SYSTEMS PROGRAMMING

Covers input/output programming, exception processing, peripheral device interrupts

Chapter 12	6850 ACIA (Asynchronous Communications Interface Adapter), 68230 PIT (Programmable Interval Timer)
Chapter 13	Exception processing, i.e. service routines and single stepping
Chapter 14	Exception processing and interrupt processing, concurrent programming.

Interrupts and exceptions
Instructions that interrupt ordinary program execution to allow access to system utilities or when certain internally generated conditions (usually errors) occur.

Conditions interrupting ordinary program execution are called exceptions. Usually are caused by sources internal to the 68000. Interrupts are exceptions which are caused by sources external to the 68000.

Exceptions transfer control to the program controlling the system (usually a monitor program or an operating system).

For example,

- user program executes a TRAP instruction for input/output which forces an exception.

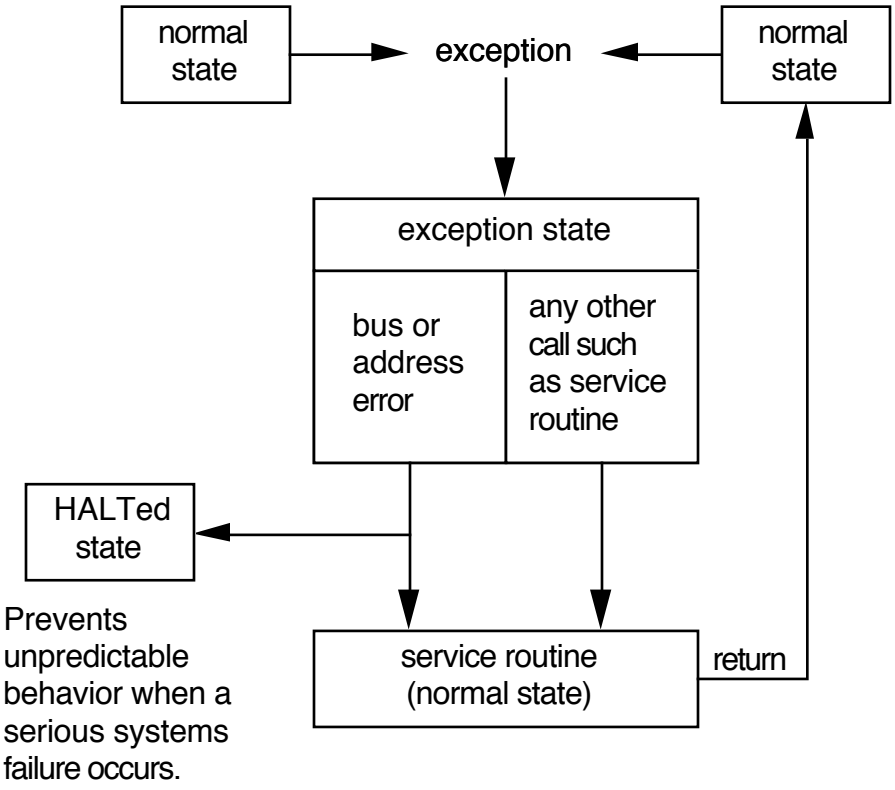
- user program becomes suspended, file input/output is done by monitor program/operating system.

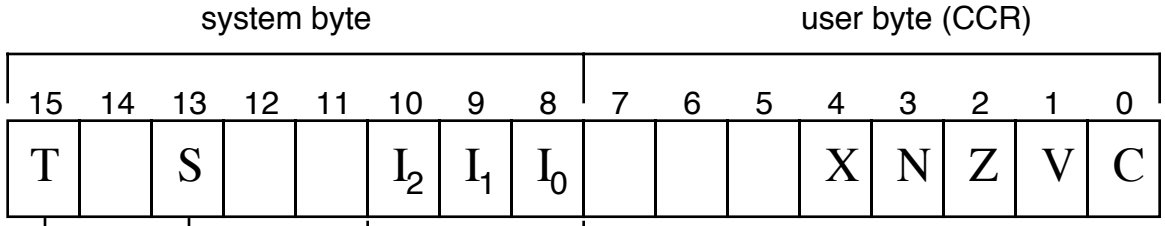
- user program is restarted where it was suspended.

As a result of any exception, the CPU switches from program execution to exception processing, services the exception request, and returns to normal program execution.

The MC68000 makes specific provision for two (actually three) operating states:

- normal state
- exception state
- HALTED state - used to prevent unpredictable behavior when a serious system failure occurs





Interrupt level

Supervisor mode
Allows a priviledged mode of execution which is essential for multi-user environment.

Trace
Sets a post-instruction routine into action.

Supervisor mode — single-user operating systems and monitor programs, all exception handling programs normally run in supervisor mode

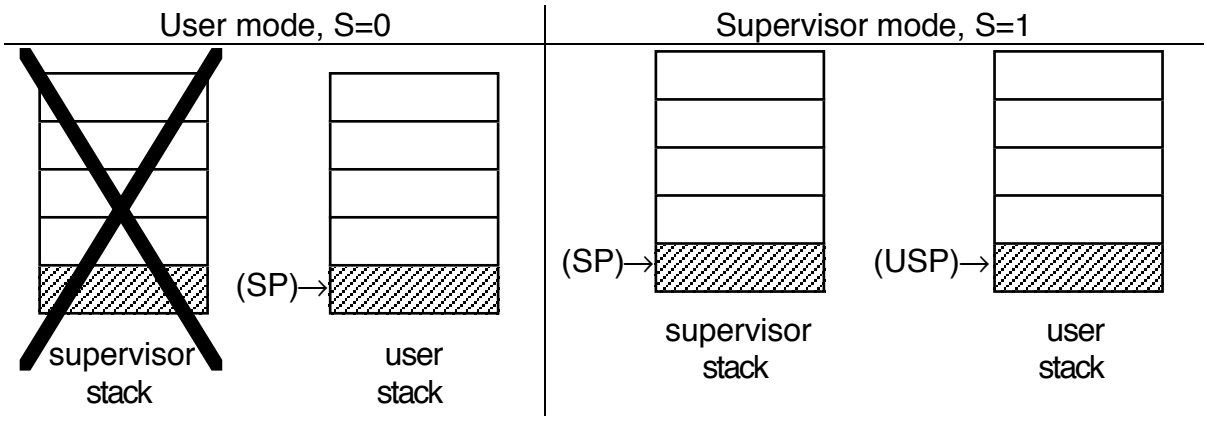
User mode restricted access to the system environment, useful in multi-user environments.

