FBUG MONITOR

Rev 1.1
Release Date
September 28, 1989

TABLE OF CONTENTS

CHAPTER 1
GENERAL INFORMATION

1.1 DESCRIPTION OF FBUG(68xxx)

The FBUG monitor is a stand alone software package designed to assist in evaluating/debugging systems which use the 68xxx Microprocessor. It has the capability to load and execute user code and includes an assembler/ disassembler designed for quick program patchwork. The monitor operates in a user-interactive command driven mode signified by the MOTOROLA> prompt. The command line entered after this prompt determines which operation is performed.

CHAPTER 2
THE FBUG COMMAND SET

2.1 INTRODUCTION

This chapter explains the FBUG monitor commands and their associated syntax. Table 2.1 summarizes the available commands and shows the section where the command is explained in greater detail.

TABLE 2.1 FBUG MONITOR COMMANDS

Command Mnemonic	Name	Section
as	Assembler/Disassembler	2.2
bf	Block of Memory Fill	2.3
bm	Block of Memory Move	2.4
br	Breakpoint	2.5
bs	Block of Memory Search	2.6
ct	Confidence Test	2.7
dc	Data Conversion	2.8
go	Go	2.9
?/he/help	Help	2.10
lo	Load S-Records	2.11
md	Memory Display	2.12
mm	Memory Modify	2.13
rd	Register Display	2.14
rm	Register Modify	2.15
sd	Symbol Define	2.16
tm	Transparent Mode	2.17
tr	Trace	2.18

The command line is composed of:

<COMMAND IDENTIFIER>: specifies which command (ex. br)
<SP>: at least one space
OPTION LIST: an option delimiter(-) with options if non-default options are allowed and are being used. (ex. [<-r>])
<SP>: at least one space
ARGUMENTS: any required arguments specified by the command separated by commas/spaces as shown in the command description (ex. <ADDR,ADDR>)

where "<>" enclose symbols that are required on the command line and "[<>]" enclose symbols that are optional on the command line. Note, in the above examples the -r option was an example of an optional symbol and that the ADDR fields are requirements on the command line. The options available with a given command are fully explained in the section that describes that command. The monitor is not case sensitive to input from the terminal. All input from the terminal is converted to lower case before being used internally (except text following a text delimiter; See TEXT below). The arguments of a given command are described using the following symbols:

<EXP>: An expression can be any numerical expression which may be evaluated using only the arithmetic + and - operators.

Ex. 1000
Ex. 1+3
Ex. 13+/reset NOTE: where /reset has been predefined (see section 2.16)

Note: Numbers may be preceded with a base designator if the default (hexadecimal) is not desired. These designators are shown below in Table 2.2:

TABLE 2.2 BASE DESIGNATORS

Base	Designator
Hexadecimal	$
Decimal	&
Octal	@
Binary	%

<ADDR>: Address field is any valid expression. Note: This address field should not be confused with the source and destination addresses required using the Assembler/Disassembler.

<COUNT>: Count field is any valid expression preceded by a COUNTDEL (count delimiter,i.e.. ":")

Ex. :100

<RANGE>: A range of memory locations denoted by either ADDR,ADDR or ADDR:COUNT.

Ex. 0,100
Ex. 0:50

<TEXT>: An ASCII string of up to 255 characters preceded by a TEXTDEL (text delimiter i.e.. ";")

Ex. ;sample text

<SIZE>: Can be either:
byte (8 bit) ====> -b
word (16 bit) ====> -w
long (32 bit) ====> -l

Note: ====> stands for "is represented by" or "returns"

<DATA>: Data can be any valid expression.

<MASK>: A mask may be any expression. After evaluating the expression 0's represent don't cares. A mask is sometimes used to qualify <DATA>. See section 2.6 for an example of usage.

2.2 ASSEMBLER/DISASSEMBLER
as <ADDR>

The assembler/disassembler is invoked at the address given and disassembles the object code at that location. Use of the Assembler/Disassembler is fully described in chapter 3.

2.3 BLOCK OF MEMORY FILL
bf [<SIZE>] <RANGE> <DATA>

The block fill command fills the specified range of memory with the data listed. If the size option is not specified the default size used is word. If a multiple of the <SIZE> of <DATA> does not fit evenly in the <RANGE> the command leaves the last partial word or long word unchanged.

Examples of use:

```
bf 100,110      &10
bf 100:8        &10
bf -w 100:8     a
bf -l 100,110   a000a
```

Note: All of these examples perform the same memory fill.
(i.e.. $00000100: $000a $000a $000a $000a $000a $000a
$0000010C: $000a $000a $0000 $0000 $0000 $0000)

2.4 BLOCK OF MEMORY MOVE

bm [<SIZE>] <RANGE> <ADDR>

The block move command allows the user to copy segments of memory to different locations in memory. The execution of this command does not destroy the original version unless the location moved to <ADDR> is within the range <RANGE> of the code being copied. The size option is only available when range is described as <ADDR>:<COUNT>. If range is being described with the <ADDR>,<ADDR> mode the size defaults to byte. The size field represents the size transfer that is used to accomplish the memory move.

