MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

Order this document
as DC408/D

DC408

MC88110 Single Stepping Code Example

By John Ralston
European High-end Microprocessor Applications Group
Motorola Ltd, East Kilbride

INTRODUCTION

This design concept describes how to implement a single step function on the MC88110 microprocessor that requires no additional hardware. The reader is assumed to have a knowledge of the MC88110 programming model and instruction syntax (see the MC88110 User's Manual, Reference MC88110UM/AD). This method uses the MC88110 feature of trapping to supervisor mode when an illegal opcode instruction is encountered. The main element of the method is a routine which performs MC88110 instructions by either executing them or emulating their actions. There is no limitation on the type of code being stepped. Both User and Supervisor code are catered for as the exception table is under control of the emulation program. This allows stepping of all the supervisor instructions including control register changes. These use a duplicate set of control registers. Exception handlers for traps, etc can also be emulated. This is done by intercepting the exception and building the exception state for the code being stepped. The vector table provided by the code being stepped is used to determine the address to be continue with. There are a series of links to debug monitor routines. These monitor routines are not included in this document. They provide, on an instruction by instruction basis, the disassembly of the instruction, register display, and the checking for code break points. A further link is provided to allow several instructions to be stepped before returning control to the monitor program.

NOTICE ABOUT WORKING PROTOTYPES

The hardware and software design described in this document is not a Motorola product nor is it intended to be a future Motorola product. The design is intended as a proof-of-principle example to Motorola customers that demonstrates the application of Motorola's semiconductor products. Any working prototype based on this design supplied by Motorola to a customer, whether sold, loaned or given, is intended solely for engineering evaluation and on a limited support basis. The hardware and software design is not guaranteed to work in any given application.

METHOD OF OPERATION

The Single Stepping Control Program (SSCP) sets up an area of memory with illegal opcodes, called the Trace code execution area, and directs the Unimplemented Opcode Exception vector to itself. (Note: all other exception vectors should point to the relevant SSCP handler.) The SSCP then builds the machine state, General Register File (GRF), and eXtended Register File (XRG) for the program which will be single stepped, called the Trace program. The machine state consists of the PSR (Processor Status Register), NIP (Next Instruction Pointer), and XIP (eXecuting Instruction Pointer).

Figure 1 gives a pictorial version of the memory areas used for the Trace program. The flow chart of the initialisation routine is shown in Figure 2. At this point the single stepping can begin. This is performed by the emulation part of the SSCP. The emulation routine flow chart is shown in Figure 3. The SSCP takes each instruction in turn from the Trace Code area. If the instruction is not one that causes a program flow change, it is copied into the Trace code execution area, (illegal opcodes now surround the instruction). A 'return from exception' is performed with the address of the copied instruction. The instruction is executed and the illegal opcode traps back to the SSCP to allow any operations to be performed on the Trace code machine state, etc. If the instruction obtained from the Trace Code area could cause a change of flow (for example unconditional/conditional branches) then the SSCP determines if the branch is to be taken and changes the machine state for the Trace program accordingly. MC88110 delayed branches are also catered for.

