Franck Di Maio

SMACC system software October 88

PS/CO/Note 89-02

SMACC system software
August 88 Revision

Contents

Part 1

User Manual

1. Bugs Corrections

1.1 Timer bomb

RMS version 4.4 kernel hold a bug which caused a dead−lock (infinite loop at interrupt level 7), about 20 days after startup. This bug has been corrected

1.2 MIOS crashes

MIOS (Monica Input/Output System) is still (and will stay forever) a fancy product. Block IO of characters are not well supported. There are 2 ways to receive blocks of charaters: turning echo on and terminating blocks with a "Carriage Return". The first solution has been used in the previous EPROM versions, it leads to system crashes under certain load circumstances.

MIOS has not been corrected in this version, but the communication software with the Macintosh has been modified to use CR termination instead of echo.

MacNodal Versions 2.1D12 and upper are compatible with this correction.

2. User Programs Monitoring

2.1 General Description

Programs monitoring is acomplish in the SMACC by an exception monitor task which may be consulted remotely by means of interface routines accessible via RPC. Only the programs (up to 8) declared by the user, currently via STD_TASK macro, are monitored this way. The purpose is to provide (1) synthetic information on the status of every user's programs and (2) detailed information on any user's program for diagnostic or debugging purposes.

2.2 Functionnality

1. Provide RPC interface to obtain global information for every SMACC's programs: name, internal status (started, stopped by an error or terminated), and RMS status if meaningful.

2. Provide RPC interface to obtain detailed information for any monitored program: 68000 registers and diagnostic message if program is stopped.

3. Provide support (diagnostic messages) for exceptions and run−time errors (from compilers and libraries). Diagnostic messages for run−time errors may be wrong when using monica vs 1.0 libraries.

4. Provide support (library + diagnostic messages) for the treatment of SMACC specific errors: RMS errors, ISR exceptions...

2.3 Structure

Event−driven exception monitor. Must be started prior to any user program. Dedicated to user programs monitoring only. Crash system if error in this task (no crash for user programs).

2.4 Interface

2.4.1 Functions

2.4.1.1 Diagnostic Functions

Global status:

```
PGSTAT(NB, NAMES., INT.ST, RMS.ST)
```

NB is integer reference,
NAMES. is string reference,
INT.ST and RMS.ST are integer arrays.
Every array size must be 8.

The number of monitored programs is put into NB, every task's ID first long word are concatenated into NAMES. (4 bytes / program), internal status and RMS status into INT.ST and RMS.ST.

- internal status:
  0 – non–existent,
  1 – started,
  2 – stopped,
  3 – terminated,

- RMS status (reduced to 16 bits) :
  bit 15 – dormant,
  bit 14 – wait,
  bit 13 – wait on semaphore,
  bit 12 – wait for event,
  bit 11 – wait for acknowledgement (from trap server),
  bit 10 – wait for command (from exception monitor),
  bit 9 – suspend,
  bit 8 – reserved,
  bit 7 – termination pending,
  bit 6 – will return to RMS when next dispatched,
  bit 5 – dispatched to ASR,
  bit 4 – ready,
  bit 3 – wakeup pending,
  bit 2 – terminating, wait for ack,
  bits 1 & 0 – reserved,
  Remark : bits 8 to 15 may be tested to know whether a program is running or waiting.

At least 32 bytes must be available in the string, otherwise,Nodal error #40 ("Result string filled") is raised. Nodal error #23 ("Array dimension error") is raised if pb with array parameters.

sample syntax (misses some checks):

```
>SE SMACC.=1
>SE NB = 0; $SE NAMES.="";DIM-I INT.ST(8) DIM-I RMS_ST(8)
>RPC#SMACC. PGSTAT W_NB[16] W_NAMES. W_INT.ST W_RMS.ST
>FOR I=1,NB;TYPE SUBS(1+(I-1)*4,4+(I-1)*4,NAMES.) INT.ST(I) ]]RMS.ST(I) !
```

Detailed information on one program:

```
PGWHAT(NAME., INT.ST, RMS.ST, D.REGS, A. REGS, PC, SR, MESS.)
```

NAME. is string value (string size = 4),
INT.ST & RMS.ST are integer references,
D.REGS & A.REGS are integer arrays (size= 16),
PC is integer array (size = 2),
SR is integer reference,
MESS. is string reference.