The supervisor bit (bit 13 of the status register) is 1 if the 68000 is in supervisor (priviledged) mode.

There are four priviledged 68000 instructions; all have the entire SR as a destination.

```
MOVE.W    <ea>,SR
ANDI.W    #N,SR
ORI.W#N,SR
EORI.W    #N,SR
```

So that the 68000 does not become confused there are two stacks



Bit 13 of the status register is used to toggle A7 between the user and supervisor modes. USP references the user stack while the 68000 is in supervisor mode.

```
MOVE.L    USP,An
MOVE.L    An,USP
```

are the only instructions that can access the user stack while the 68000 is in supervisor mode. They are both priviledged instructions and transfer only long words (32 bits).

Examples 13.3

```
ADDA.L    D0,USP      ;not a valid instruction

MOVE.L    USP,A0      ;valid instruction, puts USP on system stack
MOVE.L    A0,-(SP)    ;cannot do MOVE.L USP,-(SP) directly
```

How exceptions are processed:

1.	identify the exception	• internal (identified by the CPU) - caused by TRAP instruction , etc. • external (identified by specific signal pins) - caused by hardware assertation
2.	save information about the currently running program	save the status register to a special internal register
3.	initialize the status register (for the exception routine)	set S=1, T=0 (typically), interrupt level for external exceptions
4.	determine the vector number	(from step 1), each exception type can have a unique routine. The MC68000 allows 255 such routines and stores their location (called a vector) addresses in the first 1K of 68000 program memory. This area is called the exception vector table. vector #1 SPECIAL for system start-up vector #2 • • • vector #255 address of exception vector = 4 × exception vector number
5.	save status register and return address	push the current PC (return address for the exception routine) and the saved status register onto the system stack.
6.	set PC to (address of exception vector), i.e. to correct starting address of exception service routine	the exception service routine is typically user created or part of your operating system. You are typically responsible for setting the vector address in the exception vector table.
7.	RTE	This is a priviledged insruction (Return from Exception) and MUST be at the end of each exception service routine. It pops the status register and PC from the supervisor stack. The status register and PC from the exception

routine processing are lost.

Pseudo code exception processing cycle

identify exception vector number
save present status register to internal CPU register
set status register
    begin
    set S bit to 1
    set T bit to 0
    set interrupt level to level of present interrupt
    end
Compute exception vector address
* vector address = 4 $\times$ exception vector number
Save user information onto system stack
    begin
    push PC onto system stack
    push saved status register onto system stack
    end
Set PC to exception vector address

{Execute exception subroutine.}

RTE
* Similar to RTS, pops PC and SR from system stack,

EXCEPTION VECTOR TABLE

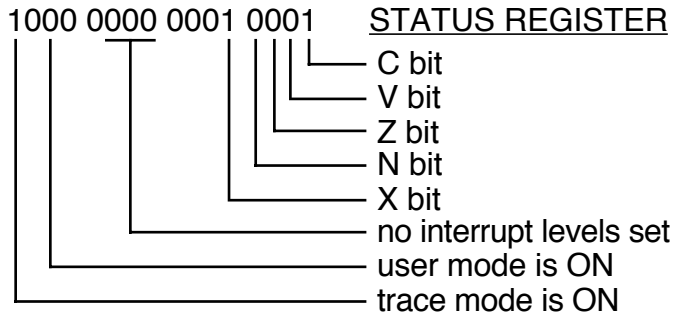
vector number (Decimal)	address (Hex)	assignment
0	0000	RESET: initial supervisor stack pointer (SSP)
1	0004	RESET: initial program counter (PC)
2	0008	bus error
3	000C	address error
4	0010	illegal instruction
5	0014	zero divide
6	0018	CHK instruction
7	001C	TRAPV instruction
8	0020	priviledge violation
9	0024	trace
10	0028	1010 instruction trap
11	002C	1111 instruction trap
12*	0030	not assigned, reserved by Motorola
13*	0034	not assigned, reserved by Motorola
14*	0038	not assigned, reserved by Motorola
15	003C	uninitialized interrupt vector
16-23*	0040-005F	not assigned, reserved by Motorola
24	0060	spurious interrupt
25	0064	Level 1 interrupt autovector
26	0068	Level 2 interrupt autovector
27	006C	Level 3 interrupt autovector
28	0070	Level 4 interrupt autovector
29	0074	Level 5 interrupt autovector
30	0078	Level 6 interrupt autovector
31	007C	Level 7 interrupt autovector
32-47	0080-00BF	TRAP instruction vectors**
48-63	00C0-00FF	not assigned, reserved by Motorola
64-255	0100-03FF	user interrupt vectors

NOTES:
* No peripheral devices should be assigned these numbers
** TRAP #N uses vector number 32+N

Example 13.4 TRACING

```
PC          instruction
            {somewhere in your program}
001FFC      MOVE.W    D0,16(A0)        ;trace service routine at $9000
002000      ADD.W     D2,D3

ORG         $9000
            {code for exception routine}
            RTE
```

The trace exception occurs after instruction is executed. For the purposes of this example, assume SR after MOVE.W is executed is $8011, i.e.