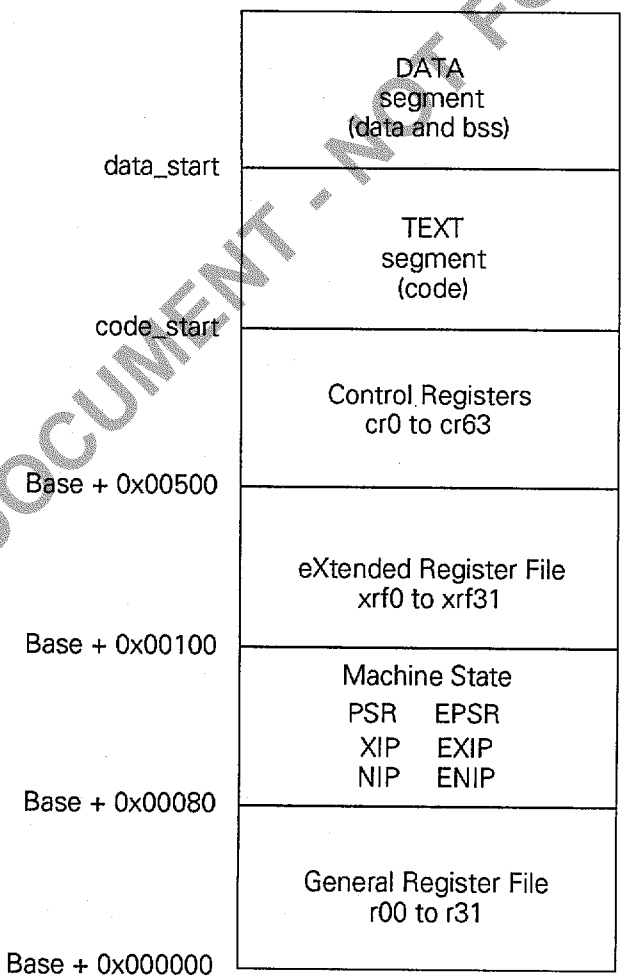

Figure 1. An example
Memory Layout of parameter block for code being stepped

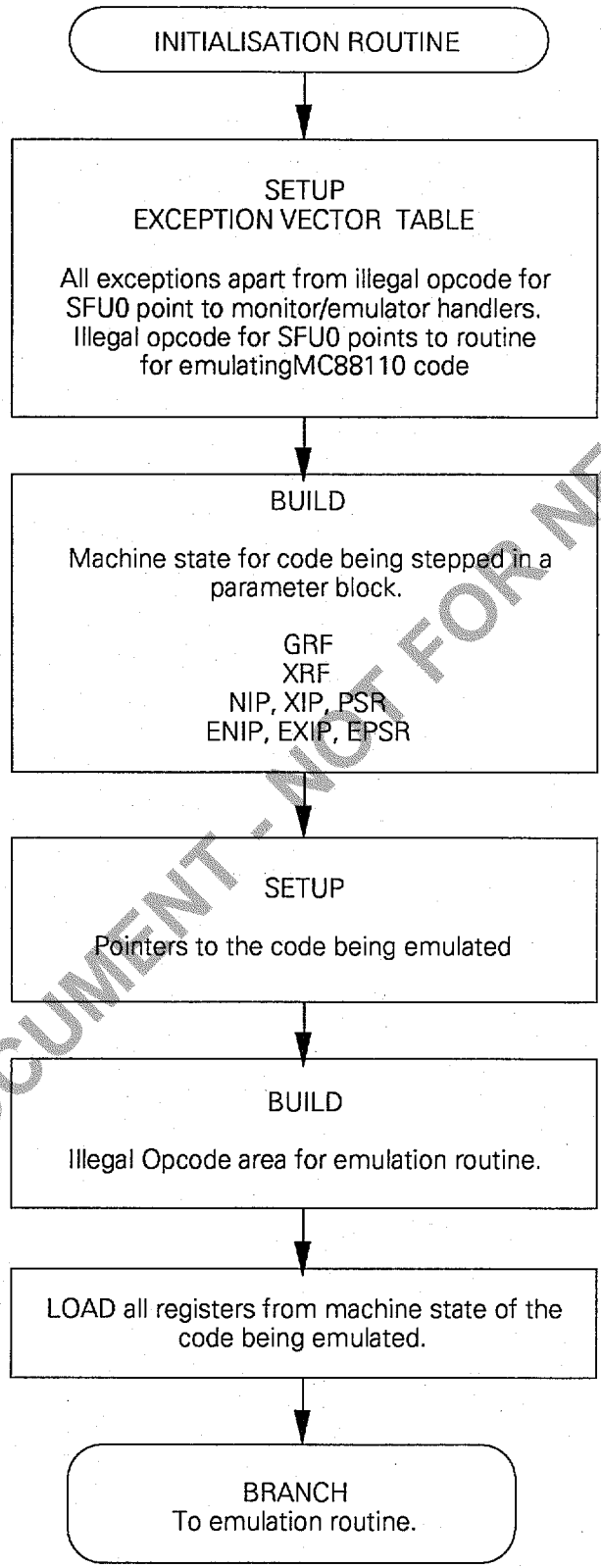

Figure 2. Initialisation Flow Chart (init.c)

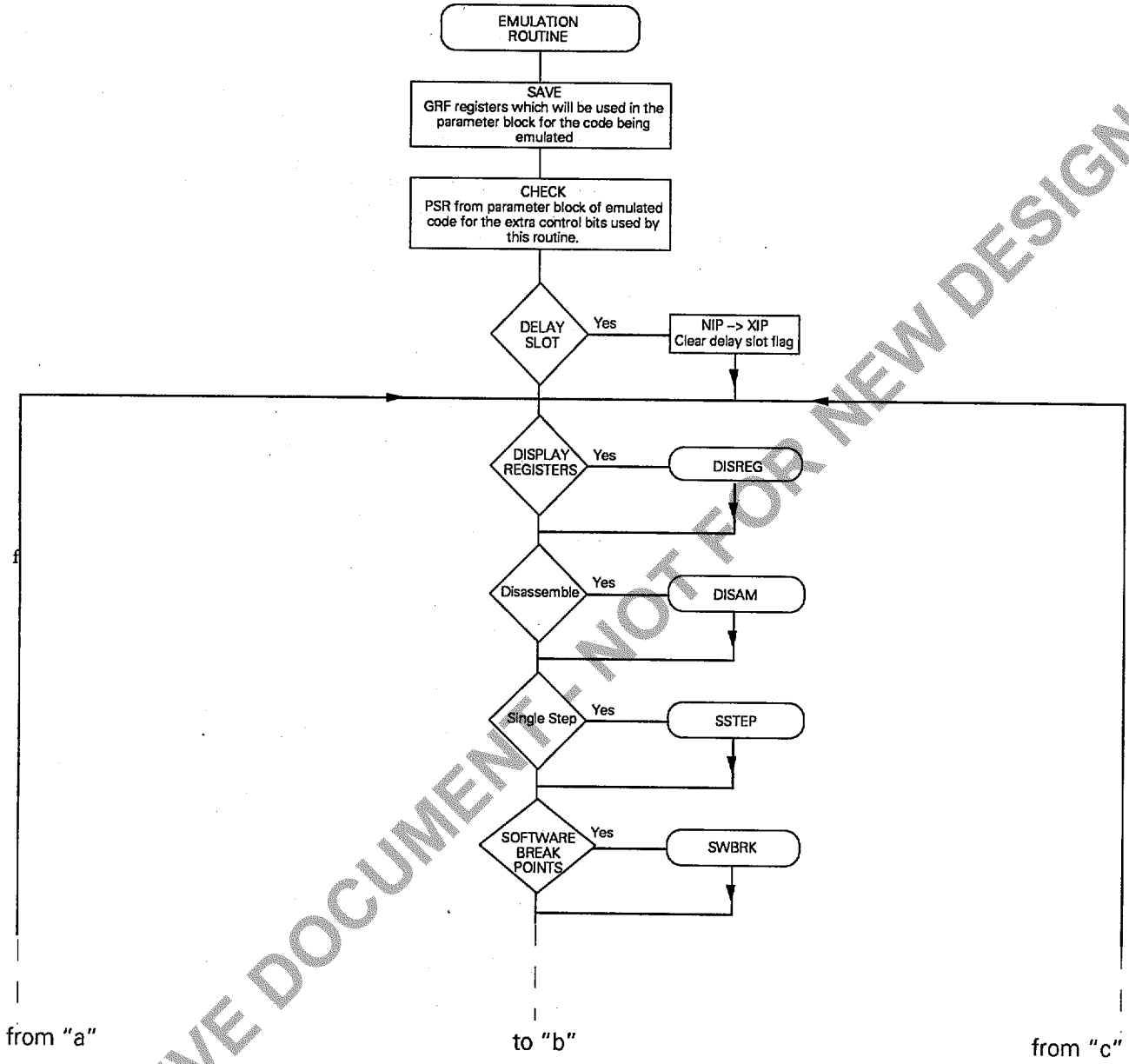

Figure 3a. Upper half of Emulation Routine Flow Chart (emulate.c)