NAME. parameter is used to pass the first long–word of the task ID. If the size is not 4, Nodal error # 55 (string function failure) is raised. INT.ST & RMS.ST receive status information, as for

PGSTAT. D.REGS & A.REGS receive Dn & An register values, D.REGS(1) is the most significant word of D0 (bits 16 to 31), D.REGS(2) is the less significant word of D0 (bits 0 to 15) etc. PC & SR receive Program Counter and Status Register. MESS. receive a diagnostic message if INT.ST is "Stopped" (i.e. 2). RMS.ST, D.REGS, A.REGS, PC and SR are valid only if INT.ST is "Started" or "Stopped".

Sample messages:
"Link–edit error",
"Stack overflow error",
"Illegal instruction",
"Arithmetic overflow in ISR, at PC=001000",
"RMS error: START – $0003"...

2.4.1.2 Error functions

A set of functions is provided in ~smacc/lib/xmon_lib.ccu in order to let the user generate well defined errors under certain circumstances: RMS directive failure, ISR exception event reception, internal fatal error detection... These user generated errors are handled in the same manner as run–time errors (ex: arithmetic overflow), this means, currently, that the faulty task is left stopped in its error state. Diagnostic messages describing these errors are build by the interface functions of this module.

These function generates "line 1010" instruction, as described in the previous chapter. No stack frame is created (i.e. no LINK/UNLK instructions) to facilitate debugging: it leaves the caller stack frame pointer in A6. These functions are therefore written in assembly language, C syntax is provided for clarity.

RMS errors checking:

Traps if a result code, as returned by a RMS directive, is != 0.

```
void chk_rms_code(code) long code; {}
```

Example:

```
chk_rms_code(C_START(&pb_start, &name));
/* will generate fatal error if illegal target task ID */
```

ISR errors checking:

Traps if an ISR event reports an exception. Load the event address in A0 before the exception.

```
void chk_isr_event(event_address) long event_address; {}
```

Example:

```
chk_rms_code (C_GTEVNT (event_address)); /* get next event (critical) */
chk_isr_event ((long) event_address); /* if isr exception, fatal error */
switch(event_address->type) ...       /* else */
```

Auto error generation:

Traps every time. The user must pass a long integer value which will be put in D0.L

```
void self_error(code, message) long code; char *message; {}
```

Example:

```
/* if data != 0, fatal error, record time */
   if (data != 0) {
               sprintf(message, "Illegal value: %d", data);
               self_error(time(),message);      /* put time in parameter */
```

2.4.2 Data

2.4.2.1 Monica – like SMACC error codes

Opcodes: $A1xx.
$A180 – RMS error
$A181 – ISR error
$A182 – Self error
$A183 – Exception return error (top frame reached)
$A184 – Illegal entry – point (dummy entry in smacc's libs)

3. Debugger Interrupt

3.1 General description

An interrupt routine is connected to the level 7 auto–int vector at startup and may also be connected to other interrupts (trace int) by the debugger (SMACC–DEBUG) in order to (1) be able to make a snap–shot of the processor's registers or (2) force the system to enter into a debugging status.

3.2 Functionnality

1. React to unmaskable debugger interrupt generated by the FEC via the CAMAC "Abort" function: F25.A2 by (1) saving the processor status (registers value) in a place known by the debugger and (2) suspending the processor (self F25.A1).

When this interrupt is raised, the SMACC may either be signalled to continue (F25.A0) or to restart (F28.A0).

3.3 Interface

3.3.1 Signals

1. The debugger signals the SMACC to enter into debugging mode by the CAMAC "Abort" function: F25.A2.
2. The SMACC inform the debugger of beeing in debugging status by issuing the CAMAC suspend function on itself: F25.A1, therefore setting the "suspend" bit (bit 14) in "Processor Status Register" (word read by F1.A2).
3. A SMACC in debugging state may be signalled to continue by the debugger via the CAMAC "continue" function : F25.A0 or to restart via the CAMAC "reset" function: F28.A0.