Consider pseudo code:

- Identifies exception vector number. For trace, exception vector number $=9_{10}$.
- Saves user SR to internal register.
- Sets new SR (upper bits only 0010 0000 = $20)
- Computes exception vector address ($9 \times 4 = 36_{10}$ = $24)
- Push user PC, then user SR onto system stack.
- Set PC = the contents of the exception vector address = ($24) = $9000
- Executes the trace routine which prints a message to the screen or printer and then clears the T bit in the user SR presently on the system stack.
- RTE

immediately after the MOVE instruction is executed	just before executing the interrupt service routine	after the RTE
SR: $8011* PC: $002000 The T bit is set indicating that TRACEing is in effect.	SR: $2011 PC: $009000 In special internal register: $8011 Notice that lower bits of SR are not altered by entering ISR.	SR: $0011* PC: $002000 * Recall that the exception routine turned off the T bit.
STACK: $8FFA $8FFC $8FFE (SP) → $9000	STACK: (SP) → $8FFA $8011 * $8FFC $0000 $8FFE $2000 ** $9000 * original SR ** original PC	STACK: $8FFA $8FFC $8FFE (SP) → $9000

HOW DOES THE 68000 START UP?

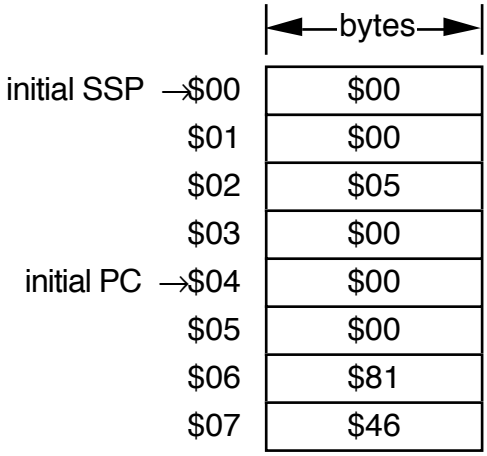
Hardware sets the RESET input pin (This is caused by circuitry external to the 68000). This causes a hardware exception which triggers the “reset exception.”

identifies the exception	RESET (#0)
no currently running program	
initialize the SR	S=1 T=0 I2I1I0 = 1112 (interrupts disabled)
determine vector number	0 in this case
save the SR and return address on the system stack	None, so the return stack must be initialized: (supervisor) SP = ($0) reset vector = [($0) ($4)]
set PC to (address of exception vector)	PC = ($4)

Example 13.5

Press RESET.	
Initialize SR.	SR=$2700
Initialize (SP).	(SP)=$0500
Initialize (PC)	(PC)=$8146
Begin execution of the 68000’s initialization routine at the starting address in memory.	

The memory (i.e. the RESET vector) in this example looks like:



Single board computers do NOT typically have an operating system. They have a simple program called a MONITOR which contains exception service routines whose starting addresses are loaded into the exception vector table at memory locations $8 - $3FF (remember the RESET vector MUST be in the first eight memory locations). Typically, the monitor will service key exceptions such as bus address errors, divide by zero, etc. with specific service routines. All other exceptions are handled by a generic service routine.

The startup sequence is special and consists of the following:

step	action	description
1.	set the status register to $2700	sets supervisor bit to 1, turns trace off, sets interrupt mask to 111
2.	set the Supervisor Stack Pointer to the contents of $0	SSP←($0.L)
3.	set the pc to the contents of $4	pc←($4.L)
4.	start program execution	

Example 13.4.2 SYSTEM INITIALIZATION

Address	exception	name of service routine
$08	bus error	VBUSERR
$0C	address error	VADDRERR
$10	illegal instruction	VILLEGINST
$14	divide by zero	VZERODIV
$18		VCHK
$1C		VTRAPV
$20	priviledged instruction violation	VPRIVINST
$24	trace (single step)	VTRACE
	generic routine	XHANDLE

Typical MONITOR routine:

```
* MONITOR INITIALIZATION ROUTINE
* ASSUMES RESET VECTOR CONTAINS ADDRESS OF INIT AT $4
STARTSP     EQU     $8000                   ;initial stack pointer
                                            value

* EXCEPTION VECTOR ADDRESSES IN SEQUENTIAL ORDER
VBUSERR     EQU     $08
VADDERR     EQU     $0C
VILLEGINST  EQU     $10
VZERODIV    EQU     $14
VCHK        EQU     $18
VTRAPV      EQU     $1C
VPRIVINST   EQU     $20
VTRACE      EQU     $24

* STORE EXCEPTION VECTORS IN THE ADDRESS TABLE

* RESET vector starts here
            ORG     $5000
INIT        LEA     STARTSP,SP              ;initialize SSP
            MOVE.L  #BUSERR,VBUSERR         ;initialize exception
            MOVE.L  #ADDERR,VADDERR         ;vector table
            MOVE.L  #ILLINST,VILLINST
            MOVE.L  #XHANDLE,VZERODIV
            MOVE.L  #XHANDLE,VCHK
            MOVE.L  #XHANDLE,VTRAPV
            MOVE.L  #PRIVIOL,VPRIVINST
            MOVE.L  #TRACE,VTRACE

            LEA     $28,A0                  ;load rest of the
                                            exception table from
                                            address $28 to
                                            $3FC with starting
                                            address of routine
                                            XHANDLE

ABINIT      MOVE.L  #XHANDLE,(A0)+
            CMPA.L  #$400,A0
            BCS.S   TABINIT

MAIN        {This is the mini-operating system and is a program that always
            runs.  It might interpret commands, etc.)
            BRA     MAIN
```

```
* EXCEPTION SERVICE ROUTINES

BUSERR      {put code for routine here}

ADDERR      {put code for routine here}

ILLINST     {put code for routine here}

PRIVIOL     {put code for routine here}

TRACE       {put code for routine here}

XHANDLE                                         ;prints error
                                                message
            MOVEQ   #0,D0                       ;clear D1
            LEA     EXCEPTMSG,A0                ;load location of
                                                message
            JSR     PUTSTRING                   ;print it
            MOVE.L  2(SP),D0                    ;get return address
                                                from system stack
            JSR     PUTHEX                      ;print it
            JSR     NEWLINE
* FLUSH THE RETURN ADDRESS AND SR FROM THE SYSTEM STACK
            ADDQ.W  #6,SP                       ;flush the stack
            BRA     MAIN                        ;return to monitor

EXCEPTMSG   DC.B    'UNEXPECTED EXCEPTION AT ',0
```