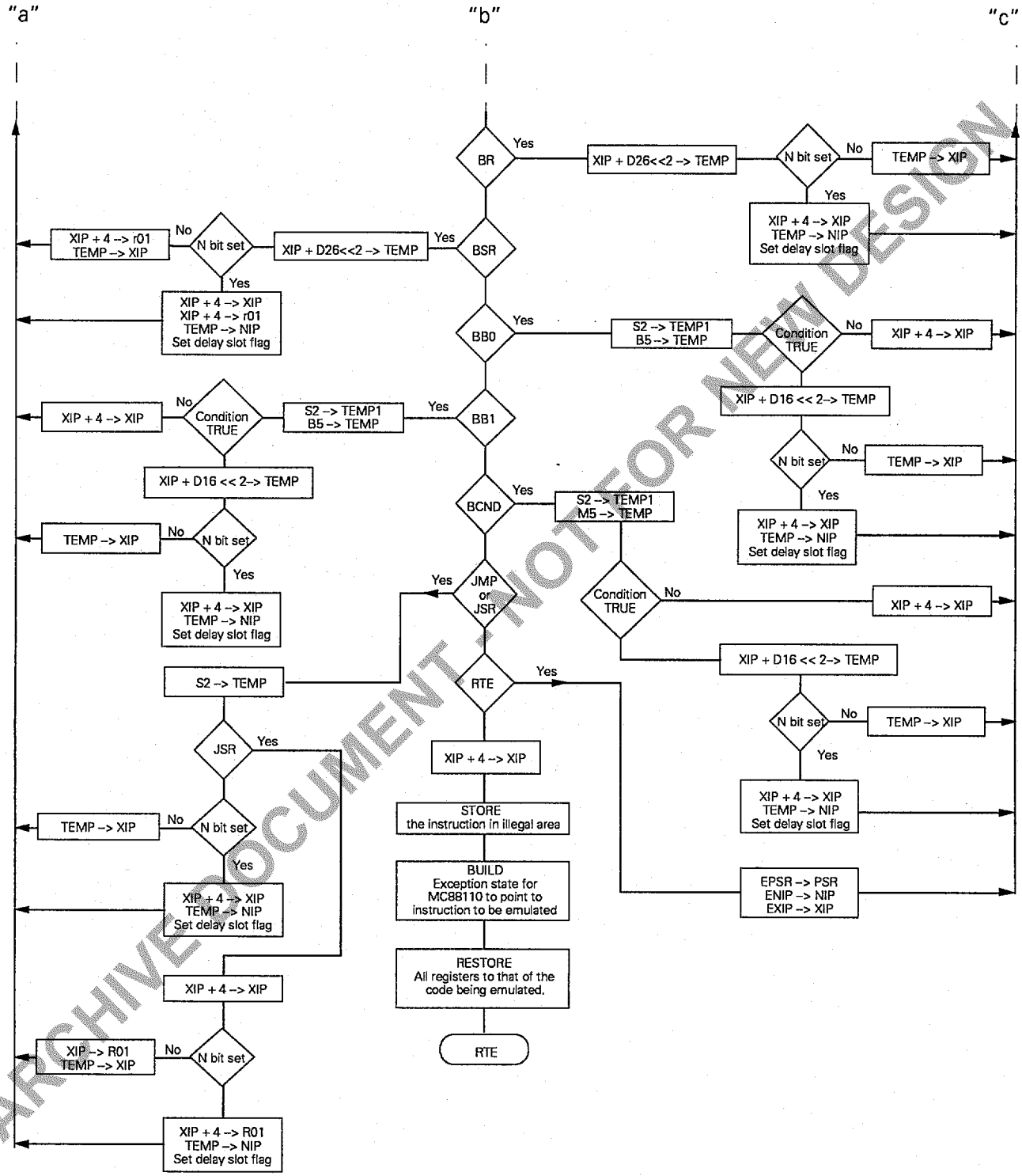

Figure 3a. Lower half of Emulation Routine Flow Chart (emulate.c)