3.3.2 Data

Global variable : "DEBREGS", pointer to the current System Stack where registers have been saved:

```
struct {
    long USP;
              /* User Stack Pointer at the time of the interrupt */
    long D_registers[8];/* D0-D7 */
    long A_registers[7];/* A0-A6 */
    long int_PC;        /* Interrupt vector + 4 */
    short SR;           /* Status Register */
    long PC;            /* Program counter */
} *DEBREGS;
```

DEBREGS value + 74 is therefore SSP value at the time of the interrupt.

A pointer to this vector is in the jump−table at $80018.

4. Interrupts monitoring

4.1 General description

A facility must be provided for exploitation purposes in order to check that interrupts are correctly triggered in a given SMACC. This facility could be used by a monitoring program in the FECs.

4.2 Functionnality

1. Provide via RPC, for LAM, FPI & INTRQ interrupts, the number of time the interrupt occured since startup (modulo 2¬15) if an Interrupt Service Routine is connected (via RMS) or a fixed value (−1) if not. As it may be executed frequently, the RPC must be optimized for FEC load sake.

2. Update internal tables for counting the occurence of LAM, FPI & INTRQ interrupts if an ISR is connected. This may only be used by specialists if the RPC is not available.

If the RMS mechanism is by−passed (i.e. if code address is directly put in interrupt vector), the count stays 0.

4.3 Interface

4.3.1 Procedures

Count values returned by procedures:
−1 if no ISR connected,
0 if ISR connected but never triggered,
> 0 otherwise.

INTCNT(LAM.C, FPI.C, IRQ.C)

Parameters:

LAM.C is integer array (size = 23), index is station number

FPI.C is integer array (size = 4), index is FPI number
IRQ.C is integer array (size = 2), index is INT.RQ number

4.3.2 Data

For specialists only. 2 tables must be consulted in case RPC cannot be used.

4.3.2.1 Interrupt vectors table

```
struct {
    long spurious;          /* vector 64 */
    long lam[23];           /* lam[0] is station 1 */
    long dummy_1[8];
    long intRq[2];          /* intRq[0] is INT.RQ2 */
    long dummy_2[2];
    long fpi[4];            /* fpi[0] is FPI4 */
```

The address of this table is $100. If vector is "COMINT" address (address in map), interrupt is not connected. "COMINT" address should be left in dummy pointers (dummy_1 & dummy_2 arrays).

4.3.2.2 Interrupt counts table

```
struct {
    short spurious;
    short lam[23];          /* lam[0] is station 1 */
    short dummy_1[8];
    short intRq[2];         /* intRq[0] is INT.RQ2 */
    short dummy_2[2];
    short fpi[4];           /* fpi[0] is FPI4 */
```

Counts are always > = 0. All values are set to 0 at system startup. The external name of the table is "_INTCOUNT" (for map use). A 0 value means (1) no interrupt service routine is connected, (2) the interrupt does not occured or (3) special code address has been directly put into the vector table. Case (1) and (2) may be identified by looking into the interrupt vectors table: the vector is either COMINT or a pointer to an entry in the I/O vector table containing the code: "JSR COMNISR" (address in map). No pointer to this table is available in application environment.

5. Clear RAM Interrupt

5.1 General description

An interrupt routine pointer is available in the jump–table. It may be connected to the level 7 auto–int vector by a FEC's program in order to force a "reset to 0" of the whole RAM. This may be used to make a quick cleaning of the memory and check that all the memory is accessible.

5.2 Functionnality

1. Clear all RAM

2. Check that the RAM is 0s. ~~all RAM~~.

3. Maximum execution time : 1s (maximum delay between the interrupt arrival and a normal execution termination).

5.3 Interface

A pointer to the vector is in the jump–table at $80028. This pointer must be copied into level 7 interrupt Autovector ($7C) before sending the interrupt.

The interrupt is raised by CAMAC "Abort" function: F25.A2.

The interrupt normal termination is signaled by (1) the SMACC beeing suspended (bit 14 in PSR, read via F1.A2) and (2) a 0.L at location 0.