Examples of use:

```
bm 1000,2000    10000
bm 1000:800     10000
bm -l 1000:400  10000        Note: This variation executes the fastest
```

Note: All of these examples perform the same memory move.

2.5 BREAKPOINT

```
br
br <ADDR>
br <ADDR> <:COUNT>
br -r [<ADDR>]
br -r
```

The breakpoint command allows the user to list, insert or delete breakpoints in the target code. This allows the user to stop executing a program and return to the monitor environment when the specified <ADDR> is prefetched. The different uses of this command are summarized below:

br	list all known breakpoints
br <ADDR>	insert a breakpoint at this address
br <ADDR> <:COUNT>	insert a breakpoint at this address, however, return to the monitor environment only after encountering the breakpoint <COUNT> number of times.
br -r [<ADDR>]	remove the breakpoint at this address
br -r	remove all breakpoints

2.6 BLOCK SEARCH

```
bs [<SIZE>] <RANGE> <DATA>
bs [<SIZE>] <RANGE> <DATA> <MASK>
```

The block search command allows the user to find a specific pattern within memory. The search area may extend beyond the <RANGE> specified if a pattern is started within <RANGE>. There are two primary types of searches:

bs [<SIZE>] <RANGE> <DATA>	searches the range for an exact match of <DATA>.
bs [<SIZE>] <RANGE> <DATA> <MASK>	searches the range for any

pattern that matches <DATA>
where there is a "1" in the
binary representation of
the mask.

Ex. With memory at location $100 as shown below, executing
"bs 100,118 $1234 $ffbf" ====>

Starting address: $00000100
Ending address: $00000117
Found at: $00000110:$1234
Found at: $00000114:$1274

Memory for the example above:

$00000100: $0000 $0000 $0000 $0000 $0000 $0000
$0000010C: $0000 $0000 $1234 $0000 $1274 $0000

2.7 CONFIDENCE TEST

ct

The confidence test command allows the user to perform a confidence test of the system being evaluated. This command is designed to write a -1 into the result variable (int confres in vardef.h) of the confidence test and then pass the address of this variable in register a0 to a called subroutine. This routine writes a 0 or 1 into confres depending on whether the test passes or fails respectively. If the routine does not place a valid result in this variable then the monitor assumes that the confidence test was not performed and displays "ERROR 14:Unable to perform Confidence Test". The confidence test to be performed is system dependent and has to developed for the system on which the monitor is running.

2.8 DATA CONVERSION

dc <EXP>

The data conversion command allows the user to evaluate an input expression and determine its hexadecimal and decimal equivalent.

Examples of use:

NOTE: The following symbols have been defined earlier in order
fpr them to be used in the examples below:
EX 1. uses /start= 0
Ex 2. uses /start= - $18
Ex 3. uses /finish= 10000 and /start=$10000
(see section 2.17)

Ex. 1 dc $17+/start ====> $17 = &23
Ex. 2 dc $17+/start ====> UNSIGNED : $FFFFFFFF = &4294967295
SIGNED : -$1 = -&1
Ex. 3 dc $/finish-/start ====> $10000 = 65536

2.9 GO

go [<ADDR>]

The go command allows the user to execute target programs. If an address is not specified on the command line then the current PC value is used. This value is either 1.)the initialized PC value if no target code has been run, 2.)the last value of the PC used in executing target code or 3.)the value placed into the PC register by a RM command (Register Modify see section 2.16). If an address is included on the command line then the PC is modified to be the specified address. and execution begins at this address. In both cases, the register state that the microprocessor is initialized to, before executing the target code at this location, can be viewed by executing a rd command (See section 2.15).

2.10 HELP

? [<symbol>]
he [<symbol>]
help [<symbol>]

The help command allows the user to view a list of allowable commands and the syntax associated with them. Symbols used to describe the command usage can be looked up also.

Examples of use:

?,he or help ====> return a complete listing of all commands with usage
? as ====> AS <addr>
help addr ====> <number>
he number ====> <hex> || <dec> || <oct> || <bin> || <symbol>

Note: <number> may also be an expression

2.11 LOAD S-RECORD

lo [<OFFSET>] ;<TEXT>

The load command allows the user to download S-Records from the host system. If an offset is present on the command line then the target address is the offset added to the address determined by the S-Record. This command sends the <TEXT> beyond the ";" to the HOST. It then expects the HOST to begin sending S-Records to the terminal.

Examples of use:

lo ;cat fbug.mx

Note: The "cat" command is a UNIX command that concatenates and then prints the specified files using standard output. This effectively sends the contents of the file to the terminal. The monitor then loads the contents of the S-Records in the file to the addresses determined by the S-Records.

lo a0000 ;cat fbug.mx

Note: This command downloads the same S-Record file used in the first example except that it is down loaded into memory at the address determined by the S-Record + $a0000 (i.e.. the offset is added in).

