

Exemples de programme: notation CALM - notation du fabricant.

You will find examples for the processors: 6502, 6809, 6805, 6811, 8048, 8051, 8080, 68000, Z80, 6800, iAPX86, and NS32000.

6502:

; Sums N (<= 256) elements, result in SUM_LSB..SUM_MSB (16 Bit)

; modified registers: A, Y, F and SUM_LSB, SUM_MSB

```

SUM:                                SUM
      MOVE    #0,A                    LDA    #0
      MOVE    A,SUM_LSB               STA    SUMLSB    ; SUMLSB = 0
      MOVE    A,SUM_MSB               STA    SUMMSB    ; SUMMSB = 0
      MOVE    A,Y                      TAY
      MOVE    {Y}+{BEGIN},A           LDA    (BEGIN),Y
      MOVE    A,Y                      TAY                ; length
LOOP$:                                LOOP
      CLRC
      MOVE    {Y}+{BEGIN},A           LDA    (BEGIN),Y
      ADDC    SUM_LSB,A               ADC    SUMLSB
      MOVE    A,SUM_LSB               STA    SUMLSB
      JUMP,CC NO_CARRY$               BCC    NOCARRY
      INC     SUM_MSB                 INC    SUMMSB
NO_CARRY$:                            NOCARRY
      DEC     Y                       DEY
      JUMP,NE LOOP$                   BNE    LOOP
      RET                               RTS

```

6809:

; transfers a block (<= 64K) of data

; modified registers: AB, IX, IY, US, F and the destination block

```

BLOCK:                                BLOCK
      MOVE    #LENGTH,AB              LDD    #LENGTH
      SR      A                       LSRA
      RRC     B                       RORB
      JUMP,CC R8^EVEN$                 BCC    EVEN
      ADD     #1,AB                     ADDD   #1
EVEN$:                                EVEN
      MOVE    #SOURCE,IY               LDY    #SOURCE
      MOVE    #DESTINATION,US           LDU    #DESTINATION
LOOP$:                                LOOP
      MOVE    {IY+2+},IX               LDX    ,Y++      ; 16 bit
      MOVE    IX,{US+2+}               STX    ,U++
      SUB     #1,AB                     SUBD   #1
      JUMP,NE LOOP$                     BNE    LOOP
      RET                               RTS

```

```

6805:
; Sums N (<= 256) elements, result in SUM_LSB..SUM_MSB (16 Bit)
; modified registers: A, X, F and SUM_LSB, SUM_MSB
SUM:
        CLR        A                CLRA
        MOVE       A,SUM_LSB        STA        SUMLSB    ; SUMLSB = 0
        MOVE       A,SUM_MSB        STA        SUMMSB    ; SUMMSB = 0
        MOVE       BEGIN,X          LDX        BEGIN    ; length
LOOP$:
        CLRC
        MOVE       {X}+BEGIN,A      LDA        BEGIN,X
        ADDC       SUM_LSB,A        ADC        SUMLSB
        MOVE       A,SUM_LSB        STA        SUMLSB
        CLR        A                CLRA
        ADDC       SUM_MSB,A        ADC        SUMMSB
        MOVE       A,SUM_MSB        STA        SUMMSB
        DEC        X                DECX
        JUMP,NE   LOOP$             BNE        LOOP
        RET
        RTS

```

```

6811:
; transfers a block (<= 64K) of data
; modified registers: AB, IX, IY, F and the destination block
BLOCK:
        MOVE       SOURCE,IX        LDX        SOURCE
        MOVE       DESTINATION,IY   LDY        DESTINATION
        MOVE       LENGTH,AB        LDD        LENGTH
LOOP$:
        PUSH       A                PSHA
        MOVE       {IX},A           LDAA       0,X
        MOVE       A,{IY}           STAA       0,Y
        POP        A                PULA
        INC        IX                INX
        INC        IY                INY
        SUB        #1,AB             SUBD       #1
        JUMP,NE   LOOP$             BNE        LOOP
        RET
        RTS

```

8048

; subtraction: R6R5 = %{R0}.16 - %{R1}.8

; modified registers: A, R5, R6, F

```

SUB:
    MOVE    %{R0},A
    NOT     A
    ADD     %{R1},A
    NOT     A
    MOVE    A,R5
    INC     R0
    MOVE    %{R0},A
    MOVE    A,R6
    DEC     R0
    JUMP,CC END$
    DEC     R6

SUB:
    MOV     A,@R0
    CPL     A
    ADD     A,@R1
    CPL     A
    MOV     R5,A
    INC     R0
    MOV     A,@R0
    MOV     R6,A
    DEC     R0
    JNC     END
    DEC     R6

END$:
    RET

END:
    RET