Part 2

Implementation Details

6. XMON Module – User Programs monitoring

6.1 Programs table

```
#define MAX_TASK 8              /* Req 2.1.4 */

typedef struct {
    long    name;
    short   status;
    short exc_code;             /* used to record exception code */
} prog_info;

static prog_info prog_table[MAX_TASK];
```

6.2 User events

When a system task : NODI or XRPC executes an interface function, it requests the target task′ status by posting a user event to the exception monitor.

```
struct {
    char length;        /* 14 */
    char code;          /* 3 */
    long taskID;        /* requesting task */
    long dummy_1;
    long name;          /* target task */
    long status_receive_area_pointer;
```

6.3 Initialization

- Clear names in program table and set status to ″non – existent″ (0)
- Allocate ASQ, size for 1 event / program.

6.4 User event handler

– acquire task state (RSTATE) – wake – up requesting task

6.5 Programs status function

Nodal function : PGSTAT.

C function:

```
void xmon_pgstat(number, names, int_status, rms_status)
short   *number;
char *names;
short   *int_status,
          *rms_status;
{}
```

- Reset number to 0 and names to "".
- Scan programs table: for every programs with (name != 0)
  - — increment number & names
  - — copy internal status into int_status,
  - — if status = = "started", acquire task state (QEVENT + WAIT) and copy rms status into rms_status parameter.

6.6 Program detailed information

Nodal function: PGWHAT.

C function:

```
void xmon_pgwhat(name, int_status, rms_status,
    d_regs, a_regs, pc, sr, message)
long name;
short   *int_status,
          *rms_status;
long    *d_regs,
          *a_regs;
long *pc;
short *sr;
char *message;
```

- Get program table entry from name.
- Acquire task's state (QEVNT + WAIT).
- Update register parameters and rms_status.
- If internal status is "stopped", build diagnostic message.

6.7 Attach event handler

- Allocate an entry in progs table & record name.
- Set status to "non-existent".
- Start task (REXMON)
- Set status to "started"

6.8 Detach event handler

- Set status to ″terminated″.

6.9 Exception event handler

- Set status to ″stopped″

7. INTSURV module – Interrupts monitoring

7.1 Module Content

- Count table.
- Init function (see next section).
- C interface function (see lower)

Interface function (external name _GETINDCOUNT) :

```
short getIntCount (lamData, fpiData, irqData)
    short *lamData, *fpiData, *irqData;
{}
```

File : ¬ smacc/rms_sys/intsurv.c

7.2 Initialisation

At startup, the monitoring task (″XMON″ or ″ERLG″) execute ″initIntCount″ routine (name ″_INITINTCOUNT″). This function set all counts to 0 and copies into an internal address the vector ″dummy_1[0]″ (see upper) which should be COMINT address.

File : ¬ smacc/xmon/xmon.c

7.3 COMNISR patch

- Increment interrupt′s counter.
- Wrap to 1 if < 0.

File : ¬ smacc/rms68k/kernel.src

7.4 Nodal interface

Internal C routine is interfaced for RPCs, the size and type of the arrays are checked, array dimension error (23) is returned if not consistent. C functions return value is also treated as Nodal error if < > 0.

File : ¬ smacc/rms_gen/nod2c_interf.asm

8. CLEARINT – Clear RAM interrupt

When the interrupt is activated:

- write $50000.L at location 0 (counter)
- while counter > 0
    - decrement counter
    - clear location
    - check value is 0
    - if not, suspend SMACC.
- suspend SMACC.