2.12 MEMORY DISPLAY

md [<SIZE>] <addr>
md [<SIZE>] <RANGE>
md -di <addr>

The memory display command allows the user to view memory. The size used to display the memory is determined by the size option. If no option is used the default is word. If the range exceeds the screen capacity, output to the screen is suspended until any key is pressed.

Examples of use:

md -l 100,110
md -l 100:4
md 100:8
md -di 100 Note: This command begins to disassemble the memory at this location.

2.13 MEMORY MODIFY

mm [<SIZE>] <ADDR>
mm <CONTROL>

The memory modify command allows the user to view and modify memory. The size used to display the memory is determined by the size option. The size default is word. Memory is displayed beginning at the address specified followed by a '?' prompt. The user may type in an <exp> to replace that memory value or hit return to view the next memory value. To exit the command, type ". <cr>" (period <carriage return>). Other available <CONTROL> characters are summarized below in Table 2.3:

TABLE 2.3 CONTROL CHARACTERS

Control Character	Designator
- <EXP>	backup <EXP> memory locations
+ <EXP>	advance " " "

Examples of use:

mm -l 100 ====> $00000100 $00000000 ?
mm 100 ====> $00000100 $0000 ?
(i.e.. uses the default "word" size)

2.14 REGISTER DISPLAY

rd
rd -f Note: Only valid if Coprocessor support has been specified
see below.

The register display command allows the user to view the contents of the registers of the mpu/fpu. Versions released before August 30,1989 only supported the 68020 programmers model and did not support the coprocessor (68881/882) programmers model. Versions released after August 30,1989 support the programmers model that the user specifies before compiling the monitor. This is done by using the DEVICE,EMULATOR and COPROCESSOR "define" statements in the "targetsys.h" header file. These defines are discussed further in the "Target System Configuration" section. The values shown for the registers are either 1.) the initialized register state if no target code has been run, 2.) the last values in the registers while executing target code or 3.) the values specifically placed into the registers by a rm command.

2.15 REGISTER MODIFY

rm [<REGISTER> [<New Value>]]

The RM command allows the user to the modify the contents of the registers of the mpu.

Examples of use:

To change the PC value:

rm pc 3000 ====> changes the PC value to 3000

or

rm ====> Which register?
pc ====> PC=00004000NEW VALUE?
3000 ====> changes the PC value to 3000

or

rm pc ====> PC=00004000NEW VALUE?
3000 ====> changes the PC value to 3000

2.16 SYMBOL DEFINE

sd [<SYMBOL> <EXP>]
sd -r <SYMBOL>

The symbol define command allows the user to define symbols. These symbols can then be used within expressions. Using a symbol in an expression results in the symbol being substituted with the expression that was used to define it. Once defined, the symbol is available until the monitor is reset. If a symbol is defined multiple times the monitor uses the first definition.

Examples of use:

```
sd                 ====>     lists which symbols are already
                             known
sd /reset 10000    ====>     defines /reset to be $10000
                             whenever it is used in an expression.
sd /start -$18     ====>     defines /start to be -$18
                             whenever it is used in an expression.
sd -r /start       ====>     removes the first definition of /start
                             from the list
```

NOTE: Symbols that have been defined using the sd command can be used in any expressions. An example of this is to use a symbol defined to enter source code while in the assembler (i.e.. bsr /startsub after defining /startsub).

2.17 TRANSPARENT MODE

tm

The transparent mode command places the user into transparent mode by establishing a software connection between the HOST and TERMINAL. Transparent mode preempts normal communication between the TERMINAL and the debugger. While in this mode all keyboard input is relayed directly to the HOST. HOST responses, in turn, are returned to the screen. Typing a CTRL A returns the user to the monitor environment.

2.18 TRACE

tr [<ADDR>][<COUNT>]

The trace command allows the user to trace though target code and observe the registers after executing the command line. If count is specified then the microprocessor executes <COUNT> number of instructions before returning to the monitor environment. Trace begins from the <ADDR> listed on the command line or from the current PC if an <ADDR> is not included. The trace instruction can be continued by hitting a carriage return. To exit, a "." must be entered.

Examples of use:

```
tr          ====>     traces 1 instruction from the current PC
tr :10      ====>     executes 10 instructions past the current
                      PC then returns to the monitor
                      environment
tr 1000     ====>      traces 1 instruction starting at $1000
```

CHAPTER 3
USING THE ONE-LINE ASSEMBLER/DISASSEMBLER

3.1 INTRODUCTION

Included in the FBUG monitor is an assembler/disassembler command which can be executed as detailed in the previous chapter. This assembler/ disassembler allows the user to modify target code. Each source line that is typed in by the user is entered into memory at the displayed address. This line is then disassembled so that the user can verify the actual code entered into memory. If no change is desired a <CR> moves the user to the next opcode in memory.