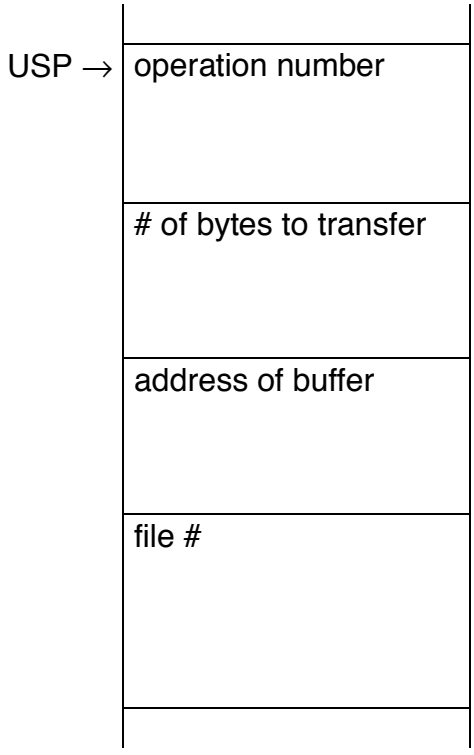
PROGRAM 13.1 TRAP HANDLER

```
VTRAP0      EQU     $80             ;trap #0 exception address,
                                    128(10)
VTRAP1      EQU     $84             ;trap #1 exception address,
                                    132(10)

STARTSP     EQU     $8000           ;initial SP value
STARTUSP    EQU     $4000           ;initial USP value
MONITOR     EQU     $8146           ;address of monitor

NULL        EQU     0
CONSOLE     EQU     0               ;console port

            XREF    INIT,PUTHEX,PUTSTRING,NEWLINE

* RESET VECTOR STARTS HERE
MAIN:       LEA     STARTSP,SP      ;initial stacks
            LEA     STARTUSP,A0
            MOVE.L  A0,USP

            MOVE.L  #TRAP0,VTRAP0   ;initialize exception vectors
            MOVE.L  #TRAP1,VTRAP1

            MOVEQ   #CONSOLE,D7     ;initialize UART specific to
                                    ECB
            JSR     INIT

* start program at address $2000 in user mode
            MOVE.L  #$2000,-(SP)    ;put starting address of user
                                    program on system stack
            MOVE.W  #$0000,-(SP)    ;clear status register
            RTE                     ;start user program

* service routine for TRAP #0.  Only a "file read" routine is simulated.
TRAP0:
            MOVEM.L D0/A0-A1,-(SP)
            MOVE.L  USP,A1          ;A1 points at user stack
            MOVE.L  (A1)+,D0        ;get the operation number
            ASL.L   #2,D0           ;4 byte index to iocalls table
                                    (multiply by 4 for byte offset)
            LEA     IOCALLS,A0      ;get base address of table
            MOVEA.L 0(A0,D0.L),A0   ;(A0) is address of the routine
* SUBROUTINE PUSHS ARGS OFF STACK
```

```
                JSR         (A0)                ;jump to routine
                MOVE.L      A1,USP              ;reset user stack pointer
                MOVEM.L     (SP)+,D0/A0-A1
                RTE

* ROUTINE TO CREATE AND OPEN A  FILE ARE PLACED HERE
CREATE:
OPEN:

* READ ROUTINE DUMPS AND PRINTS PARAMETERS ON USER STACK
READ:           LEA         BYTEREAD,A0         ;get the bytes to read
                JSR         P[UTSTRING
                MOVE.L      (A1)+,D0
                JSR         PUTHEX
                JSR         NEWLINE
                LEA         BUFADDR,A0          ;get address of input buffer
                JSR         PUTSTRING
                MOVE.L      (A1)+,D0
                JSR         PUTHEX
                JSR         NEWLINE
                LEA         FILENUMBER,A0       ;get the system file number
                JSR         PUTSTRING
                MOVE.L      (A1)+,D0
                JSR         PUTHEX
                JSR         NEWLINE
                RTS

* ROUTINES TO WRITE TO A FILE AND CLOSE A FILE ARE PLACED HERE
WRITE:
CLOSE:

TRAP1:          JMP         MONITOR             ;perform the jump in
                                                supervisor mode
* DATA SECTION
IOCALLS         DC.L        CREATE,OPEN,READ,WRITE,CLOSE
BYTEREAD        DC.B        'BYTES TO READ:   ',NULL
BUFADDR         DC.B        'ADDRESS OF INPUT BUFFER:  ',NULL
FILENUMBER      DC.B        'FILE NUMBER:  ',NULL

                END
```

THE FOLLOWING PROGRAM IS ASSEMBLED AND LOADED AT ADDRESS $2000. PROGRAM 13.1 INITIATES EXECUTION OF THE PROGRAM AND PRODUCES THE OUTPUT BELOW:

```
            ORG     $2000
START:
* put input parameters on stack
            MOVE.L  #3,-(SP)      *FILE NUMBER IS 3
            MOVE.L  #BUF,-(SP)    *ADDRESS OF INPUT BUFFER
            MOVE.L  #512,-(SP)    *NUMBER OF BYTES TO READ
            MOVE.L  #2,-(SP)      *READ OPERATION IS 2
            TRAP    #0            *DO THE READ
            TRAP    #1            *TRAP #1 RETURNS CONTROL TO
                                  MONITOR WHEN IN USER MODE

BUF:        DS.B    512
            END
```



