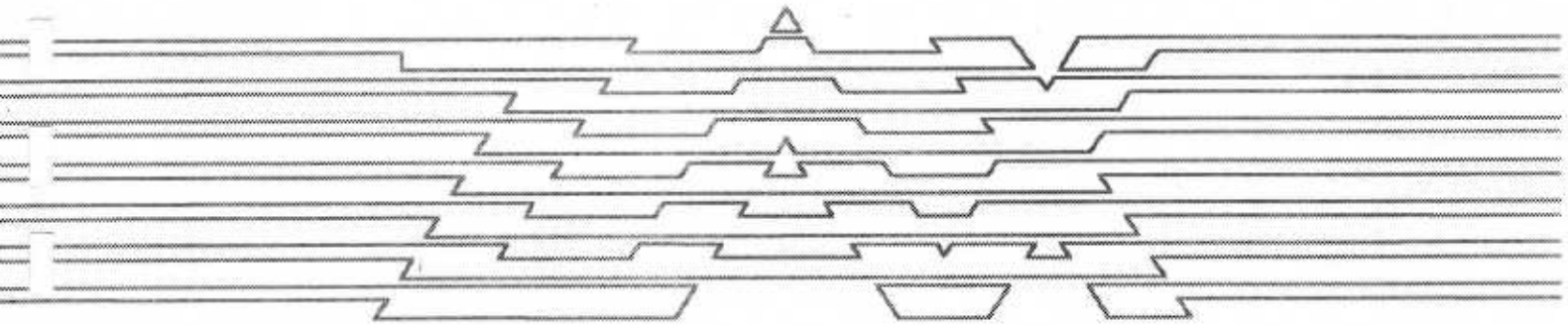
ERX318 for 68000

User's Manual



Zax Corporation

USER'S GUIDE

ZAX

ERX318 for 68000 • 68010

User's Guide

Rev. E1.01

OUTLINE

The **ERX318** is an in－circuit emulator for developing and evaluating hardware and software of microprocessor products.

The **ERX318** has automated most of the true debugging procedures for development. For example, there is the **auto－breakpoint** function that sets breakpoints in all instruction steps with symbols by downloading a program ; **batch** and **macro** functions that enable up to 10－level nesting ; **stub function** that realizes subroutines substitution and I/O simulation. In addition, the **ERX318** has the measurement functions such as **performance** and **coverage** analysis for improved product quality.

ERX318 for 68000 User's Guide

CONTENTS

1. SPECIFICATION

1.1 Configuration

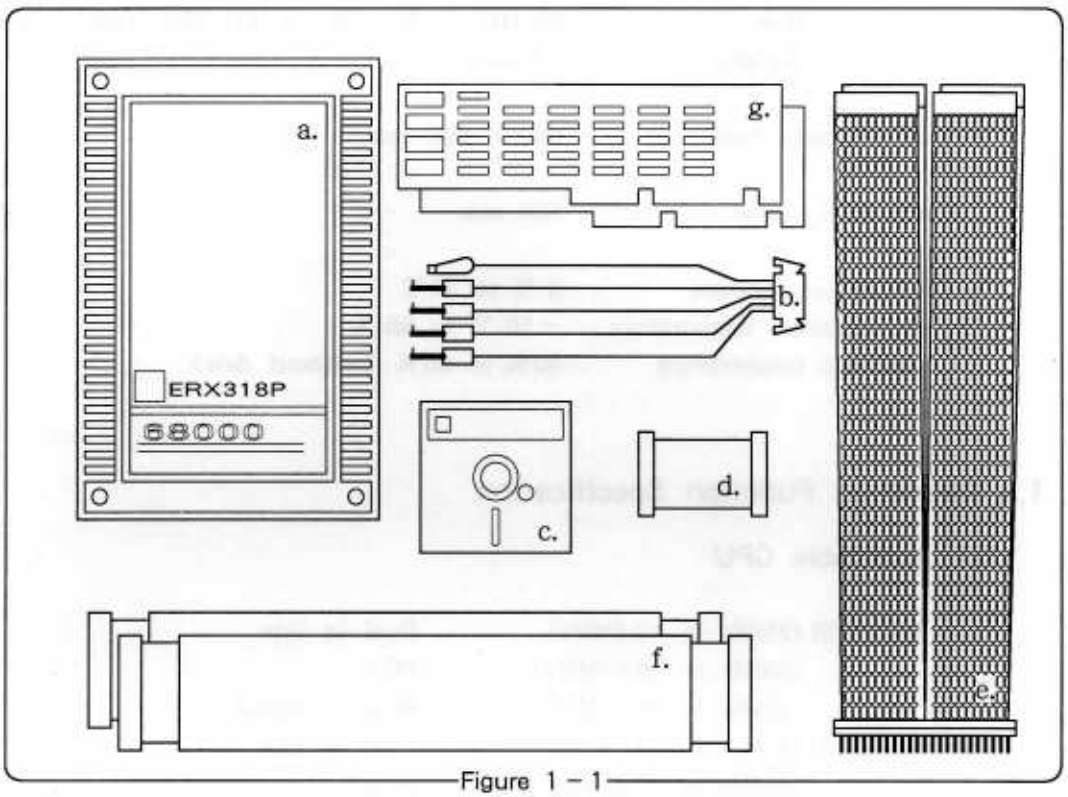


Figure 1 - 1

List of fixtures

a. One ERX Pod
b. One external probe
c. One ERice Diskette
d. One bus cable
e. One in - circuit probe cable
f. Two connecting cables
g. Two internal boards