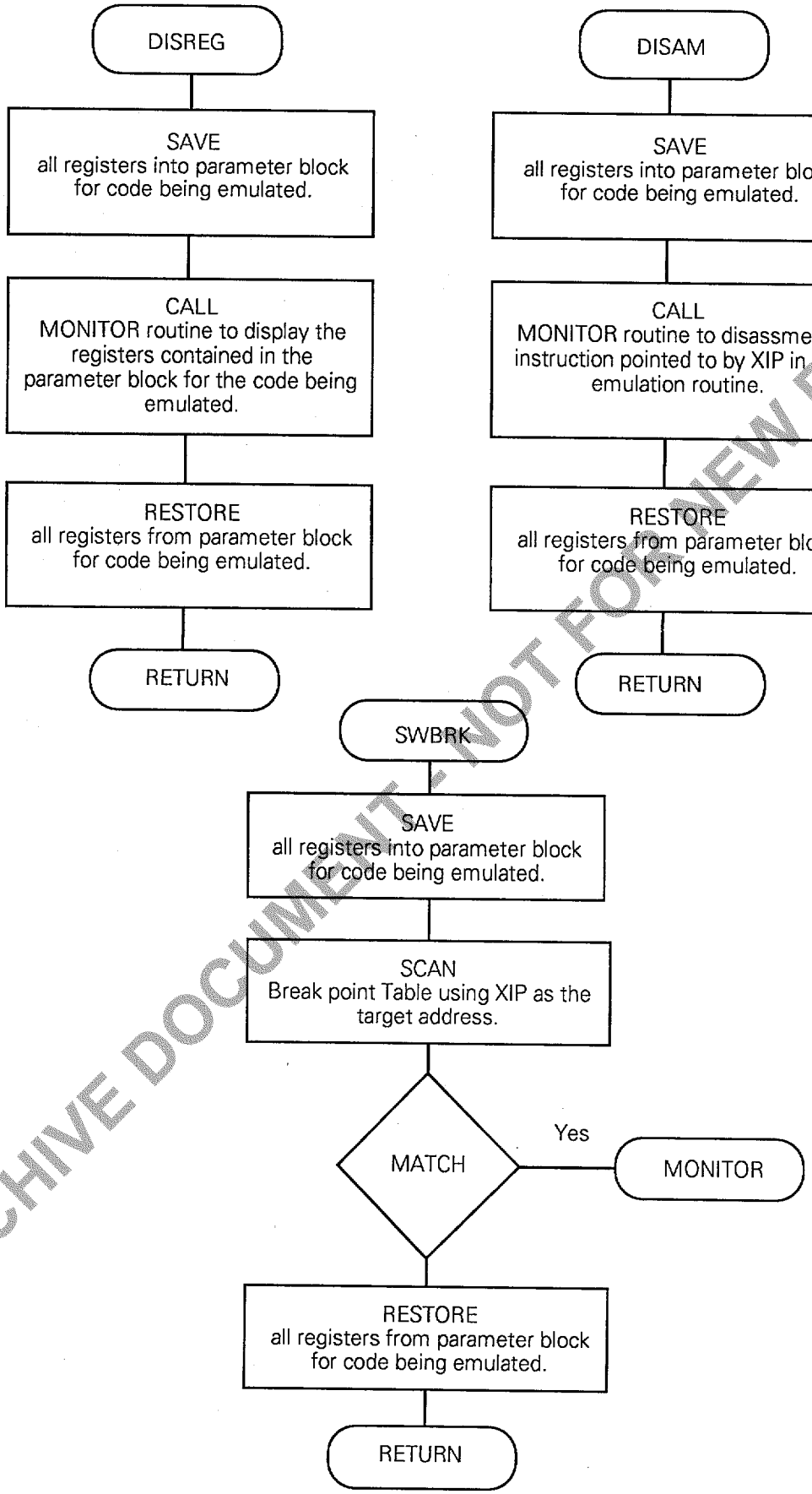

Figure 3b. Emulation Routine MONITOR interface Flow Charts

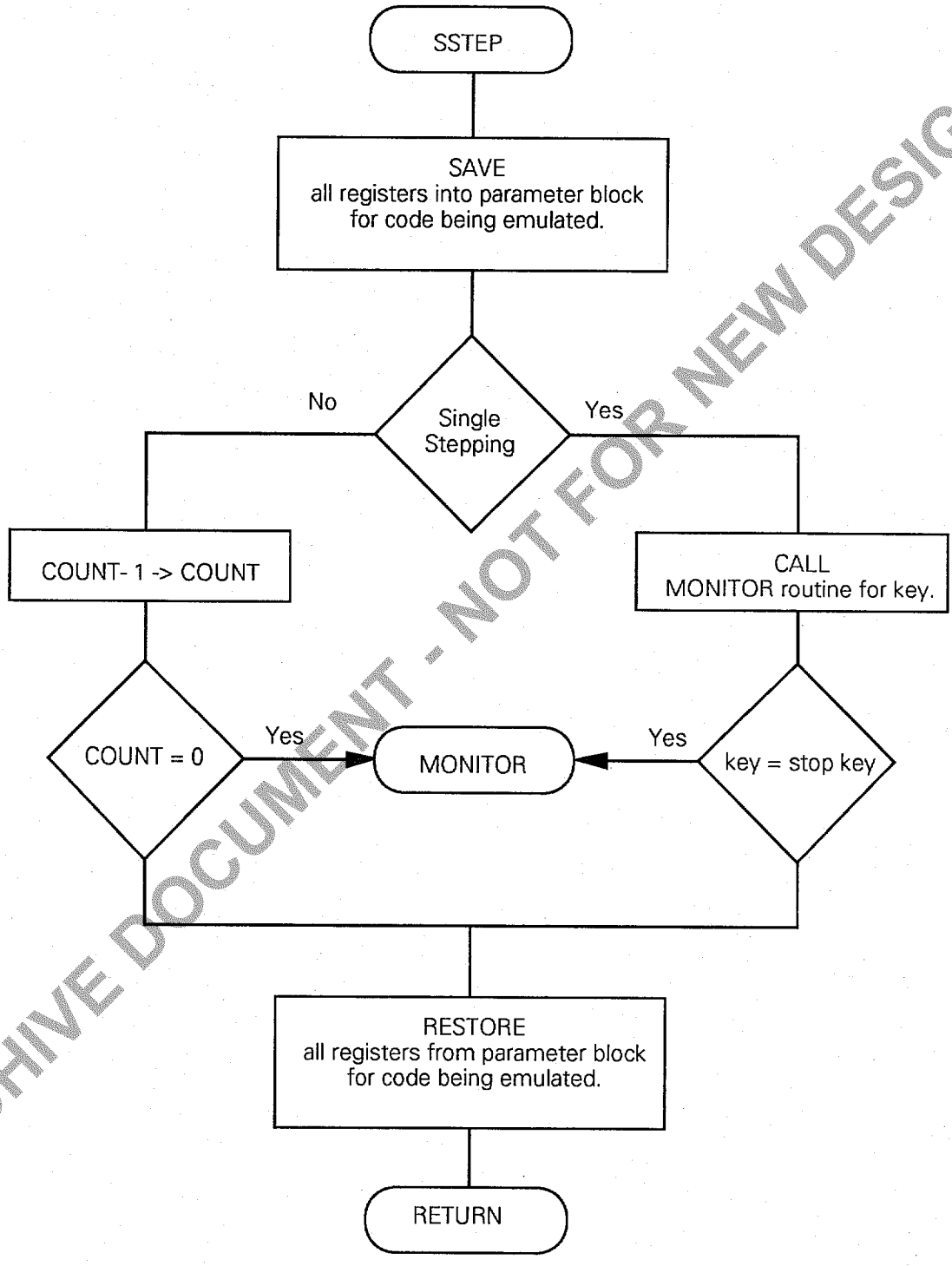

Figure 3b. Emulation Routine MONITOR interface Flow Charts (continued)

CODE EXAMPLE

The following is a Preliminary version of the Single Stepping code for the MC88110.

WARNING: This code has not been tested or debugged

The files used to make up the code are:

vector.s - The vector table for the MC88110

init.c - Initialises the machine state etc for Trace Program

emulate.c - Main part of the Single Step Control Program

storage.h - Header file with variable definitions.

The development package used was the XSIM simulator (see the RISCit User's Guide for details). To generate object code the following command sequence was used in a SYS VRel 3 Unix environment, in conjunction with the MC88110 Development tools:

a) as110 - assembler for MC88110

b) ld110 - ELF linker from release 4 of UNIX

c) cpp - Standard C preprocessor

Command sequence:

```
as110          vector.s vector.o
cpp            init.c front.s
as110          front.s front.o
cpp            emulate.c emulate.s
as110          emulate.s emulate.o
ld110 -M single.map -o single.out     vector.o front.o emulate.o
```