<RUN>

OUTPUT:

BYTES TO READ: 00000200
ADDRESS OF INPUT BUFFER: 0000201C
FILE NUMBER: 00000003

PROGRAM 13.2 ERROR HANDLER:

```
VDIV        EQU     $14                 ;divide by zero exception
                                        vector
VTRAPV      EQU     $1C                 ;trap on overflow exception
                                        address
MONITOR     EQU     $8146               ;address of monitor in
                                        ROM
NULL        EQU     0
RTNADDR     EQU     2                   ;offset to return address
STARTSP     EQU     $8000               ;starting SSP

            XREF    INIT,PUTSTRING,PUTHEX,NEWLINE

            ORG     $1000               ;program would be started
                                        by MONITOR program
START:      MOVE.L  #STARTSP,SP         ;initialize supervisor stack
            MOVE.L  #DIVERR,VDIV.W      ;initialize exception vectors
            MOVE.L  #OVRFLERR,VTRAPV.W

* INITIAL ACIA CONSOLE PORT
            MOVEQ   #0,D7
            JSR     INIT

* START UP A USER PROGRAM AT ADDRESS $2000
            MOVE.L  #$2000,-(SP)
            MOVE.W  #$2000,-(SP)
            RTE

* EXCEPTION SERVICE ROUTINES

DIVERR:     MOVEQ   #0,D7               ;the ACIA is the terminal
            LEA     DIVMSG,A0
            JSR     PUTSTRING
            MOVE.L  2(SP),D0            ;load return address into
                                        D0
            JSR     PUTHEX              ;print return address
            JSR     NEWLINE
            JMP     MONITOR             return user to MONITOR
* RTE IS NOT DONE HERE.  JUST ABORT THE PROGRAM
* PROBABLY SHOULD FLUSH STACK

OVRFLERR:   MOVEQ   #7,D1
```

```
                LEA     OVRFLMSG,A0
                JSR     PUTSTRING
                JSR     NEWLINE
                RTE

DIVMSG:         DC.B    'DIVIDE BY ZERO: PC =  ',NULL
OVRFLMSG:       DC.B    'OVERFLOW ',NULL

                ORG     $2000
START:          MOVE.W  #$5000,D0
                ADDI.W  #$4000,D0               ;V will be set
                TRAPV
                DIVS    #0,D0
                END

END
```

THE FOLLOWING PROGRAM IS ASSEMBLED AND LOADED AT ADDRESS $2000. PROGRAM 13.2 INITIATES EXECUTION OF THE PROGRAM AND PRODUCES THE OUTPUT BELOW:

```
<RUN>
OUTPUT:         OVERFLOW
                DIVIDE BY ZERO: PC =  0000200E
```