NOTE

1.2 General specification

Probe box		
	Size	85 (H) × 220 (W) × 316 (D) mm
	Weight	3.00 Kg
In – circuit probe		500 × 500 mm
External probe		500 mm
Usage temperature		0 ℃ to 35℃
Preservation temperature		– 10 ℃ to 55 ℃
Ambient temperature		30% to 85% (without dew)

1.3 Emulation Function Specification

1.3.1 Applicable CPU

MC68000 (8MHz to 12.5MHz)	Dual in line
(8MHz to 16.67MHz)	PGA
(8MHz to 16.67MHz)	PLCC (Option)
MC68010 (4MHz to 12.5MHz)	Dual in line
(4MHz to 12.5MHz)	PGA

1.3.2 Emulation memory function

Capacity of emulation memory	256K bytes	
Memory space	16M bytes	
Emulation memory specification unit	4K bytes	
Mapping types	Read – only memory	(RO)
	Read/write memory	(RW)
	User memory	(US)
	Nonexistent memory	(NO)

NOTE

1.3.3 Event Trigger function

Range of address space	Up to 64K bytes/64K – byte boundary
Range of data	Up to 16 bits
Status types	Memory read, memory write, memory access, Inturrupt Acknowledge
Pass count range	16 bits
Number of events	Four blocks
Sequential specification	Four level (Max)
External trigger specification	level trigger [High level / Low level]
	edge trigger [Positive edge / Negative edge]

1.3.4 Break function

Break function	Stops in – circuit state
Step break function	Stops after one instruction is executed
Map state break functions	Function that stops emulation when data is written in the read – only (mapped) memory area or when the non – existent memory is accessed.

1.3.5 Real – time trace function

Capacity of trace memory	8191 – word storage
Trace data widths	
Address width	23 bits
Data width	16 bits
Status width	7 bits
Auto – start function	Starts trace simultaneously when emulation starts
Auto – stop function	Stops trace simultaneously when emulation stops
Event stop function	Stops trace if an event occurs The delay cycle function that sets a trigger in a definite cycle after an event has occurred is also available.

NOTE

Freeze function	Determines whether trace should be stopped automatically when the trace memory is full or the storage address counter should be rotated
Freeze break function	Stops trace automatically and generates a break signal when the trace memory is full.

1.3.6 Coverage function

Coverage memory space	64 K bytes × 4 ch/64K – byte boundary

1.3.7 Performance function

Emulation time counter	48 bits
Measurement time counter	48 bits × 2ch
Measurement number counter	
Resolution	1 μs
Error range	±5%
Start function	Starts measurement if an event occurs
Stop function	Stops measurement if an event occurs