The map file used for the ELF format linker is given below:

```
#   MAP File to link for Non Unix environment
#   Used for linkage of ELF format
#
text=LOAD ?RX V0x00000000 P0x00000000;
text:$PROGBITS ?A!W;

data=LOAD ?RW;
data:$PROGBITS ?AW;
data:$NOBITS ?AW;

note=NOTE;
note:$NOTE;
#   END MAP FILE
```

```
;*************************************************************************
;
;               PROGRAM NAME:     storage.h
;                    VERSION:     01
;              LAST MODIFIED:     6 March 1991
;                   LANGUAGE:     MC88110 assembler uses the cpp for macro and defs
;                     AUTHOR:     jwr (EKB)
;                    COMPANY:     Motorola LTD Copyright (C)
;              TARGET SYSTEM:     MC88110
;                ENVIRONMENT:     Low level debugger (standalone)
;                DESCRIPTION:     Definition file used for single stepping code
;
;               DEPENDENCIES:     None
;
;
;                    HISTORY:
;
;*************************************************************************

#define       ofreg01 (4*1)         /*  r01 offset in save table                */
#define       ofreg02 (4*2)         /*  r02 offset in save table                */
#define       ofreg03 (4*3)         /*  r03 offset in save table                */
#define       ofreg04 (4*4)         /*  r04 offset in save table                */
#define       ofreg05 (4*5)         /*  r05 offset in save table                */
#define       ofreg06 (4*6)         /*  r06 offset in save table                */
#define       ofreg07 (4*7)         /*  r07 offset in save table                */
#define       ofreg08 (4*8)         /*  r08 offset in save table                */
#define       ofreg09 (4*9)         /*  r09 offset in save table                */
#define       ofreg10 (4*10)        /*  r10 offset in save table                */
#define       ofreg11 (4*11)        /*  r11 offset in save table                */
#define       ofreg12 (4*12)        /*  r12 offset in save table                */
#define       ofreg13 (4*13)        /*  r13 offset in save table                */
#define       ofreg14 (4*14)        /*  r14 offset in save table                */
#define       ofreg15 (4*15)        /*  r15 offset in save table                */
#define       ofreg16 (4*16)        /*  r16 offset in save table                */
#define       ofreg17 (4*17)        /*  r17 offset in save table                */
#define       ofreg18 (4*18)        /*  r18 offset in save table                */
#define       ofreg19 (4*19)        /*  r19 offset in save table                */
#define       ofreg20 (4*20)        /*  r20 offset in save table                */
#define       ofreg21 (4*21)        /*  r21 offset in save table                */
#define       ofreg22 (4*22)        /*  r22 offset in save table                */
#define       ofreg23 (4*23)        /*  r23 offset in save table                */
#define       ofreg24 (4*24)        /*  r24 offset in save table                */
#define       ofreg25 (4*25)        /*  r25 offset in save table                */
#define       ofreg26 (4*26)        /*  r26 offset in save table                */
#define       ofreg27 (4*27)        /*  r27 offset in save table                */
#define       ofreg28 (4*28)        /*  r28 offset in save table                */
#define       ofreg29 (4*29)        /*  r29 offset in save table                */
#define       ofreg30 (4*30)        /*  r30 offset in save table                */
#define       ofreg31 (4*31)        /*  r31 offset in save table                */
/*
              Machine State
*/
#define       ofnip (4*32)          /*  NIP trace program next instruction pointer  */
#define       ofxip (4*33)          /*  XIP trace programs instruction pointer      */
#define       ofpsr (4*34)          /*  PSR trace programs status register          */
#define       ofepsr (4*35)         /*  EPSR trace programs exception PSR           */
#define       ofenip (4*36)         /*  ENIP trace programs excepetion NIP          */
#define       ofexip (4*37)         /*  EXIP trace programs excepetion XIP          */
/*
```

```
                Control Register Storage
*/
#define         ofcr0 (4*38)            /*  cr0 offset in save table            */
#define         ofcr1 (4*39)            /*  cr1 offset in save table            */
#define         ofcr2 (4*40)            /*  cr2 offset in save table            */
#define         ofcr3 (4*41)            /*  cr3 offset in save table            */
#define         ofcr4 (4*42)            /*  cr4 offset in save table            */
#define         ofcr5 (4*43)            /*  cr5 offset in save table            */
#define         ofcr6 (4*44)            /*  cr6 offset in save table            */
#define         ofcr7 (4*45)            /*  cr7 offset in save table            */
#define         ofcr8 (4*46)            /*  cr8 offset in save table            */
#define         ofcr9 (4*47)            /*  cr9 offset in save table            */
#define         ofcr10 (4*48)           /*  cr10 offset in save table           */
#define         ofcr11 (4*49)           /*  cr11 offset in save table           */
#define         ofcr12 (4*50)           /*  cr12 offset in save table           */
#define         ofcr13 (4*51)           /*  cr13 offset in save table           */
#define         ofcr14 (4*52)           /*  cr14 offset in save table           */
#define         ofcr15 (4*53)           /*  cr15 offset in save table           */
#define         ofcr16 (4*54)           /*  cr16 offset in save table           */
#define         ofcr17 (4*55)           /*  cr17 offset in save table           */
#define         ofcr18 (4*56)           /*  cr18 offset in save table           */
#define         ofcr19 (4*57)           /*  cr19 offset in save table           */
#define         ofcr20 (4*58)           /*  cr20 offset in save table           */
#define         ofcr21 (4*59)           /*  cr21 offset in save table           */
#define         ofcr22 (4*60)           /*  cr22 offset in save table           */
#define         ofcr23 (4*61)           /*  cr23 offset in save table           */
#define         ofcr24 (4*62)           /*  cr24 offset in save table           */
#define         ofcr25 (4*63)           /*  cr25 offset in save table           */
#define         ofcr26 (4*64)           /*  cr26 offset in save table           */
#define         ofcr27 (4*65)           /*  cr27 offset in save table           */
#define         ofcr28 (4*66)           /*  cr28 offset in save table           */
#define         ofcr29 (4*67)           /*  cr29 offset in save table           */
#define         ofcr30 (4*68)           /*  cr30 offset in save table           */
#define         ofcr31 (4*69)           /*  cr31 offset in save table           */
#define         ofcr32 (4*70)           /*  cr32 offset in save table           */
#define         ofcr33 (4*71)           /*  cr33 offset in save table           */
#define         ofcr34 (4*72)           /*  cr34 offset in save table           */
#define         ofcr35 (4*73)           /*  cr35 offset in save table           */
#define         ofcr36 (4*74)           /*  cr36 offset in save table           */
#define         ofcr37 (4*75)           /*  cr37 offset in save table           */
#define         ofcr38 (4*76)           /*  cr38 offset in save table           */
#define         ofcr39 (4*77)           /*  cr39 offset in save table           */
#define         ofcr40 (4*78)           /*  cr40 offset in save table           */
#define         ofcr41 (4*79)           /*  cr41 offset in save table           */
#define         ofcr42 (4*80)           /*  cr42 offset in save table           */
#define         ofcr43 (4*81)           /*  cr43 offset in save table           */
#define         ofcr44 (4*82)           /*  cr44 offset in save table           */
#define         ofcr45 (4*83)           /*  cr45 offset in save table           */
#define         ofcr46 (4*84)           /*  cr46 offset in save table           */
#define         ofcr47 (4*85)           /*  cr47 offset in save table           */
#define         ofcr48 (4*86)           /*  cr48 offset in save table           */
#define         ofcr49 (4*87)           /*  cr49 offset in save table           */
#define         ofcr50 (4*88)           /*  cr50 offset in save table           */
#define         ofcr51 (4*89)           /*  cr51 offset in save table           */
#define         ofcr52 (4*90)           /*  cr52 offset in save table           */
#define         ofcr53 (4*91)           /*  cr53 offset in save table           */
#define         ofcr54 (4*92)           /*  cr54 offset in save table           */