```

8051

; send a .ASCIZ character string to the serial port

; The character string is located directly after the call:

; CALL SENDASCIZ

; .ASCIZ "text"

; modified registers: A, DP, F

```

SENDASCIZ:
    POP     D
    POP     P
    CLR     A
    JUMP    R8^SIGN$

SENDASCIZ:
    POP     DPH ; DP ^ASCIZ
    POP     DPL
    CLR     A
    SJMP    SIGN

LOOP$:
    TESTJUMP,BC SCON:#TI,APC
    CLR     SCON:#TI
    MOVE    A,SBUF
    INC     DP

LOOP:
    JNB     TI,$ ; sender ready ?
    CLR     TI
    MOV     SBUF,A
    INC     DPTR

SIGN$:
    CLR     A
    MOVE    {DP}+{A},A
    COMPJUMP,NE #0,A,LOOP$
    MOVE    #1,A
    JUMP    {DP}+{A}

SIGN:
    CLR     A
    MOVC   A,@A+DPTR
    CJNE   A,#0,LOOP
    MOV     A,#1
    JMP    @A+DPTR

```

8080

; multiplication: RESULT.16 = MUL1.8 * MUL2.8
 ; modified registers: A, B, D, E, H, L, F

```

MULT:                                MULT:
      MOVE    MUL1,A                 LDA    MUL1
      MOVE    A,E                    MOV    E,A
      MOVE    #0,D                   MVI    D,0
      MOVE    MUL2,A                 LDA    MUL2
      MOVE    #0,HL                  LXI    H,0
      MOVE    #8,B                   MVI    B,8

LOOP$:                                LOOP:
      ADD     HL,HL                   DAD    H
      RLC     A                      RAL
      JUMP,CC NEXT$                 JNC    NEXT
      ADD     DE,HL                  DAD    D

NEXT$:                                NEXT:
      DEC     B                      DCR    B
      JUMP,NE LOOP$                 JNZ    LOOP
      MOVE    HL,RESULT              SHLD   RESULT
      RET
  
```

68000

; division: D4 = D5D4 / D3, rest in D5, CS if error
 ; modified registers: D3, D4, D5, F

```

DIV64:                                DIV64
      TEST.32 D3                    TST.L  D3
      JUMP,ZS R8^DIV_ZERO$         BEQ.S  ZERO
      PUSH.32 D0                   MOVE.L D0,-(A7)
      MOVE.32 #32-1,D0             MOVEQ  #32-1,D0

DIV_LOOP$:                             LOOP
      SETX
      RLX.32 D4                    ROLX.L D4
      RLX.32 D5                    ROLX.L D5
      JUMP,CS R8^DIV_OVER$         BCS.S  OVER
      SUB.32 D3,D5                 SUB.L  D3,D5
      JUMP,HS R8^DIV_OK$           BCC.S  OK
      ADD.32 D3,D5                 ADD.L  D3,D5
      TCLR.32 D4:#0               BCLR  #0,D4

DIV_OK$:                                OK
      DJ.16,NMO D0,DIV_LOOP$       DBRA  D0,LOOP
      POP.32 D0                    MOVE.L (A7)+,D0
      CLRC                          ANDI  #$FE,CCR
      RET                          RTS

DIV_OVER$:                             OVER
      POP.32 D0                    MOVE.L (A7)+,D0

DIV_ZERO$:                             ZERO
      SETC                          ORI   #$1,CCR
      RET                          RTS
  
```

Z80

; transfers a block of data. The subroutine checks for overlapping.
 ; modified registers: A, BC, DE, HL, F

TRANSF:		TRANSF:	
MOVE	SOURCE,HL	LD	HL,(SOURCE)
MOVE	DESTINATION,DE	LD	DE,(DESTINATION)
MOVE	LENGTH,BC	LD	BC,(LENGTH)
OR	A,A	OR	A
SUBC	DE,HL	SBC	HL,DE
ADD	DE,HL	ADD	HL,DE
JUMP,CC	R8^NO_OVER\$	JR	NC,NOOVER
DEC	BC	DEC	BC
EX	DE,HL	EX	DE,HL
ADD	BC,HL	ADD	HL,BC
EX	DE,HL	EX	DE,HL
ADD	BC,HL	ADD	HL,BC
INC	BC	INC	BC
LDDR		LDDR	
JUMP	R8^END\$	JR	END
NO_OVER\$:		NOOVER:	
	LDIR		LDIR
END\$:		END:	
	RET		RET