Appendix A

Sample NODAL program using new features (SMACC – DEBUG overlay).

```
%c SAMPLE PROGRAM USING VS 2.2 (AUG 88) EPROM FACILITIES
%c  F. DI MAIO

%odal
1.05 DO 5
1.10 TY "1 - Processor snapshot"!
1.20 TY "2 - Interrupts monitoring"!
1.30 TY "3 - Programs monitoring"!
1.40 TY "4 - Clear & test whole RAM"!
1.91 ASK "What" OPT.
1.92 IF OPT. > 4 ; GOTO 1.10
1.93 DO OPT.*10
1.99 GOTO 1.10

5.05 DIM-I T(256)
5.10 SE LP=0;SE CR=0;SE ST=0
5.15 ADACC(ARG(1),LP,CR,ST);IF LP=-1;TY "Illegal ACC#" ARG(1)!;END
5.20 SE XQ=0

6.10 IF BIT(14,XQ)<>0; RET
6.15 TY "No X Response,"
6.20 TY " loop " %1 LP  ", crate " %1 CR !
6.30 END

% Processor Snapshot
10.05 SE V=SCAM(LP,CR,ST,2,1,XQ); DO 6; % PSR
10.06 IF BIT(14,V) = 1; TY "SMACC is already SUSPENDed" !; RET
10.10 SE V=SCAM(LP,CR,ST,2,25,XQ);DO 6 ;% "Abort" function
10.15 SE V=SCAM(LP,CR,ST,2,1,XQ); DO 6; % PSR
10.20 IF BIT(14,V) = 0; TY "SMACC does not respond" !; RET
10.25 SE T(1)=8; SE T(2)=[[0018
10.30 GET(ADT(T,1),T,2);GET(ADT(T,1),T,2);SE DR=ADT(T,1); GET(DR,T,37)
10.35 SE DR=DR+74 ;% System SP at the time of interrupt
10.40 TY "SSP = " ]](DR/65536) ]]MOD(DR,65536)
10.55 TY " - USP = " ]]T(1) ]]T(2) !
10.57 TY "PC = " ]]T(36) ]]T(37) " - SR = " ]]T(35) !
10.60 FOR I=0,7;TY &3 "D" %1 I "=" ]]T(3+I*2) ]]T(4+I*2);IF MOD(I,4)=3;TY !
10.65 FOR I=0,6;TY &3 "A" %1 I "=" ]]T(19+I*2) ]]T(20+I*2);IF MOD(I,4)=3;TY !
10.66 TY !
10.70 SE V=SCAM(LP,CR,ST,0,25,XQ); DO 6;% Signal the SMACC to continue
10.75 SE V=SCAM(LP,CR,ST,2,1,XQ); DO 6; % PSR
10.76 IF BIT(14,V) = 1; TY "SMACC does not CONTINUE"!

% Interrupts Monitoring
20.05 DIM-I LAM.C(23); DIM-I FPI.C(4); DIM-I IRQ.C(2)
20.10 RPC#ARG(1) INTCNT W_LAM.C W_FPI.C W_IRQ.C
20.15 TY &2 "LAMs " &45 "FPIs " &5 "INT.RQ"!
20.20 FOR I=1,5; DO 21
        21.10 TY %2 I ":" %5 LAM.C(I) " |"
        21.11 TY %2 I+5 ":" %5 LAM.C(I+5) " |"
        21.12 TY %2 I+10 ":" %5 LAM.C(I+10) " |"
        21.13 TY %2 I+15 ":" %5 LAM.C(I+15) " |"
        21.14 IF I<=3; TY %2 I+20 ":" %5 LAM.C(I+20) " |"
        21.15 IF I=4 ; TY &10
        21.20 IF I<= 4; TY "| " %1 I ":" %5 FPI.C(I) &1
        21.30 IF I<= 2; TY "| " %1 I ":" %5 IRQ.C(I)
        21.40 TY !

% Programs Monitoring
30.05 se nb=0; $se nm=""; dim-i is(8); dim-i rs(8)
30.10 RPC#ARG(1)PGSTAT W_NB[16] W_NM W_IS W_RS
30.15 IF NB = 0 ; TY "No Program found"; end
30.20 F I=1,NB; DO 31
   31.10 TY "'" SUBS(1+(I-1)*4,4+(I-1)*4,NM) "' - "
   31.20 IF IS(I) = 0; TY "Non existent"!;RET
   31.25 IF IS(I) = 2 ; TY "Stopped (exception)" !; RET
   31.30 IF IS(I) = 3 ; TY "Terminated"! ; RET
   31.40 IF IS(I)= 1 ; TY "Started" ; GOTO 31.50
   31.45 TY "--- XMON ERROR --- prog status =" %2 IS(I); RET !
   31.50 TY ", RMS status=" %4 ]]RS(I)
   31.55 IF RS(I) > 256; TY "(Task is waiting)"
   31.60 TY !
30.30 SE I.S =0; SE R.S =0;DIM-I D.R(16); DIM-I A.R(16)
30.35 DIM-I PC(2); SE SR=0; $SE MES.=""
30.40 $ASK "Additional info on which task ? (4 chars mandatory)" nm
30.45 RPC#ARG(1)PGWHAT R_NM W_I.S[16] W_R.S[16] W_D.R W_A.R W_PC W_SR[16] W_MES.
30.50 DO 32

32.05 IF I.S <> 1 ; IF I.S <> 2;TY "No info on that task"!; END
32.10 TY "PC =" ]]PC(1) ]]PC(2)
32.15 IF I.S = 2; TY "Stopped - " MES.
32.16 TY !
32.20 FOR I=0,7;TY &3 "D" %1 I "=" ]]D.R(1+I*2) ]]D.R(2+I*2);IF MOD(I,4)=3;TY !
```

```
32.21 FOR I=0,7;TY &3 "A" %1 I "=" ]]A.R(1+I*2) ]]A.R(2+I*2);IF MOD(I,4)=3;TY !

% CLEAR LAM INTERRUPT
40.10 SE T(1)=8; SE T(2)=[[0028
40.15 GET(ADT(T,1),T,2)
40.20 PUT([[7C,T,2)
40.30 SE V=SCAM(LP,CR,ST,2,25,XQ);DO 6 ;% "Abort" function
40.40 WAIT-T 2
40.50 SE U=0; GET(U,T,2)
40.60 IF T(1) <> 0 OR T(2) <> 0; TY "Failed, address:" ]]T(1) ]]T(2) !

99.10 SAVE (NEW-RMS)DEMO-EPROM-22

SAVE (VOL)TEST-XMON
```