#define         ofcr55 (4*93)           /*  cr55 offset in save table           */
#define         ofcr56 (4*94)           /*  cr56 offset in save table           */
#define         ofcr57 (4*95)           /*  cr57 offset in save table           */
#define         ofcr58 (4*96)           /*  cr58 offset in save table           */
#define         ofcr59 (4*97)           /*  cr59 offset in save table           */
```

```
#define     ofcr60 (4*98)       /*  cr60 offset in save table                  */
#define     ofcr61 (4*99)       /*  cr61 offset in save table                  */
#define     ofcr62 (4*100)      /*  cr62 offset in save table                  */
#define     ofcr63 (4*101)      /*  cr63 offset in save table                  */
#define     delta    (4*102)    /* Delta in location of the emulation code run time
                                    address and the location used by the parameter
                                    block                                      */

/*
          Condition Bits  (Monitor Use)
*/
#define     bdis                16      /* Display registers                   */
#define     bdasm               17      /* Disassemble instruction             */
#define     bsing               18      /* Single-step                         */
#define     bsbrkr              19      /* Software Break points on            */
#define     bnx                 20      /* Delayed branch slot                 */
#define     bitn                21      /* Iterations not single               */
```

```
;*********************************************************************
;
;               PROGRAM NAME:     init.c
;                    VERSION:     01
;              LAST MODIFIED:     6 March 1991
;                   LANGUAGE:     MC88110 assembler uses the cpp for macro and defs
;                     AUTHOR:     jwr (EKB)
;                    COMPANY:     Motorola LTD Copyright (C)
;              TARGET SYSTEM:     MC88110
;                ENVIRONMENT:     Low level debugger (standalone)
;                DESCRIPTION:     Front end to the debuger.
;
;               DEPENDENCIES:     Requires other standalone code.
;
;
;                    HISTORY:
;
;*********************************************************************

#include          "storage.h"

        text
        global    startup

startup:                                    ; pointed to by either monitor or reset
                                            ; setup PSR
        or.u      r02,r0,hi16(0xa00002fa)
        or        r02,r02,lo16(0xa00002fa)
        stcr      r02,cr1                   ; write PSR
                                            ; invalidate caches
                                            ; invalidate BTC
                                            ; invalidate ATC entries
                                            ; setup s/w pointers
        or.u      r02,r0,hi16(savetab)      ; setup save table address
        or        r02,r02,lo16(savetab)     ;
        st        r02,r0,r0                 ; zero r0 of parameter block
        stcr      r02,cr19                  ; save it off
        or.u      r08,r0,hi16(0x120003fe)   ; usr, big e, serial
        or        r08,r08,lo16(0x120003fe)  ; no sfus, no misaligned data, no ints
        stcr      r08,cr18                  ; save it off PSR
        or.u      r06,r0,hi16(tcode)        ; setup trace code (supplied by monitor)
        or        r06,r06,lo16(tcode)       ;
        stcr      r06,cr16                  ;
        or.u      r09,r0,hi16(ilcode)       ; pointer illegal area
        or        r09,r09,lo16(ilcode)      ;
        stcr      r09,cr06                  ; save it
                                            ; setup registers etc for trace code
        br        _emulate                  ; off we go.
;
;                                         Trace execution Area filled with illegal opcodes
;

        word      0xf400fc01
        word      0xf400fc01
        word      0xf400fc01
        word      0xf400fc01
        word      0xf400fc01
        word      0xf400fc01
        word      0xf400fc01
        word      0xf400fc01
        word      0xf400fc01
        word      0xf400fc01
```

```
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
ilcode:
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01
        word        0xf400fc01

;
;         Save area
;
;
        global      savetab
bss     savetab,1024
```

```
;*****************************************************************
;
;               PROGRAM NAME:     emulate.c
;                    VERSION:     01
;               LAST MODIFIED:    6 March 1991
;                    LANGUAGE:    MC88110 assembler and cpp for macros
;                                  and defs
;                      AUTHOR:    jwr (EKB)
;                     COMPANY:    Motorola LTD Copyright (C)
;               TARGET SYSTEM:    MC88110
;                 ENVIRONMENT:    Low level debugger (standalone)
;                 DESCRIPTION:    This routine performs the single stepping of
;                                  user code.
;                   The flow of the routine is as follows:-
;                   Instruction fetch and decode.
;                   Monitor functions like disassemble code
;                                          display registers
;                                          single step interface
;                                          software break points
;                   If normal then placed in the illegal error and executed.
;                   If unconditional branch/jump and not delayed then change
;                   program counter. (If delayed execute delayed before.)
;                        If conditional then check for the condition then branch/
;                           nobranch remembering to check the delayed slot.
;
;       DEPENDENCIES:                      Illegal opcode and transfer from USER to
;                   supervisor.
;                   Also requires Monitor work areas (savetab and tcode).
;                   The unimplemented opcode vector must point to _emulate.
;                   All other exceptions should vector to _prmexcp.
;
;
;
;         HISTORY:
;
;*****************************************************************

#include          "storage.h"
        text
        global    _emulate

_emulate:
;
;                 First save all of the GRF so that the routine can access the registers
;                 using an index.
;
        xcr       r02,r02,cr19          ; get Data storage pointer Note could have been a
                                        ; constant load rather than using a control register
        st        r02,r03,ofreg03       ; save r03
        ldcr      r03,cr19
        st        r02,r03,ofreg01       ; save r01
        st        r02,r03,ofreg02       ; save r02    Used as TEMP1
        st        r02,r04,ofreg04       ; save r04    Used as TEMP2
        st        r02,r05,ofreg05       ; save r05    Used as TEMP3
        st        r02,r06,ofreg06       ; save r06
        st        r02,r07,ofreg07       ; save r07
        st        r02,r08,ofreg08       ; save r08
        st        r02,r09,ofreg09       ; save r09
        st        r02,r10,ofreg10       ; save r10    Used as TEMP4
        st        r02,r10,ofreg11       ; save r11
        st        r02,r10,ofreg12       ; save r12
        st        r02,r10,ofreg13       ; save r13
```

```
        st      r02,r10,ofreg14         ; save r14
        st      r02,r10,ofreg15         ; save r15
        st      r02,r10,ofreg16         ; save r16
        st      r02,r10,ofreg17         ; save r17
        st      r02,r10,ofreg18         ; save r18
        st      r02,r10,ofreg19         ; save r19
        st      r02,r10,ofreg20         ; save r20
        st      r02,r10,ofreg21         ; save r21
        st      r02,r10,ofreg22         ; save r22
        st      r02,r10,ofreg23         ; save r23
        st      r02,r10,ofreg24         ; save r24
        st      r02,r10,ofreg25         ; save r25
        st      r02,r10,ofreg26         ; save r26
        st      r02,r10,ofreg27         ; save r27
        st      r02,r10,ofreg28         ; save r28
        st      r02,r10,ofreg29         ; save r29
        st      r02,r10,ofreg30         ; save r30
        st      r02,r10,ofreg31         ; save r31

;
;               Get pointer to illegal area
;
        xcr     r09,r09,cr06            ; this could have been a constant load rather than
                                        ; using a control register
;
;               Now get the Trace program machine state
;
        xcr     r08,r08,cr18            ; get PSR
        xcr     r07,r07,cr17            ; get NIP
        xcr     r06,r06,cr16            ; get XIP
;