NOTE: The instruction set that is available to the user is determined when the monitor is compiled. The choice of the values of DEVICE, EMULATOR and COPROCESSOR determine which instruction set is available. See the "Target System Configuration" section below.

CAUTION: This assembler/disassembler does not insert code into the source program; it merely overwrites memory at that location. As a result, a program patch that requires code insertion can be accomplished by first Block Moving code to free up an insertion area and then inserting into that area.

3.2 ENTERING AND MODIFYING SOURCE PROGRAM

In order to enter and modify source code, the as command should be executed as detailed in section 2.2 (i.e.. as <ADDR>). This places the user into the Assembler/Disassembler routine. Table 3.1 summarizes the commands that can be executed within this routine:

TABLE 3.1 ASSEMBLER/DISASSEMBLER SUB COMMANDS

Command	Designator
BACKUP <EXP>	- <EXP>
ADVANCE <EXP>	+ <EXP>
FINISH	.
HELP	?
STEP PAST	carriage return
DEFINE CONSTANT	DC #<EXP>

Note: Executing a '?' while in the assembler/disassembler returns the DEVICE that the assembler/disassembler is supporting.

3.3 ENTERING A SOURCE LINE

After executing an as <ADDR> command, the assembler/disassembler returns with the disassembly of the code found at that location. At this time the user may execute an assembler command shown in section 3.2 or type in the source line that is to replace the displayed source code. While entering source the standard MOTOROLA effective addressing modes are used. These modes are summarized below in Table 3.2:

TABLE 3.2 ASSEMBLER/DISASSEMBLER EFFECTIVE ADDRESSING MODES

Effective Addressing Mode	Syntax
Register Direct	Dn
Address register direct	An
Address register indirect	(An)
Address register indirect with Postincrement	(An)+
Address register indirect with Predecrement	-(An)
Address register indirect with Displacement	(d16,An)
Address register indirect with Index (d8)	(d8,An,Xn.SIZE*SCALE)
Address register indirect with Index (base displacement)	(bd,An,Xn.SIZE*SCALE)
Memory indirect Post-indexed	([bd,An],Xn.SIZE*SCALE,od)
Memory indirect Pre-indexed	([bd,An,Xn.SIZE*SCALE],od)
PC indirect with displacement	(d16,PC)
PC indirect with index (d8)	(d8,PC,Xn.SIZE*SCALE)
PC indirect with index (bd)	(bd,PC,Xn.SIZE*SCALE)
PC memory indirect Post-indexed	([bd,PC],Xn.SIZE*SCALE,od)
PC memory indirect Pre-indexed	([bd,PC,Xn.SIZE*SCALE],od)
Absolute Short Address	(xxx).W
Absolute Long Address	(xxx).L
Absolute Address	xxx optimizes (bwl)
Immediate Data	#xxx

While using the POST or PRE indexed modes, fields may be skipped by using a comma. An example is shown below:

Ex. andi #12,([,],,) ====> andi.b #$12,([$0,ZA0],ZD0.W*1,$0)

Other examples of source lines are shown below:

Ex. ori.l #12,(a1) =====> ori.l #$12,(a1)

Ex. addq #1,(a1) =====> addq.b #$1,(a1)

There are only limited error screening abilities included within the monitor. Examples of this are shown below:

Ex. jmp (123).w =====> jmp ($123).W

Note: When executed results in a bus error.

14 4/18/21

14 4/18/21

Ex. bsr (123) =====>

ERROR 10: illegal change of flow ===> bsr (123)

Note: The bsr instruction does check for illegal changes of flow.

NOTE: Flow may not be changed to an odd address.

Upper digits of data are NOT truncated when a mismatch between size and immediate data is found if the byte or word size option was specifically entered. If the long size option is specified and data exceeds this range then upper digits ARE truncated.

Ex. addi.b #12345678,(a1) ====>

ERROR 11:immediate data/size option error ===> addi.b #12345678,(a1)

Ex. addi.l #123456789,(a1) ====> addi.l #$23456789,(a1)
Ex. addi #123456789,(a1) ====> addi.l #$23456789,(a1)
(defaults to the long option)
truncated----------^

Upper digits of data are truncated on commands that have a limited field in the opword to store the immediate data. Examples of this are shown below.

Ex. addq #10,(d0) =====> addq.b #$0,(d0)

Ex. trap #10 =====> trap #$0 input is hex default

Ex. trap #&10 =====> trap #$A

3.4 ENTERING CHANGE OF FLOW INSTRUCTIONS

Since the assembler/disassembler does not use labels, all instructions that use <label> as an effective addressing mode must have their displacement determined. If initially unknown, space for this displacement must be reserved and then the user needs to come back and enter the displacement. Once the displacement has been determined it may be entered as shown in the following example:

Ex. In this example the location of the target instruction of a branch is known to be $100000 and a BRA is needed at location 0. After executing "AS 0" and obtaining the disassembly found at 0 the user could type:

```
BRA    100000      or
BRA    (100000)    or
BRA.l  #ffffe
```

The absolute addressing mode can be used if the target address of a branch is known (as in the first 2 examples) or the displacement (last example) can be entered using the immediate data addressing mode.