Appendix B

RMS Events

Exception monitor events:

```
struct {
    char    length;     /* always 12 */
    char    code;       /* always 8 */
    long    task;       /* task ID */
    long    dummy_1;    /* Task session */
    char    exc_code;   /* exception code */
    char    exc_type;   /* exception type */
                            /*   1 - task attached */
                            /*   2 - task detached */
                            /*   3 - Exception */
```

ISR events:

```
typedef struct {
    char    length;     /* 10 if error */
    char    code;       /* always 2 */
    long    error_code; /* $FFFFF000 + code */
    long    PC;         /* program counter at error time */
} ISR_event_rec;
```

Appendix C

Exception Codes

- $00 : reserved.
- $01 to $0F : Trap instructions.
- $10 : bus error
- $11 : address error
- $12 : illegal instruction
- $13 : zero divide
- $14 : CHK instruction
- $15 : TRAPV
- $16 : privilege violation
- $17 : line 1010 emulator ($Axxx instructions)
- $18 : line 1111 emulator ($Fxxx instructions)
- $19 & $1A : Not used.
- $1B to $1E : Exception manager codes:
  - $1B – Maximum count reached
  - $1C – Traced 1 instruction
  - $1D – Value change occured
  - $1E – Value equal occured

Exception codes found in ISRs events are identical, in the range $10 to $18.

Appendix D

Monica error codes

Monica error codes are inserted in the "line 1010 emulator" instructions: opcode is: $A1xx, with xx being the error code.