;               Check the additional bits in the PSR
;               to see if anything needs to be done
;
        bb1     bn,r08,delslot          ; Check if in a delayed slot
_get01:
        bb1     bdis,r08,disreg         ; Display registers
_get10:
        bb1     bdasm,r08,disam         ; Disassemble instruction
_get20:
        bb1     bsing,r08,sstep         ; Single step
_get30:
        bb1     bsbrk,r08,swbrk         ; Software Breaks on
_get40:
;
;               Now check if the instruction could
;               cause a change of program flow
;
        ld      r10,r06,0               ; get instruction
        extu    r03,r10,5<27>           ; isolate the opcode
        cmp     r04,r03,0x18            ; is it br
        bb1     eq,r04,brinst           ;
        cmp     r04,r03,0x19            ; is it bsr
        bb1     eq,r04,bsrinst
        cmp     r04,r03,0x1a            ; is it bb0
        bb1     eq,r04,bb0inst
        cmp     r04,r03,0x1b            ; is it bb1
        bb1     eq,r04,bb1inst
        cmp     r04,r03,0x1d            ; is it bcnd
        bb1     eq,r04,bcndinst
        or.u    r04,r0,hi16(0xf400c000) ; build mask for jmp/jsr
        or      r04,r04,lo16(0xf400c000)
```

```
        and     r03,r10,0xc000
        cmp     r04,r03,r04             ; check it
        bb1     eq,r04,jinst
        or.u    r05,r0,hi16(0xf400fc00) ; build mask for rte
        or      r05,r05,lo16(0xf400fc00)
        cmp     r04,r10,r05
        bb1     eq,r04,rteinst
;
;               At this point just an ordinary instruction
;               rebuild the general register file and
;               place the instruction in the illegal area
;               and set the EPSR,EXIP and ENIP.
;
        addu    r06,r06,04              ; XIP + 4 -> XIP point to next instruction

_goinst:

        st      r09,r10,0               ; store the instruction in the illegal area
        stcr    r09,cr4                 ; EXIP setup.
        addu    r09,r09,4               ; Point to next instruction.
        and.u   r03,r08,0xff00          ; maskout the monitor status bits
        stcr    r03,cr2                 ; EPSR setup
        stcr    r00,cr5                 ; ENIP setup (never let branchs occur)
;
;               Load back all the registers that were
;               saved while in this section
;
        ld      r03,r02,ofreg03         ; restore r03
        ld      r04,r02,ofreg04         ; restore r04
        ld      r05,r02,ofreg05         ; restore r05
        ld      r10,r02,ofreg10         ; restore r10
;
;               Save the Trace program machine state
;
        xcr     r06,r06,cr16            ; save XIP
        xcr     r07,r07,cr17            ; save NIP
        xcr     r08,r08,cr18            ; save PSR
        xcr     r02,r02,cr19            ; save Data storage pointer
        xcr     r09,r09,cr06            ; save pointer to illegal area
        rte                             ; Off we go
;
;       Change to branch address
;
delslot:
        and     r06,r07,r07             ; NIP -> XIP
        clr     r08,r08,1<bnx>          ; Clear delay slot flag.
        bsr     _get01
;
;       br instruction
;
brinst:
        ext     r03,r10,26<0>           ; get D26
        mak     r03,r03,<2>             ; D26 << 2
        add     r04,r06,r03             ; XIP + D26 << 2 -> TEMP (branch address)
        bb0     26,r10,brlb01           ; Check N bit set
        addu    r06,r06,4               ; XIP + 4 -> XIP
        and     r07,r04,r04             ; setup NIP
        set     r08,r08,1<bnx>          ; Set delayed slot flag
        br      _get01                  ; go and do delay slot.
brlb01:
        and     r06,r04,r04             ; TEMP -> XIP
        br      _get01                  ; go do next instruction
```

```
;
;       bsr instruction
;
bsrinst:
        ext         r03,r10,26<0>          ; get D26
        mak         r03,r03,<2>            ; D26 << 2
        add         r04,r06,r03            ; XIP + D26 << 2 -> TEMP  (branch address)
        bb0         26,r10,bsrlb01         ; Check N bit set
        addu        r06,r06,4              ; XIP + 4 -> XIP
        addu        r01,r06,4              ; XIP + 4 -> r01    (retrun address)
        st          r01,r02,ofreg01        ; Save it in table
        and         r07,r04,r04            ; TEMP -> NIP
        set         r08,r08,1<bnx>         ; set delayed slot flag
        br          _get01                 ; go and do delay slot.
bsrlb01:
        addu        r01,r06,4              ; XIP + 4 -> r01
        st          r02,r01,ofreg01        ; Save it in table
        and         r07,r0,r0              ; Clear NIP
        and         r06,r04,r04            ; TEMP  -> XIP
        br          _get01                 ; go do next instruction

;
;       jmp or jsr instruction
;
jinst:
        extu        r03,r10,5<0>           ; get S2
        ld          r04,r02[r03]           ; get contents of S2
        bb1         11,r10,jsr01           ; jsr instruction
        bb1         10,r10,jmp01           ; Check N bit set
        and         r06,r04,r04            ; S2 -> XIP   at this
                                           ; point the signed delta
                                           ; should be added to compensate
                                           ; for the difference in location
                                           ; between run time location and the
                                           ; location of the parameter block.
        clr         r08,r08,1<bnx>         ; clear delay slot flag
        br          _get01                 ; return to main flow
;
jmp01:
        and         r07,r04,r04            ; S2 -> NIP at this
                                           ; point the signed delta
                                           ; should be added to compensate
                                           ; for the difference in location
                                           ; between run time location and the
                                           ; loaction of the parameter block.
        addu        r06,r06,4              ; XIP + 4 -> XIP
        set         r08,r08,1<bnx>         ; Set that we are in delayed slot
        br          _get01                 ; return to main flow
;
jsr01:
        addu        r06,r06,4              ; XIP + 4 -> XIP
        bb1         10,r10,jsr02           ; Check N bit set
        and         r01,r06,r06            ; XIP + 4 -> r01
        st          r01,r02,ofreg01        ; Save in table
        and         r06,r04,r04            ; S2 -> XIP
        clr         r08,r08,1<bnx>         ; Set that we are in delayed slot
        br          _get01                 ; return to main flow
;
jsr02:
        addu        r01,r06,4              ; XIP + 8 -> R1
        st          r01,r02,ofreg01        ; Save in table
```

```
        and     r07,r04,r04             ; S2 -> NIP
        set     r08,r08,1<bnx>          ; Set that we are in delayed slot
        br      _get01                  ;
;
;       bcnd Instruction
;
bcndinst:
        extu    r03,r10,5<16>           ; S1
        ld      r05,r02[r03]            ; Contents of S1
        extu    r03,r10,5<21>           ; M5
        extu    r04,r05,1<31>           ; sign bit
        mak     r04,r04,<1>             ; << 1
        clr     r05,r05,1<31>           ; clr sign bit
        cmp     r05,r05,r0              ;
        bb1     ne,r05,nonzero          ;
        set     r04,r04,1<0>            ; set bit 0
nonzero:
        set     r04,r04,1<5>            ; width 1
        extu    r03,r03,r04             ; get bit from M5
        bb1     0,r03,bcnd01            ; Are we branching
        addu    r06,r06,4               ; XIP + 4 -> XIP
        clr     r08,r08,1<bnx>          ; no delayed slot as we are not branching
        br      _get01                  ; return to main flow
;
bcnd01:
        ext     r03,r10,16<0>           ;  D16
        mak     r03,r03,<2>             ; D16 << 2
        add     r04,r06,r03             ; XIP + D16 << 2
        bb0     26,r10,bcnd02           ; Check N bit
        addu    r06,r06,4               ; XIP + 4 -> XIP
        and     r07,r04,r04             ; XIP + D16 << 2 -> NIP
        set     r08,r08,1<bnx>          ; Set that we are in delayed slot
        br      _get01                  ; return to main flow
;
bcnd02:
        and     r06,r04,r04             ; XIP + D16 << 2 -> XIP
        clr     r08,r08,1<bnx>          ; not in delayed slot
        br      _get01                  ; return to main flow
;
;               bb0 Instruction
;
bb0inst:
        extu    r03,r10,5<16>           ; S1
        ld      r05,r02[r03]            ; get contents of S1
        extu    r03,r10,5<21>           ; B5
        set     r03,r03,1<5>            ; length = 1