CAUTION: In some instances unexpected results can occur while using change of flow statements. These instances are summarized below with examples.

Ex. 1 If the degenerate case of a branch statement in used (i.e.. attempting to use a short branch to branch to the following instruction) the assembler mistakenly assembles this .b option. However, since the displacement is zero this is a .w opcode and the disassembler correctly displays this fact to the user.

```
$00004000      nop  ? bra.b  ====> results in an INCORRECT assembly
```

Ex. 2 If the user attempts to force a particular size branch statement and the actual branch requires a greater displacement than was reserved then the assembler prints an error message: "ERR0R 16: OUT OF DISPLACEMENT RANGE" .

```
$00004000      nop  ?bra.w (100000)
```

One way to assure this does not occur is to not enter a size option. This allows the assembler to choose the correct size for the displacement.

3.5 ENTERING REGISTERS and REGISTER LISTS

The move multiple register instruction (movem) uses a register list as an effective address. This list may be entered in the following method:

Ex.

a0	====>	single address register
d3	====>	single data register
a0-a3	====>	series of registers
a0-a3/a7	====>	combination of previous examples
a0-a7/d0-d7	====>	all of the registers

If coprocessor support is specified then the floating point registers can be entered as shown below:

Ex.

fp0	====>	single floating point register
fp0-fp2	====>	series of registers
fp0-fp3/fp7	====>	combination of previous examples

Many of the commands require the entering of registers other than data or address registers. Tables 3.3-8 show listings of the registers that are used and the abbreviations accepted by the assembler:

TABLE 3.3 68000/68HC000/68008 REGISTERS

Name	Syntax
User Stack Pointer	USP
Program Counter	PC
Condition Code Register	CCR
Supervisor Stack Pointer	SSP
Status Register	SR

TABLE 3.4 68010 REGISTERS

Name	Syntax
User Stack Pointer	USP
Program Counter	PC
Condition Code Register	CCR
Supervisor Stack Pointer	SSP
Status Register	SR
*Vector Break Register	VBR
*Source Function Code Register	SFC
*Destination Function Code Register	DFC

Note: * Represents a change from the previous model

TABLE 3.5 68020 REGISTERS

Name	Syntax
User Stack Pointer	USP
Status Register	SR
Condition Code Register	CCR
Program Counter	PC
*Master Stack Pointer	MSP
*Interrupt Stack Pointer	ISP
Vector Break Register	VBR
Source Function Code Register	SFC
Destination Function Code Register	DFC
*Cache Control Register	CACR
*Cache Address Register	CAAR

TABLE 3.6 68030 REGISTERS

Name	Syntax
User Stack Pointer	USP
Status Register	SR
Condition Code Register	CCR
Program Counter	PC
Master Stack Pointer	MSP
Interrupt Stack Pointer	ISP
Vector Break Register	VBR
Source Function Code Register	SFC
Destination Function Code Register	DFC
Cache Control Register	CACR
Cache Address Register	CAAR
*CPU Root Pointer Register	CRP
*Supervisor Root Pointer Register	SRP
*Translation Control Register	TC
*Transparent Translation Register 0	TT0
*Transparent Translation Register 1	TT1
*MMU Status Register	PSR

TABLE 3.7 68040 REGISTERS

Name	Syntax
User Stack Pointer	USP
Status Register	SR
Condition Code Register	CCR
Program Counter	PC
Master Stack Pointer	MSP
Interrupt Stack Pointer	ISP
Vector Break Register	VBR
Source Function Code Register	SFC
Destination Function Code Register	DFC
Cache Control Register	CACR
Cache Address Register	CAAR
User Root Pointer Register	URP
Supervisor Root Pointer Register	SRP
Translation Control Register	TC
*Data Transparent Translation Register 0	DTT0
*Data Transparent Translation Register 1	DTT1
*Instruction Transparent Trans. Reg. 0	ITT0
*Instruction Transparent Trans. Reg. 1	ITT1
MMU Status Register	PSR

Note: Also the floating point registers below

TABLE 3.8 FLOATING POINT REGISTERS

Name	Syntax
Floating Point Control Register	FPCR
Floating Point Status Register	FPSR
Floating Point Inst. Address Register	FPIAR
Floating Point Data Register	FP0-FP7

NOTE: PMMU (68851) support is not available under release 1.0.