Version 1.0

$00 E_RelErr	Pusher relocation error
$01 E_Return	return to MoniCa monitor
$02 E_BusErr	bus error
$03 E_AdrErr	address error
$04 E_IlgIns	illegal instruction
$05 E_DivZer	zero divide
$06 E_ChkErr	boundary error
$07 E_OvfTrp	overflow trap
$08 E_PriVio	privilege violation
$09 E_TrcErr	trace vector
$0A E_Em1Trp	line 1010 emulator
$0B E_Em2Trp	line 1111 emulator
$0C E_UnxInt	exception vector #
$0D E_NoSpac	no more space for NEW
$0E E_StckLow	initial stack too small
$0F E_StckOv	stack overflow
$10 E_StckCor	stack/heap corrupted
$11 E_SetErr	set element error
$12 E_IntOvf	integer overflow
$13 E_AriOvf	real number overflow
$14 E_EOF	end of file
$15 E_IllAcc	illegal file access
$16 E_IllInt	illegal integer on file
$17 E_InpErr	char. lost/ buffer overfl.
$18 E_SysErr	system error message
$19 E_SubVio	subrange violation
$1A E_ChaVio	character range violation
$1B E_PtrErr	pointer outside range
$1C E_CasErr	case statement error
$1D E_GloJmp	global jump error
$1E E_Halt	halt requested
$1F E_Break	break received
$20 E_Auto	auto vector interrupt
$21 E_BkpFound	breakpoint found
$22 E_NoFunRes	no function result defined
$23 E_BadStack	bad SSP/MSP

Codes $02 to $0C are considered illegal in our context.

Version 1.1

$00 – E_RelErr	Pusher relocation error

$01	–	E_Return	return to MoniCa monitor
$02	–	E_BusErr	bus error
$03	–	E_AdrErr	address error
$04	–	E_IlgIns	illegal instruction
$05	–	E_DivZer	zero divide
$06	–	E_ChkErr	boundary error
$07	–	E_OvfTrp	overflow trap
$08	–	E_PriVio	privilege violation
$09	–	E_TrcErr	trace vector
$0A	–	E_Em1Trp	line 1010 emulator
$0B	–	E_Em2Trp	line 1111 emulator
$0C	–	E_CopVio	coprocessor violation
$0D	–	E_FormEr	format error
$0E	–	E_UniInt	uninitialized interrupt
$0F	–	E_SpuInt	spurious interrupt
$10	–	E_FPCP48	MC 68881 coprocessor
$11	–	E_FPCP49	MC 68881 coprocessor
$12	–	E_FPCP50	MC 68881 coprocessor
$13	–	E_FPCP51	MC 68881 coprocessor
$14	–	E_FPCP52	MC 68881 coprocessor
$15	–	E_FPCP53	MC 68881 coprocessor
$16	–	E_FPCP54	MC 68881 coprocessor
$17	–	E_PMMU56	MC 68851 coprocessor
$18	–	E_PMMU57	MC 68851 coprocessor
$19	–	E_PMMU58	MC 68851 coprocessor
$1A	–	E_UnxInt	exception vector #
$1B	–	E_NoSpac	no more space for NEW
$1C	–	E_StckLow	initial stack too small
$1D	–	E_StckOv	stack overflow
$1E	–	E_StckCor	stack/heap corrupted
$1F	–	E_SetErr	set element error
$20	–	E_IntOvf	integer overflow
$21	–	E_AriOvf	real number overflow
$22	–	E_EOF	end of file
$23	–	E_IllAcc	illegal file access
$24	–	E_IllInt	illegal integer on file
$25	–	E_InpErr	char. lost/ buffer overfl.
$26	–	E_SysErr	system error message
$27	–	E_SubVio	subrange violation
$28	–	E_ChaVio	character range violation
$29	–	E_PtrErr	pointer outside range
$2A	–	E_CasErr	case statement error
$2B	–	E_GloJmp	global jump error
$2C	–	E_Halt	halt requested
$2D	–	E_Break	break received
$2E	–	E_BkpFound	breakpoint found
$2F	–	E_NoFunRes	no function result defined
$30	–	E_BadStack	bad SSP/MSP

Codes $02 to $1A are considered illegal in our context.

List de distribution   *PS/CO/Note 89-2*

V. Adorni
G. Benincasa
P. Burla
L. Casalegno
G. Cuisinier
G. Daems
F. di Maio
A. Gagnaire
F. Giudici
W. Heinze
M. Lelaizant
J. Lewis
L. Mérard
N. de Metz-Noblat
F. Perriollat
U. Raich
Ch. Sere
C.H. Sicard

M. Arruat
J. Bouchéron
P. Fernier
A. Poncet
L. Soby
B. Vandorpe