PROGRAM 13.3 SINGLE STEPPING

```
* CONTROL CHARACTERS
LINEFEED    EQU     $0A
NULL        EQU     $00

* EXCEPTION VECTORS
VTRACE                  EQU     $24
VBUSERROR               EQU     $08
VADDRESSERROR           EQU     $0C
VILLEGALINSTRUCTION     EQU     $10
VPRIVILEDGEVIOLATION    EQU     $20
VTRAP0                  EQU     $80
VTRAP1                  EQU     $84

MONITOR     EQU     $8146

            XREF    INIT,PUTHEX,GETCHAR,NEWLINE
            XREF    ECHOFF,PUTSTRING,GETSTRING

* INITIALIZE THE SUPERVISOR AND USER STACK POINTERS
START:      LEA     $8000,SP
            LEA     $4000,A0
            MOVE.L  A0,USP

* INITIALIZE THE EXCEPTION VECTORS
            MOVE.L  #TRACE,VTRACE.W
            MOVE.L  #FATALERROR,VBUSERROR.W
            MOVE.L  #FATALERROR,VADDRESSERROR.W
            MOVE.L  #FATALERROR,VILLEGAL INSTRUCTION.W
            MOVE.L  #FATALERROR,VPRIVILEDGEVIOLATION.W
            MOVE.L  #OUTPUT,VTRAP0.W
            MOVE.L  #EXIT,VTRAP1.W

* INITIALIZE THE ACIA
            MOVEQ   #0,D7
            JSR     INIT
            JSR     ECHOFF

* START THE PROGRAM AT $3C00
            MOVE.L  #$3C00,-(SP)        ;starting address is $3C00
            MOVE.W  #$8000,-(SP)        ;start program in user
                                        mode, trace on, interrupt
                                        level 0
```

```
                RTE                             ;start the program

ENDRUN
                JMP     MONITOR                 ;return to main monitor

* BUS/ADDRESS ERROR, ILLEGAL INSTRUCTION, PRIVILEDGE VIOLATION,
* TRAP SERVICE ROUTINES

FATALERROR:
                MOVEQ   #0,D7
                LEA     FATALMSG,A0
                JSR     PUTSTRING
                BRA     ENDRUN                  ;return to MONITOR

* USER PROGRAM EXECUTING TRAP #1 CAUSES TRAP TO HERE
EXIT:           BRA     ENDRUN                  ;return to MONITOR

* USER PROGRAM EXECUTING TRAP #0 CAUSES
* TRAP HERE FOR HEX OUTPUT
OUTPUT:         MOVE.L  D7,-(SP)
                MOVEQ   #0,D7
                JSR     NEWLINE
                JSR     PUTHEX                  ;output (D0)
                JSR     NEWLINE
                MOVE.L  (SP)+,D7
                RTE

* PRIMARY ROUTINE - CATCHES TRACE TRAP, DISPLAYS REGISTERS,
* AND HANDLES PROMPT FOR ANOTHER INSTRUCTION

PROGCOUNTER     EQU     2
TSIZE           EQU     6

TRACE:
                MOVEM.L D0-D7/A0-A6,GENREG.W
                MOVE.L  PROGCOUNTER(SP),D0      ;get PC
                MOVEQ   #0,D1                   ;get SR as a long word
                MOVE.W  (SP),D1
                LEA     TSIZE(SP),A0            ;original value of SP
                MOVE.L  USP,A1                  ;get USP
                MOVEM.L D0-D1/A0-A1,REGS.W      ;load PC/SR/SSP/USP to
                                                print
                MOVEQ   #0,D1                   ;output to console port
```

```
                MOVEQ   #1,D2                   ;allow four registers per
                                                line
                MOVEQ   #18,D3                  ;19 entries to print
                LEA     REGS.W,A1               ;saved registers begin at
                                                ADDRESS(SP)
                LEA     REGMSGS.W,A0
REGPL:          JSR     PUTSTRING
                MOVE.L  (A1)+,D0
                JSR     PUTHEX
                ADDA.L  #8,A0
                ADDQ.W  #1,D2                   ;count 4 registers per line
                CMPI.W  #4,D2
                BLE.S   NEXT
                MOVEQ   #1,D2
                JSR     NEWLINE
NEXT:           DBRA    D3,REGPL
                LEA     OPMSG,A0                ;print message about
                                                opcode word
                JSR     PUTSTRING
                MOVE.L  PROGCOUNTER(SP),A0      ;get opcode word of next
                                                instruction
                MOVEQ   #0,D0
                MOVE.W  (A0),D0
                JSR     PUTHEX                  ;print it
                LEA     PROMPT,A0               ;">>" prompt
                JSR     PUTSTRING
                JSR     GETCHAR                 ;read a character from
                                                terminal
                CMPI.B  #LINEFEED,D0            ;carriage return, continue
                BEQ.S   RET
                ANDI.W  #$7FFF,(SP)             ;turn tracing off
RET:            JSR     NEWLINE
                MOVEM.L GENREGS,D0-D7/A0-A6
                RTE                             ;back to user program

REGS:
                DS.L    4                       ;to contain
                                                PC/SR/SSP/USP

GENREGS:        DS.L    15                      ;to contain D0-D7/A0-A6 at
                                                each trace exception

REGMSGS:
        DC.B    '  PC = ',NULL,'  SR =  ',NULL,'  SSP= ',NULL
```

```
        DC.B    'USP = ',NULL,'  D0 =  ',NULL,'  D1= ',NULL
        DC.B    '  D2 = ',NULL,'  D3 =  ',NULL,'  D4= ',NULL
        DC.B    '  D5 = ',NULL,'  D6 =  ',NULL,'  D7= ',NULL
        DC.B    '  A0 = ',NULL,'  A1 =  ',NULL,'  A2= ',NULL
        DC.B    '  A3 = ',NULL,'  A4 =  ',NULL,'  A5= ',NULL
        DC.B    '  A6 = ',NULL'
OPMSG:
        DC.B    LINEFEED,'OPCODE WORD NEXT INSTRUCTION = 'NULL
PROMPT
        DC.B    LINEFEED,LINEFEED,'>> ',NULL
FATALMSG:
        DC.B    LINEFEED,'FATAL ERROR HAS OCCURRED',LINEFEED,NULL

                ORG     $3C00
START:          MOVE.L  #5,-(SP)
                MOVEQ   #5,D5
                LEA     OUTPUT(PC),A3
                MOVE.W  #$001F,CCR
OUTPUT:         MOVE.L  #$55553333,D0
                TRAP    #0
                TRAP    #1

                END
```

The actual program assembles to:

```
                ORG     $3C00
START:
003C00          2F3C 0000 0005      MOVE.L  #5,-(SP)
003C06          7A05                MOVEQ   #5,D5
003C08          47FA 0006 LEA       OUTPUT(PC),A3
003C0C          44FC 001F MOVE.W #$001F,CCR
OUTPUT:
003C10          203C 5555 3333      MOVE.L  #$55553333,D0
003C16          4E40                TRAP    #0
003C18          4E41                TRAP    #1
```