3.6 ENTERING/EVALUATING FLOATING POINT IMMEDIATE DATA

Floating point immediate data must be entered using a decimal point with at least one (1) digit in front of the decimal place (even if it is a '0'). Ex. 0.0012. Since the C compiler used was not based on the draft proposed version of ANSI C the software is incapable of performing the 'assembling' of extended immediate data to extended precision. The monitor makes the correct conversion up to double precision and places this result in an extended format. If the compiler that is being used does conform to allowing a 'long double' type then changing the type of the variable 'weight' in the EAallowed routine (in the asm68.c file) from double to long double should provide the added precision. Examples of floating point immediate data shown below:

```
Ex.  fmove.s #5.0,fp0        ====> fmove.s 1_400000_E_2,FP0
```

The format on the disassembly is integer part_fraction field_E_exponent field, where the fraction bits represent weighting of 1/2 ,1/4,....etc. from the left to the right. The exponent bits represent the unbiased power that 2 should be raised to. A conversion to decimal can be accomplished by evaluating:

```
integer + evaluated fraction  x 2^exponent field
```

In the above example this equates to:

```
(1 + .25) x 2^2 = 5.0
```

NOTE: The monitor uses the round toward zero rounding mode in the assembler when assembling floating point immediate data.

3.7 ENTERING MMU INSTRUCTIONS

MMU instructions should NOT be entered with a size descriptor. The assembler defaults to the correct size.

```
Ex.     pmove (a0),tt1        ;assembles

        pmove.l (a0),tt1      ;does not assemble even though the operation
                              ;is a long operation.
```

A summary of the mnemonic representation of the MMU registers can be found in Table 3.5 and 3.6 for the 68030 and 68040 monitors respectively.

CHAPTER 4
SOFTWARE MODIFICATIONS

4.1 SOFTWARE OVERVIEW

The FBUG monitor uses 4 header files. These header files are briefly described below.

TARGETSYS.H
This file is the header file that is used to configure the monitor to the target system. All of the predefined variables that are used by the monitor are found here. It is accessed when any call to #include "userdef.h" is made.

USERDEF.H
This is the file that defines all of the structures that are used by the monitor.

TEXTDEF.H
This file has all of the strings that are used by the monitor. It is also where most of the structures that are used by the monitor are implemented.

VARDEF.H
This file has all of the global variables that are used by the monitor.

The flow of the FBUG monitor is outlined below:

asmstartport.s ===> coldstartport.c ===> main.c ----|
^-------|

The first source file (asmstartport.s) initializes the microprocessor to a known state and then branches to the main{} statement in the coldstartport.c routine. The coldstartport.c routine performs all of the initialization for the monitor before branching to the mainloop{} routine in the main.c file. The mainloop{} routine is a continuous loop used by the monitor. From this loop the various commands of the monitor are called.

4.2 ADDING/DELETING COMMANDS

The process of adding or deleting commands from the monitor is a relatively simple process. Adding a command involves making 2 additions to the textdef.h header file and then developing/including the command description file with the monitor.

The two changes that need to be made to the textdef.h header file are:

1.) include the function name that implements the command as an externally defined routine. For example the line in the textdef.h file that states

extern trcmd(),pfcmd(),ascmd(),sdcmd(),ctcmd();

accomplishes this for those five commands.

2.) include the command in the f[] structure with the appropriate information. For an example of how this is done the "ct" command entry is discussed:

The line

{"ct Confidence Test \n",ctcmd,"ct \n"}

includes the ct command in the f[] structure. The first textstring (i.e.. ct Confidence Test \n) is the string that is used for the command listing print out performed when the help or ? commands are executed. Also, the first word of this string is the command name that is searched for when attempting to execute this command. The function name "ctcmd" is the name of the function that is executed when this command is used by the monitor. The last text string (i.e.. ct \n) is the string that describes the type of syntax used with this command.

Next, the command description file has to be written and included in the makefile of the monitor (see CHP6 LINKING INFORMATION for more information concerning the linking process). The calling convention for the functions which perform the command is to pass the function two parameters. These two parameters are argc and argv. They are the number of arguments on the command line and the actual command line respectively.

Deleting a command is accomplished by following an analogous path. Removing the 2 lines that incorporate a command into the monitor and removing the command from the linking process effectively remove the command from the monitor.

4.3 SWITCHING DRIVERS

The I/O driver used by the monitor in version 1.0 is for a MC68681 DUART. If the target system uses some other driver, then the following changes needs to be made:

1.) The DUART initialization performed in coldstartport.c has to be modified to accomplish the same results with the new driver.

2.) The user definitions associated with the new driver has to be updated in the targetsys.h file. (i.e.. the offsets to the various buffers of the new driver)

3.) It may be necessary to modify the getch and putch routines found in getline.c and print.c respectively.

CHAPTER 5
TARGET SYSTEM CONFIGURATION

5.1 INTRODUCTION

The FBUG monitor can be configured to support any of the 68xxx microprocessors. This is accomplished by a series of define statements found within the "targetsys.h" header file. By defining which configuration is supported additional software can be excluded from the code at compilation time to result in a smaller monitor. These space savings are generally accomplished based on the size of the assembler/disassembler for the associated configuration. The monitor can also be configured to a specific memory configuration.