6800

; convert a binary number to BCD (8 bit)
 ; modified registers: A, B, IX, F

BINBCD:		BINBCD	
CLR	BCD	CLR	BCD
MOVE	BIN,A	LDA	A,BIN
COMP	#100,A	CMP	A,#100
JUMP,HS	OVERFLOW	BHS	OVERFLOW ; too big?
MOVE	#8,IX	LDX	#\$08
LOOP\$:		LOOP	
MOVE	BCD,A	LDA	A,BCD
MOVE	A,B	TAB	
AND	#16'F,A	AND	A,\$0F
SUB	#16'5,A	SUB	A,\$05
JUMP,MI	LSD\$	BMI	LSD
ADD	#16'3,B	ADD	B,\$03
LSD\$:		LSD	
MOVE	B,A	TBA	
AND	#16'F0,A	AND	A,\$0F0
SUB	#16'50,A	SUB	A,\$50
JUMP,MI	MSD\$	BMI	MSD
ADD	#16'30,B	ADD	B,\$30
MSD\$:		MSD	
MOVE	B,BCD	STA	B,BCD
SL	BIN	ASL	BIN
RLC	BCD	ROL	BCD
DEC	IX	DEX	
JUMP,NE	LOOP	BNE	LOOP
RET		RTS	

iAPX86

; translate an EBCDIC character string to ASCII-codes (<CR>
 ; terminates); assume: ES = DS, [DS] is equivalent to {DS}*16
 ; modified registers: AL, BX, CX, DI, SI, F

```

EBCDIC_ASCII:          EBCDIC_ASCII:  PROC    NEAR
    MOVE.16    #CONV_TAB,BX          MOV     BX,OFFSET CONV_TAB
    MOVE.16    #EBCDIC_CHAR,SI       MOV     SI,OFFSET EBCDIC_CHAR
    MOVE.16    #ASCII_CHAR,DI        MOV     DI,OFFSET ASCII_CHAR
    MOVE.16    [DS]+ASCII_LENGTH,CX  MOV     CX,SIZE ASCII_LENGTH
    AUTOINC
LOOP$:                LOOP:
    MOVE.8     [DS]+{SI!},AL          LODS   EBCDIC_CHAR
    MOVE.8     [DS]+{BX}+{AL},AL      XLAT   CONV_TAB ; translate
    MOVE.8     AL,[ES]+{DI!}          STOS   ASCII_CHAR
    COMP.8     #16'D,AL               CMP    AL,0DH
SKIP,EQ DJ.16,NE CX,R8^LOOP$          LOOPNE LOOP
    RET.16
                                RET     ; EQ: CR found
  
```

NS32000

; compares a character string (register SB points to it) with a list.
 ; The elements in the list are all 8 characters long. When a character
 ; string is found, the program jumps to the correspondant subroutine.
 ; modified registers: R1, R6, R7, F

```

COMP_TEXT:            COMPTXT:
    MOVE.32    #{SB}+0,R1            ADDR   0(SB),R1
    MOVE.32    #0,R7                 MOVQD  0,R7
    JUMP       R7^COMPARE$           BR     COMPARE:B
NEXT$:              NEXT:
    ADD.32     #1,R7                 ADDQD  1,R7
COMPARE$:          COMPARE:
    COMPM.8    {R1}+0%#8,{R7}*8+TAB$ CMPMB  0(R1),TAB[R7:Q],8
    JUMP,EQ    R14^FOUND$            BEQ    FOUND:W
    CHECK.A8   LIMITS,R7,R6.32       CHECKB R6,LIMITS,R7
    JUMP,VC    NEXT$                 BFC    NEXT
    JUMP       ERROR                 BR     ERROR
FOUND$:           FOUND:
    JUMP       {PC}+{{R7}*2+R7^TJUMP$} CASEW  TJUMP:B[R7:W]
TJUMP$:          TJUMP:
    .16       AD_BEGI-FOUND$         .WORD  AD_BEGI-FOUND
    .16       AD_END-FOUND$         .WORD  AD_END-FOUND
    .16       AD_PROG-FOUND$        .WORD  AD_PROG-FOUND
    .16       AD_PROC-FOUND$        .WORD  AD_PROC-FOUND
TAB$:            TAB:
    .ASCII    "BEGIN  "             .BYTE  "BEGIN  "
    .ASCII    "END    "              .BYTE  "END    "
    .ASCII    "PROGRAM "            .BYTE  "PROGRAM "
    .ASCII    "PROCEDUR"           .BYTE  "PROCEDUR"
LIMITS:         LIMITS:
    .8        (LIMITS-TAB$)/8-1     .BYTE  (LIMITS-TAB)/8-1
    .8        0                     .BYTE  0
  
```