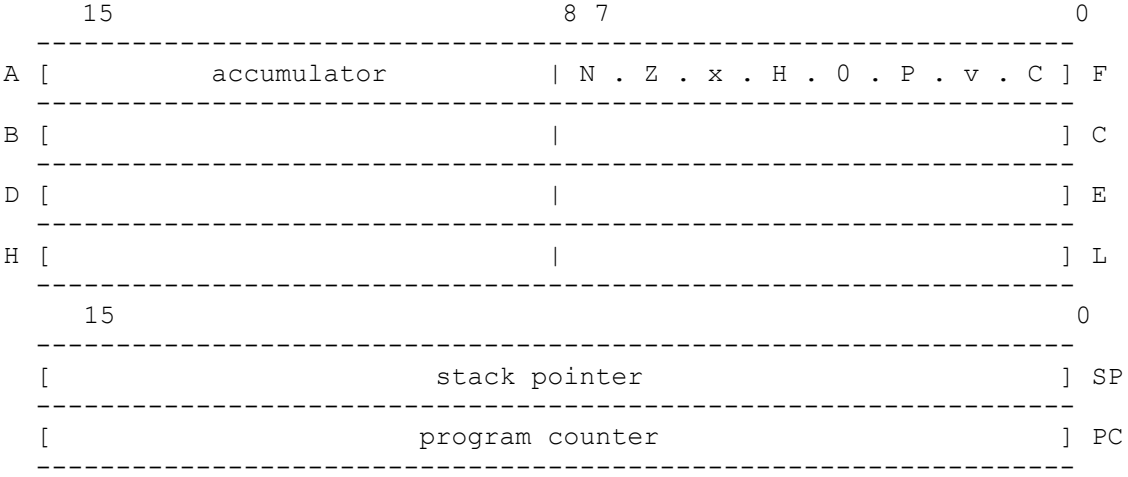
The CALM reference card of the microprocessor 8080/5 is presentedin the following pages.

8080/8085 - 1

8080/8085
CALM REFERENCE CARD

8080/8085 Description

Programming Model



General
Address: 16 bit
Data: 8 bit (8085: data multiplexed with addresses A0-A7)

Abbreviations used
v 16'0, 16'8, 16'10, 16'18, 16'20, 16'28, 16'30, 16'38
r8 A B C D E H L
s8 B C D E H L
r16 BC DE HL SP
i8 {BC} {DE} {HL}
VAL8 8-bit value
VAL16 16-bit value
cc EQ NE CS CC MI PL PO PE

Modifications versus CALM Standard
8 Bit: All transfers are 8 bits wide, except those determined by register names (1 letter = 8 bit, 2 letters = 16 bits).
Flag v Unspecified 8085 flag: 2's complement overflow (in arithmetic 8-bit and 16-bit operations). 8080: flag is always 1. (U8085)
Flag x Unspecified 8085 flag: sign(op1)*sign(op2) + sign(op1)*sign (result)(U8085) + sign(op2)*sign(result). For COMP and SUB, invert sign(op2). 8080: flag is always 0.

Remarks
- flag equalities: EQ=ZS, NE=ZC, CS=LO, CC=HS, MI=NS, PL=NC.
- Reset:
```
IOFF
JUMP 16'0
```
- Interrupt:
```
IOFF
CALL v
```
Additional interrupt addresses for 8085: 16'2C, 16'34, 16'3C. (8085)
- NMI:
```
IOFF (8085)
CALL 16'24
```
- CALM - Intel register names: equal except: F=PSW and 16 bit names.
- CALM - Intel flag names: N=S, Z=Z, H=AC, P=P, C=C.

8080/8085 - 2

```
Transfer instructions
MOVE       #VAL8 |,A    []
           VAL16 |
           r8    |
           i8    |
           $VAL8 |
           A,| VAL16
             | r8
             | i8
             | $n
(MVI LDA MOV LDAX IN STA MOV STAX OUT)

MOVE       #VAL8 |,s8   []
           s8    |
           {HL}  |
           s8,{HL}
(MVI MOV MOV)

MOVE       #VAL16,r16   []
           VAL16,HL
           HL,VAL16
           HL,SP
(LXI LHLD SHLD SPHL)

MOVE       HL,{DE}      []       (U8085)
           {DE},HL
           #{HL}+VAL8,DE
           #{SP}+VAL8,DE
(SHLX LHLX LDHI LDSI)

PUSH |     r16          [], r16 without SP
POP  |     AF           [], [all] if POP AF
(PUSH POP)

SETC                    [C=1]
(STC)

EX         DE,HL        []
               {SP},HL
(XCHG XTHL)

Arithmetic instructions
ADD  |     #VAL8 |,A    [N,Z,H,P,C]
ADDC |     r8    |      [N,Z,H,P,C]
SUB  |     {HL}  |      [N,Z,H,P,C]
SUBC |                  [N,Z,H,P,C]
COMP |                  [N,Z,H,P,C]
(ADI ADD ACI ADC SUI SUB SBI SBB CPI CMP)

ADD        r16,HL       [C]
(DAD)

SUB        BC,HL        [N,Z,x,H,P,v,C] (U8085)
(DSUB)

INC |      r8           [N,Z,H,P]
DEC |      {HL}         [N,Z,H,P]
(INR DCR)

INC |      r16          []
DEC |
(INX DCX)
```

8080/8085 - 3

```
 Logical instructions
AND |     #VAL8 |,A    [N,Z,H,P,C=0]
OR  |     r8    |      [N,Z,H,P,C=0]
XOR |     {HL}  |      [N,Z,H,P,C=0]
(ANA ANI ORA ORI XRA XRI)

NOT       A            []
NOTC                   [C]
(CMA CMC)

Shift instructions
RR  |     A            [C = A:#0]
RRC |                  [C = A:#0]
RL  |                  [C = A:#7]
RLC |                  [C = A:#7]
(RRC RAR RLC RAL)

ASR       HL           [C = L:#0] (U8085)
RLC       DE           [v,C = D:#7] (U8085)
(ARHL RDEL)

Program flow instructions
JUMP,cc | VAL16        []
JUMP    |
CALL,cc |
CALL    |
JUMP      {HL}         []
JUMP,XC | VAL16        []       (U8085)
JUMP,XS |
(J- JMP C- CALL PCHL JNX5 JX5)

RST       v            []
RST,VS    16'40        []       (U8085)
(RST RSTV)             one byte call (restart)

RET,cc                 []
RET                    []
WAIT                   []
NOP                    []
ION                    []
IOFF                   []
(R- RET HLT NOP EI DI)

Special instructions
DAA       A            [N,Z,H,P,C]
(DAA)                  Decimal Adjust A, only valid after ADD and ADDC

RIM |     A            []       (8085)
SIM |                  RIM: read interrupt mask
(RIM SIM)              SIM: set interrupt mask

RETEM                  [all]   return from emulation mode (V20)
(RETEM)                POP.16 IP; POP.16 CS; POP.16 SF;
                                  MD bit write disable

TRAPNATIVE #VAL8       [MD=1]   trap to native mode (8086)(V20)
(CALLN)                PUSH.16 SF; PUSH.16 CS; PUSH.16 IP; SET MD;
                                  return with RETI.32
Notes
(8085)    only available in 8085.
(U8085)   unspecified 8085 flag or operation code.
(V20)     only available in V20, V30, V40, and V50 (8080 emulation mode).
                                          (c) Patrick Faeh, June 1985.
```