5.2 ASSEMBLER/DISASSEMBLER

The assembler/disassembler supported in the 'AS' command is determined by the definition of the DEVICE, EMULATOR and COPROCESSOR variables found in the targetsys.h header file. The choices for DEVICE are 68000, 68008, 68010, 68020, 68030 or 68040. This also determines the programmer's model that is supported in the 'RD' and 'RM' commands. If the DEVICE selected is '68040' then the EMULATOR define is used to determine whether 68881/68882 commands not implemented in hardware should be supported. Placing a 'TRUE' here stipulates that all 68881/882 instructions should be supported. If there is no software emulation of these commands then a 'FALSE' should be placed here. If the DEVICE selected is either 68020 or 68030 then the COPROCESSOR (TRUE or FALSE) define statement determines whether the 68881/882 instruction set should be supported.

Note: As of release 1.0 the 68040 register model is visible but supervisor registers other than those implemented on the 68020 are not supported. As a result of this the monitor displays the entire 68040 register set and APPEARS to allow the user to modify these registers with a RM command. However, when executing target code (GO or TR command) the monitor does NOT use these values to initialize the state of the microprocessor before executing the target code. In addition, these registers are NOT captured by the monitor if they have been modified by target code.

5.3 MEMORY REQUIREMENTS

The "targetsys.h" file contains all of the variables that determine the target system configuration. These variables are listed below and needs to be changed to reflect the configuration of the target system.

```
/* ********************************************************************** */
/* ********************** MEMORY MAP FOR DEBUGGER ********************** */

#define DUARTALOC 0xffe02000     /* MC68681 duart channel a */
#define DUARTBLOC 0xffe02008     /* MC68681 duart channel b */

#define ROMLOC 0x100000          /* system ROM space                    */
```

```
                                /* .text segment should be loaded here      */
                                /* .data segment should be loaded here      */

#define SYSRAMLOC  0x000fe000
                                /* system RAM space                         */
                                /* NOTE: space must be reserved for the
                                         location of the .bss used by
                                         the monitor.                       */
                                /* Provided the .bss is not larger than
                                   1000 (i.e.. VBRLOC resides at SYSRAMLOC
                                   + 1000) placing the .bss from the
                                   monitor at this location is allowed      */

#define ISPLOC SYSRAMLOC+0x2000 /* defines for monitor       */
#define MSPLOC SYSRAMLOC+0x1c00
#define USPLOC SYSRAMLOC+0x1b00
#define VBRLOC SYSRAMLOC+0x1000

#define USERPCLOC 0x4000        /* defines for usercode      */
#define USERVBRLOC 0x0

/* ************************************************************************ */
#if(DEVICE<68020)
#define USERSSPLOC USERVBRLOC+0x2000      /* Note: Only one of these is
                                             used based on the device
                                             being implemented */
#else
#define USERISPLOC USERVBRLOC+0x2000
#endif
/* ************************************************************************ */

#define USERMSPLOC USERVBRLOC+0x1c00
#define USERUSPLOC USERVBRLOC+0x1b00

#define SRSTART 0x2700            /* status register      */

**************************************************************************
```

The DUART assumed is a MC68681 and its respective channels are found at the stated locations. If this is not the case, then the initialization routine in coldstartport.c has to be changed.

The next 5 locations describe the memory map of the system used for the port.

The next location describes the value that is placed in the VBR register.

NOTE: This is the location of the vector offset table. This space is used by the monitor to set up the offset table.

The next 5 locations describe the initial values that the monitor uses for "user code".

The next location describes the value that is placed in the user's VBR register.

NOTE: This is the location of the user's vector offset table and is also initialized by the monitor.

SRSTART is the initialized value of the MC68xxx Status Register

CHAPTER 6
LINKING INFORMATION

6.1 INTRODUCTION

The FBUG monitor is a stand alone software package written in C and developed under the UNIX environment. The combined .text and .data section of the object file varies from approx. 54K to 87 bytes of memory. Different configurations are summarized below in Table 6.1. The purpose of this chapter is to describe the method to compile and link the various files that make up the FBUG monitor. There is also a brief description of the necessary changes that need to be made in order to port the FBUG monitor to a MC68xxx based board.

TABLE 6.1 68000/68HC000/68008/68010 REGISTERS

DEVICE	COPROCESSOR	EMULATOR	SIZE (KB)
68000/008/010	----NA-----	---NA---	54K
68020	FALSE	---NA---	61K
68020	TRUE	---NA---	81K
68030	FALSE	---NA---	65K
68030	TRUE	---NA---	87K
68040	----NA-----	FALSE	82K
68040	----NA-----	TRUE	84K

NOTE: ----NA---- represents a don't care for that particular DEVICE.