and gives the following output:

```
<RUN>
PC = 00003C06  SR = 00008000  SSP= 00008000   USP=00003FFC
D0 = 0000100D  D1 = 4000544D  D2 = 21FC104D   D3 =00000000
D4 = 0000FC30  D5 = 0000002C  D6 = 00000006   D7 =00000000
A0 = 00004000  A1 = 0000836C  A2 = 00000414   A3 =00000554
A4 = 0000090C  A5 = 00000560  A6 = 00000560
OPCODE WORD NEXT INSTRUCTION = 00007A05
```

```
>>
PC = 00003C08  SR = 00008000  SSP= 00008000   USP=00003FFC
D0 = 0000100D  D1 = 4000544D  D2 = 21FC104D   D3 =00000000
D4 = 0000FC30  D5 = 00000005  D6 = 00000006   D7 =00000000
A0 = 00004000  A1 = 0000836C  A2 = 00000414   A3 =00000554
A4 = 0000090C  A5 = 00000560  A6 = 00000560
OPCODE WORD NEXT INSTRUCTION = 000047FA

>>
PC = 00003C0C  SR = 00008000  SSP= 00008000   USP=00003FFC
D0 = 0000100D  D1 = 4000544D  D2 = 21FC104D   D3 =00000000
D4 = 0000FC30  D5 = 00000005  D6 = 00000006   D7 =00000000
A0 = 00004000  A1 = 0000836C  A2 = 00000414   A3 =00003C10
A4 = 0000090C  A5 = 00000560  A6 = 00000560
OPCODE WORD NEXT INSTRUCTION = 000044FC

>>
PC = 00003C10  SR = 0000801F  SSP= 00008000   USP=00003FFC
D0 = 0000100D  D1 = 4000544D  D2 = 21FC104D   D3 =00000000
D4 = 0000FC30  D5 = 00000005  D6 = 00000006   D7 =00000000
A0 = 00004000  A1 = 0000836C  A2 = 00000414   A3 =00003C10
A4 = 0000090C  A5 = 00000560  A6 = 00000560
OPCODE WORD NEXT INSTRUCTION = 0000203C

>>
PC = 00003C16  SR = 00008010  SSP= 00008000   USP=00003FFC
D0 = 55553333  D1 = 4000544D  D2 = 21FC104D   D3 =00000000
D4 = 0000FC30  D5 = 00000005  D6 = 00000006   D7 =00000000
A0 = 00004000  A1 = 0000836C  A2 = 00000414   A3 =00003C10
A4 = 0000090C  A5 = 00000560  A6 = 00000560
OPCODE WORD NEXT INSTRUCTION = 00004E40

>>
PC = 00000984  SR = 00002010  SSP= 00007FFA   USP=00003FFC
D0 = 55553333  D1 = 4000544D  D2 = 21FC104D   D3 =00000000
D4 = 0000FC30  D5 = 00000005  D6 = 00000006   D7 =00000000
A0 = 00004000  A1 = 0000836C  A2 = 00000414   A3 =00003C10
A4 = 0000090C  A5 = 00000560  A6 = 00000560
OPCODE WORD NEXT INSTRUCTION = 00002F07

>>
PC = 00000982  SR = 00002010  SSP= 00007FFA   USP=00003FFC
D0 = 55553333  D1 = 4000544D  D2 = 21FC104D   D3 =00000000
D4 = 0000FC30  D5 = 00000005  D6 = 00000006   D7 =00000000
A0 = 00004000  A1 = 0000836C  A2 = 00000414   A3 =00003C10
A4 = 0000090C  A5 = 00000560  A6 = 00000560
OPCODE WORD NEXT INSTRUCTION = 000060DC
```

>>

PROGRAM 13.4 ADDRESS ERROR TEST

```
CONSOLEPORT:    EQU     0
LINEFEED:       EQU     $0A
NULL:           EQU     $00
VADDERR:        EQU     $0C
MONITOR:        EQU     $8146

                XREF    INIT, PUTHEX, NEWLINE, PUTSTRING

* INITIALIZE REGISTERS AND CONSOLE PORT
START:          LEA     $8000,SP
                MOVE.L  #ADDRERROR,VADDERR.W
                MOVEQ   #CONSOLEPORT,D7
                JSR     INIT

* TEST PROGRAM
                LEA     $1005, A0
                MOVE.W  2(A0),$7000     * ADDR ERROR -
                                          DATA REF
*               LEA     $1001,A0
*               JMP     (A0)            * ADDR ERROR -
                                          PROGRAM REF

* ADDRESS ERROR SERVICE ROUTINE
PROGCOUNTER:    EQU     10
STATUSREG:      EQU     8
OPCODEWORD:     EQU     6
FAULTADDR:      EQU     2
STATUSWORD:     EQU     0
ASIZE:          EQU     14

ADDRERROR:
                MOVEQ   #CONSOLEPORT,D7
                LEA     ADDERRMSG,A0
                JSR     PUTSTRING
                MOVE.W  OPCODEWORD(SP),D2
                MOVE.L  PROGCOUNTER(SP),A0
* GET VALUE OF PC SAVED ON THE STACK AND SEARCH FOR THE
* OPCODE WORD IN MEMORY
```

```
AGAIN:              CMP.W     -(A0),D2
                    BNE.S     AGAIN
                    MOVE.L    A0,D0
* PRINT PC AT INSTRUCTION START
                    JSR       PUTHEX
                    BSR       SPACES

                    MOVE.W    STATUSREG(SP),D0      * PRINT SR
                    EXT.L     D0
                    JSR       PUTHEX
                    BSR       SPACES
                    MOVE.W    OPCODEWORD(SP),D0
* PRINT OPCODE WORD
                    EXT.L     D0
                    JSR       PUTHEX
                    BSR       SPACES
                    MOVE.L    FAULTADDR(SP),D0
* PRINT ADDRESS ACCESSED  WHEN THE FAULT OCCURRED
                    JSR       PUTHEX
                    BSR       SPACES
                    MOVE.W    STATUSWORD(SP),D0
* PRINT STATUS WORD

                    ANDI.L    #$1F,D0               * MASK OFF ALL
                                                    * UNUSED BITS
                    JSR       PUTHEX
                    JSR       NEWLINE
                    ADDA.W    #ASIZE,SP
                    JMP       MONITOR

SPACES:             MOVE.L    A0,-(SP)
                    LEA       BLANKS,A0
                    JSR       PUTSTRING
                    MOVE.L    (SP)+,A0
                    RTS

ADDERRMSG:
                    DC.B      'ADDRESS ERROR:' ,LINEFEED
                    DC.B      'PC              SR              OPCODE WORD '
                    DC.B      'BAD ADD STATUS WORD',LINEFEED,NULL
BLANKS:             DC.B      '      ',NULL

                    END
```

```
<RUN>
ADDRESS ERROR:
PC          SR          OPCODE WORD   BAD ADDR      STATUS WORD
0000091C    00002004    000033E8      00001007      00000015
```