        extu    r04,r05,r03             ; get the bit to test
        bb0     0,r05,bb0_02            ; are we branching
        addu    r06,r06,4               ; No XIP + 4 -> XIP
        clr     r08,r08,1<bnx>          ; not in delayed slot
        br      _get01                  ; return to main flow
;
bb0_02:
        ext     r04,r10,16<0>           ; D16
        mak     r03,r03,<2>             ; D16 << 2
        add     r04,r06,r03             ; XIP + D16 << 2
        bb0     26,r10,bb0_03           ; check N bit
        clr     r08,r08,1<bnx>          ; Not in delayed slot
        and     r06,r06,r04             ; XIP + D16 << 2 -> XIP
        br      _get01                  ; return to main flow
;
bb0_03:
```

```
        addu        r06,r06,4               ; XIP + 4 -> XIP
        and         r08,r04,r04             ; XIP + D16 <<2 -> NIP
        set         r08,r08,1<bnx>          ; Flag that we are in delayed slot
        br          _get01                  ; return to main flow.
;
;                   bb1 Instruction
;
bb1inst:
        extu        r03,r10,5<16>           ; S1
        ld          r05,r02[r03]            ; get contents of S1
        extu        r03,r10,5<21>           ; B5
        set         r03,r03,1<5>            ; length = 1
        extu        r04,r05,r03             ; get the bit to test
        bb1         0,r05,bb0_02            ; are we branching
        addu        r06,r06,4               ; No XIP + 4 -> XIP
        clr         r08,r08,1<bnx>          ; not in delayed slot
        br          _get01                  ; return to main flow
;
bb1_02:
        ext         r04,r10,16<0>           ; D16
        mak         r03,r03,<2>             ; D16 << 2
        add         r04,r06,r03             ; XIP + D16 << 2
        bb0         26,r10,bb0_03           ; check N bit
        clr         r08,r08,1<bnx>          ; Not in delayed slot
        and         r06,r06,r04             ; XIP + D16 << 2 -> XIP
        br          _get01                  ; return to main flow
;
bb1_03:
        addu        r06,r06,4               ; XIP + 4 -> XIP
        and         r08,r04,r04             ; XIP + D16 <<2 -> NIP
        set         r08,r08,1<bnx>          ; Flag that we are in delayed slot
        br          _get01                  ; return to main flow.

;
;                   Addition routines for debug monitor
;
;
;       ROUTINE: disreg                     (Displays the General registers)
;
;       DEPENDENCIES: The registers are contained in the savetab
;                     in the monitor work space.
;
disreg:
                                                    ; Save all working registers
                                                    ; display the registers on current
stream
                                                    ; restore the working registers
        br          _get10                  ; return to main flow

;
;       ROUTINE: diasm                      (Disassembles instruction)
;
;       DEPENDENCIES: r06 must point to the instruction to be disassembled.
;
disam:
        ; Save all working registers
        ; get and disassemble instruction
        ; restore the working registers
        br          _get20                  ; return to main flow
;
;
```

```
;       ROUTINE: Single step                    (Single Step instructions)
;
;       DEPENDENCIES: Needs keyboard input and storage area for
;                           iteration number ITNUM (32 bits)
;
sstep:
                                                ; Save all working registers
                                                ; check if we are just do a number of
                                                ; iterations or are we single stepping?
                                                ; if single stepping wait for key input
                                                ; if iterations check not at end of iteration
                                                ;     if at end of iterations trap to monitor
                                                ; if going back increment counter
                                                ; restore registers
        br          _get30                  ; return to main flow

;
;
;       ROUTINE: Software Break Point
;
;       DEPENDENCIES: Break table in ram and hit count table in ram.
;
;
swbrk:
                                                ; Save all working registers
                                                ; Load break table point
                                                ; Scan table
                                                ; if match
                                                ;     then increment hit count for this break
point
                                                ;          if count limit reached
                                                ;               then trap to monitor
                                                ; restore registers
        br          _get40                  ; return to main flow

;
;       rte instruction
;
;
rteinst:
        ld          r08,r02,ofepsr          ; EPSR -> PSR
        ld          r07,r02,ofesnip         ; ENIP -> NIP
        ld          r06,r02,ofexip          ; EXIP -> XIP
        br          _get01                  ; return to main flow.

;
;
;       All traps point here
;
        global      _prmexcp
```

```
_prmexcp:
                                                    ; save off exception registers with correct
                                                    ; values from the trace buffers.
                                                    ; setup trace program EPSR, ENIP, EXIP for the
handler
                                                    ; if the exception is to be handled by the
SSCP
;       then        do what is required and return to the
;                   emulating the TRACE program.
;       else        if emulating the handler
;                   then                    setup PSR, NIP XIP to point to
;                                           the handler and return to main flow.
```

```
;/*****************************************************************************/
;/*                                                                           */
;/*            PROGRAM NAME:    vector.s                                      */
;/*                 VERSION:    01                                            */
;/*           LAST MODIFIED:    1 March 1991                                  */
;/*                LANGUAGE:    MC88110 assembler                             */
;/*                  AUTHOR:    jwr (EKB)                                     */
;/*                 COMPANY:    Motorola LTD Copyright (C)                    */
;/*             DESCRIPTION:    Vector Table for Single Stepping Control      */
;/*                             Program.                                      */
;/*                                                                           */
;/*            DEPENDENCIES:                                                  */
;/*                                                                           */
;/*                 HISTORY:                                                  */
;/*                                                                           */
;/*****************************************************************************/

        file "vector.s"
        text
vecbase:
        or          r00,r00,r00             ;
        br          startup                 ; 000: reset
        bsr         _prmexcp
        or          r00,r00,r00             ; 001: interrupt
        bsr         _prmexcp
        or          r00,r00,r00             ; 002: code access fault
        bsr         _prmexcp
        or          r00,r00,r00             ; 003: data access fault
        bsr         _prmexcp
        or          r00,r00,r00             ; 004: misaligned access
        bsr         _prmexcp
        or          r00,r00,r00             ; 005: unimplemented opcode
        bsr         _emulate                ; single stepping code entry point.
        or          r00,r00,r00             ; 006: privilege violation
        bsr         _prmexcp
        or          r00,r00,r00             ; 007: array bounds violation
        bsr         _prmexcp
        or          r00,r00,r00             ; 008: integer divide error
        bsr         _prmexcp
        or          r00,r00,r00             ; 009: integer overflow
        bsr         _prmexcp
        or          r00,r00,r00             ; 010: error, exception when shadowing frozen
        bsr         _prmexcp
        or          r00,r00,r00             ; 011: reserved
        bsr         _prmexcp
;
;               The above two instructions are repeated for the
;               remaining vectors.
;
```