6.2 DESCRIPTION OF FBUG(68xxx) MAKEFILE

The makefile used to create the FBUG monitor under the UNIX operating system is shown below and can be found in the file named "makefile" in this directory.

MAKEFILE

```
portbug : coldport.o main.o printport.o getlineport.o general.o \
        asmprocport.o \
```

```
        bf.o bm.o br.o bs.o dc.o he.o lo.o rd.o rm.o tm.o \
        mm.o md.o go.o tr.o sd.o as.o asm68.o disasm68.o asmcode68.o \
        asmhandler.o handler.o  asmstartport.o conf.o fproutines.o 68def.o
        ld ./ifilemb  asmstartport.o asmhandler.o coldport.o \
        printport.o getlineport.o general.o asmprocport.o main.o bf.o bm.o \
        bs.o dc.o he.o lo.o rd.o rm.o tm.o mm.o md.o go.o tr.o br.o \
        sd.o as.o disasm68.o asm68.o asmcode68.o handler.o conf.o \
        fproutines.o  68def.o \
        -m -o fbugxxxd >memmap
        ubuilds fbugxxxd

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
NOTE: In fbugxxxd     xxx ===> 000,008,010,020 etc...
                      d ===> T or F for whether Coprocessor supported
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

main.o :        main.c userdef.h textdef.h vardef.h
        :
        :
        :
        :
        :
asmproc.o : asmproc.c userdef.h textdef.h vardef.h

asmstartport.o : textdef.h vardef.h userdef.h targetsys.h
                 /lib/cpp asmstartport.s  > temp.s
                 as  -o asmstartport.o temp.s
                 rm temp.s
asmcode68.o : textdef.h vardef.h userdef.h targetsys.h
                 /lib/cpp asmcode68.s  > temp.s
                 as  -o asmcode68.o temp.s
                 rm temp.s
asmhandler.o : textdef.h vardef.h userdef.h targetsys.h
                 /lib/cpp asmhandler.s  > temp.s
                 as  -o asmhandler.o temp.s
                 rm temp.s

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        NOTE:  In order to ease the porting of the monitor to a
               specific system, a C preprocessor is executed on the
               three (3) .s files above.  This performs a text
               substitution into these files from the targetsys.h
               header file.  If the development system being used does
               not support this capability then these text substitutions
               must be made manually.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Executing the "make" command under UNIX with this makefile produces the file "fbug.mx" which is the stand alone monitor. The make command performs the following actions:

1. Checks to see if all of the current .o files on which fbugxxxd depends are current. If not then compile the source files to create those object files. It also checks to verify that the files on which these files depend on are up to date. If they are not then they are compiled first.

/* ********************************!!!!!!!!!********************************** */
NOTE: These dependencies are not conclusive and if a change is made to any of the header files then all of the files should be recompiled until a thorough understanding of the interdependencies between the modules is achieved.

/* ********************************!!!!!!!!!********************************** */

2. After all the required .o files have been brought up to date they are linked together in the order described after the "ld" command. The "ifilemb" file is used as a ld directive to describe the memory configuration of the target system. In this instance, the target system has the following memory configuration:

00000000-00000255 RAM
000FE000-00100000 RAM
00100000-00110000 ROM

The ifilemb informs the UNIX linker of this target configuration and also where to place each section. In this instance the .text section is being placed into ROM and the .data and .bss section are being placed in RAM. Refer to the "ifilemb" file in this directory.

/* ********************************!!!!!!!!!********************************** */
NOTE: Space has to be made available for both the monitor's vector offset table and the user's vector offset table. (i.e.. 256 32-bit words has to be available at locations VBRLOC and USERVBRLOC. (i.e.. 00000000-00000255 RAM)

NOTE: The monitor uses approx. $1000Kb ram for the .bss variables (placed ,at SYSRAMLOC). Next the vector offset table is placed at SYSRAMLOC + $1000. Then the stack space used by the monitor is placed above this. Using the default configuration provided requires that approx. $2500 bytes of RAM be reserved for the monitor beginning at location SYSRAMLOC and that 256 32-Bit words be reserved at the USERVBRLOC (which is at location 0 in the released version). If user code is run then the user is responsible for making sure that the stack space used by the user does not interfere with the monitors space.
/* ********************************!!!!!!!!!********************************** */

3. These files are now linked and this file is labeled "fbugxxxd"

4. Next, the "ubuilds fbugxxxd" command creates an S-Record from the fbug file that was previously created. This S-Record can be used to download to systems that have the capability to load S-Records.

The environment that is being worked under has to be able to perform the functions stated above in order for the user to create their FBUG object file.

6.3 MONITOR USED TO BOOT THE SYSTEM

If it is desired for the monitor to be at the reset location (i.e.. logical 0 in memory),then the following two lines should be added above the start: routine in the asmstartport.s file:

```
INITSP:long 0x100000
INITPC long start
```

NOTE: The ifilemb directive must still accurately reflect the location of ROM on the system.