PROGRAM 13.5 SIZING MEMORY

```
XREF INIT,PUTHEX, PUTSTRING, NEWLINE

VBUSERROR:  EQU         $08
NULL:       EQU         $00
MONITOR:    EQU         $8146

* INITIALIZE REGISTERS, EXCEPTION VECTOR, AND CONSOLE PORT

START:
        LEA     $1000,SP        * LET STACK GROW DOWN FROM $1000
        MOVE.L  #ENDMEM,VBUSERROR.W
        MOVEQ   #0,D7           * OUTPUT TO CONSOLE UART
        JSR     INIT            ;initialize UART
        LEA     $1000,A0

* ROUTINE TO TEST MEMORY SIZE AND GENERATE A BUS ERROR

SIZE:  MOVE.W #7,(A0)+          * WRITE DATA  INTO MEMORY
        BRA.S   SIZE            * LOOP UNTIL BUS ERROR

* BUS ERROR EXCEPTION SERVICE ROUTINE

ENDMEM:
        MOVE.L  A0,D0           * STORE FIRST ADDRESS PAST
                                * RAM MEMORY IN A0
        LEA     MSG,A0
        JSR     PUTSTRING
        SUBQ.L  #8,D0           * DELETE STORAGE FOR RESET VECTOR
        JSR     PUTHEX
        JSR     NEWLINE
        JMP     MONITOR         ;reset the O/S

MSG:  DC.B      'BYTES OF AVAILABLE  RAM: ',NULL

        END
```

<RUN> OUTPUT: BYTES OF AVAILABLE RAM: 00007FF8

A final comment about address and bus errors is necessary. If an address or bus error occurs during exception

processing for a bus error, address error, or reset, the processor is halted. Only the external RESET signal can restart a halted processor.

```
PROBLEM 13.8
V1111EMULATOR       EQU                         $2C
VTRACE      EQU     $24
VADDRESS    EQU     $0C

            XREF    PUTHEX,NEWLINE

START:      LEA     $8000,SP
            MOVE.L  #S1111,V1111EMULATOR.W
            MOVE.L  #TRACE,VTRACE.W
            MOVE.L  #ADDRESSERROR,VADDRESS.W

            DC.W    $FFFF
            ORI.W   #$8000,SR
            MOVE.W  D0,D1
            ANDI.W  #$F000,D1
            ANDI.W  #$7FFF,SR

            MOVE.W  #7,$7FFF

PCOUNTER    EQU     14

S1111:      MOVEM.L D0-D1/A0,-(SP)
            MOVEA.L PCOUNTER(SP),A0
            MOVE.W  (A0),D0
            ANDI.W  #$0FFF,D0
            MOVEQ   #0,D1
            JSR     PUTHEX
            JSR     NEWLINE
            ADDI.L  #2,PCOUNTER(SP)
            MOVEM.L (SP)+,D0-D1/A0
            RTE

TRACE       MOVEM.L D0-D1/A0,-(SP)
            MOVEA.L PCOUNTER(SP),A0
            MOVE.W  (A0),D0
            MOVEQ   #0,D1
            JSR     PUTHEX
            JSR     NEWLINE
            MOVEM.L (SP)+,D0-D1/A0
            RTE

PCADDERR    EQU     10
STATUSREGISTER      EQU                         8
```

```
OPCODEWORDEQU     6
FAULTADDRESS      EQU             2
STATUSWORD EQU    0
ASIZE     EQU     14

ADDRESSERROR:
          MOVEQ   #0,D1
          MOVE.L  FAULTADDRESS(SP),D0
          JSR     PUTHEX
          JSR     NEWLINE
          MOVE.W  OPCODEWORD(SP),D0
          EXT.L   D0
          JSR     PUTHEX
          JSR     NEWLINE
          MOVE.W  STATUSREGISTER(SP),D0
          EXT.L   D0
          JSR     PUTHEX
          JSR     NEWLINE
          MOVE.L  PCADDERR(SP),D0
          JSR     PUTHEX
          JSR     NEWLINE

IDLE:     BRA.S   IDLE

          END
```

PROBLEM 13.9

```
V1010EMULATOR           EQU                     $28
VTRAP0          EQU     $80
VTRACE          EQU     $24

                XREF    PUTHEX,NEWLINE

START:          LEA     $8000,SP
                MOVE.L  #EMU,V1010EMULATOR.W
                MOVE.L  #PR,VTRAP0.W
                MOVE.L  #TRACE,VTRACE.W

                DC.W    $A000

                ORI.W   #$8000,SR
                MOVE.W  #1,D0
                MOVE.W  #2,D0
                MOVE.W  #3,D0
                ANDI.W  #$7FFF,SR

ID:             BRA.S   ID

PROGCOUNTER:            EQU                     $14

EMU:            MOVEM.L D0-D1/A0,-(SP)
                MOVEQ   #0,D1
                MOVE.L  PROGCOUNTER(SP),A0
                MOVEQ   #0,D0
                MOVE.W  (A0),D0
                JSR     PUTHEX
                JSR     NEWLINE
                ADDI.L  #2,PROGCOUNTER(SP)
                MOVEM.L (SP)+,D0-D1/A0
                RTE

TRACE:          TRAP    #0
                RTE

PR:             MOVE.L  D1,-(SP)
                MOVEQ   #0,D1
                JSR     PUTHEX
                JSR     NEWLINE
                MOVE.L  (SP)+,D1
```

```
RTE

END
```

PROBLEM 13.11

Specify what happens when the following code segment runs on a 32K system.

```
BUSERROR    EQU     $08

START:      MOVE.L  #BERR,BUSERROR.W    ;load bus error
                                        exception vector
            LEA     $8000,SP            ;start system stack
                                        at $8000 (32K)
            MOVE.W  #7,6(SP)            ;put something past
                                        32K - GENERATES
                                        BUS ERROR
                                        EXCEPTION

BERR:                                   ;bus error service
                                        routine
            LEA     26(SP),SP           ;loads address into
                                        A7 which is beyond
                                        32K, does not
                                        causes
                                        EXCEPTION
            RTE                         ;when RTE causes
                                        stack access, the
                                        value of SP causes
                                        another BUS
                                        ERROR
                                        EXCEPTION -
                                        68000 HALTS
```