NOTE

2. SYSTEM CONFIGURATION

2.1 System Configuration

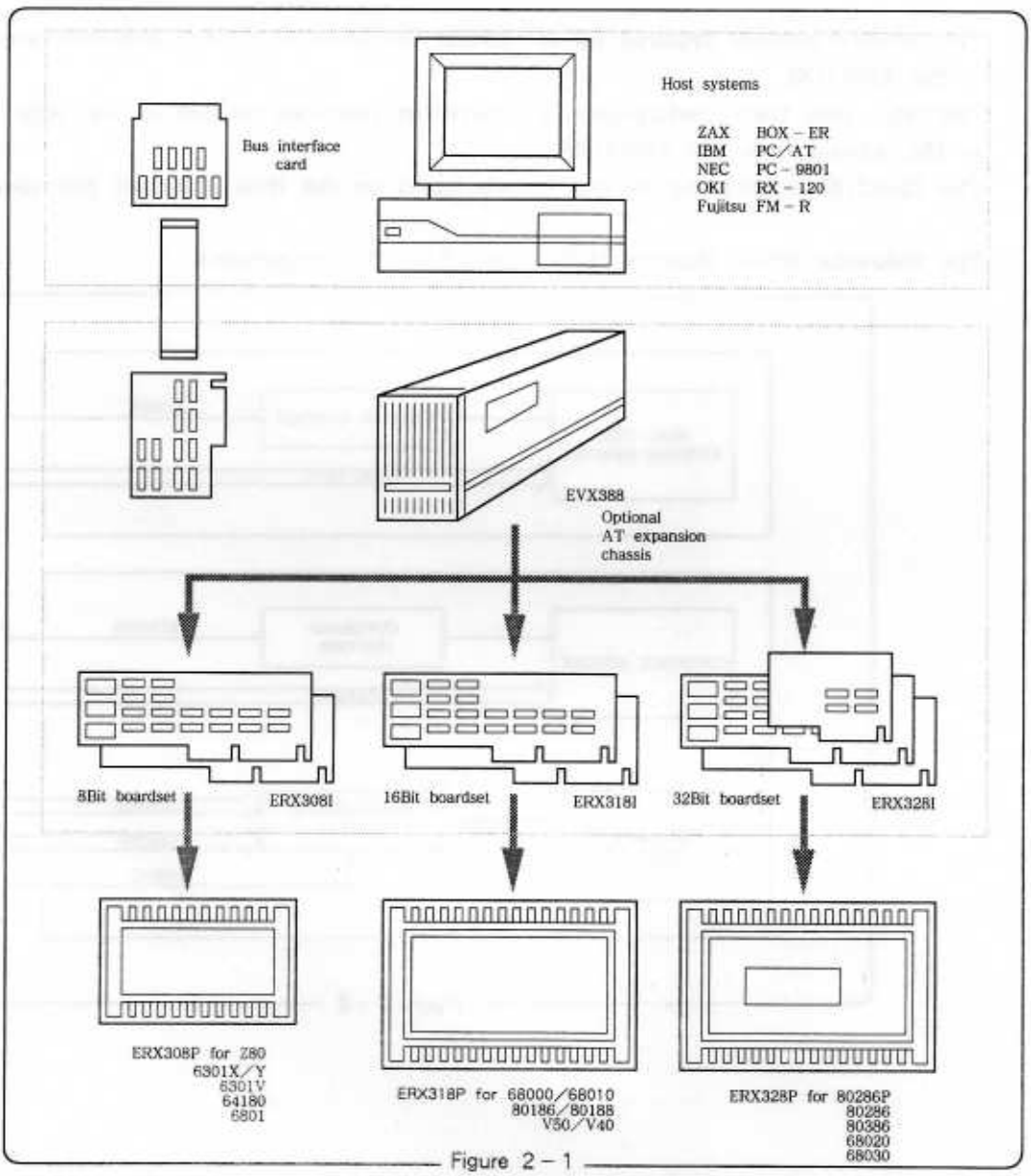


Figure 2 – 1

NOTE

2.2 ERX318I Configuration

The common sections required for in-circuit emulation of a 16bit processor are mounted in the ERX318I.
The real-time trace, performance, and coverage functions operate in real time according to the signals from the ERX318P.
The ERX318I is made up of two boards based on the Host extended bus specification.

The following block diagram shows the ERX318I configuration.

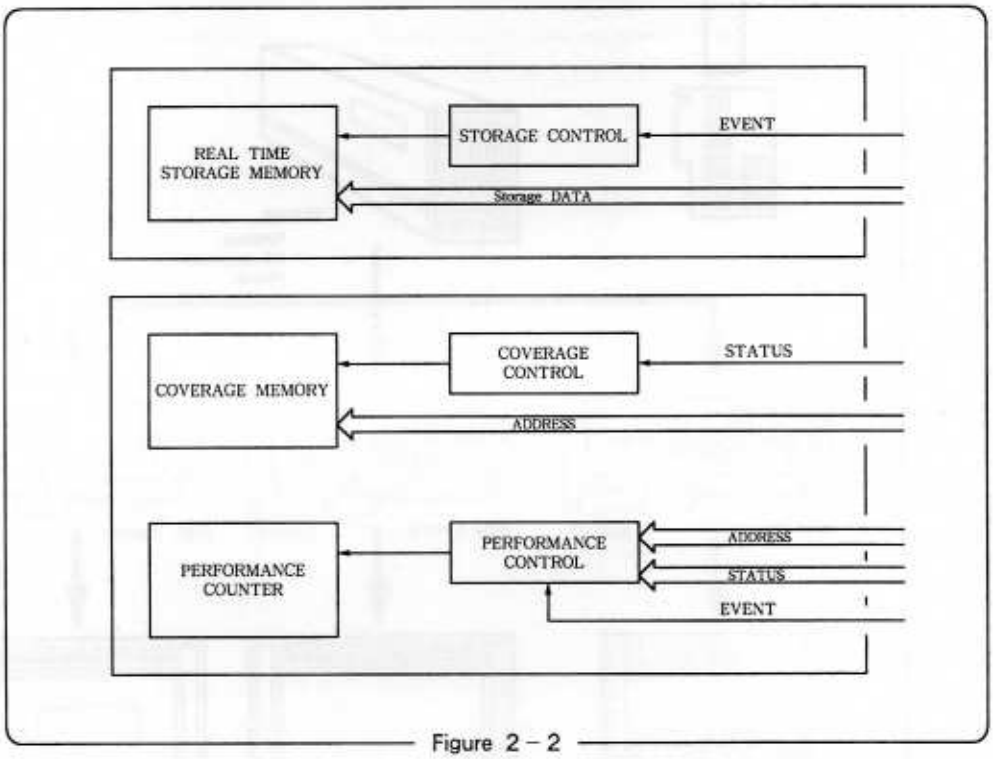


Figure 2-2

NOTE

2.3 ERX318P Configuration

The ERX318P is the unit outside the Host mainframe. The control hardware specific to the target processor is mounted in the ERX318P.
It contains the break sequence control, event control, emulation control, emulation memory control, and memory and I/O access control programs and, the target probe.

The following block diagram shows the ERX318P configuration.

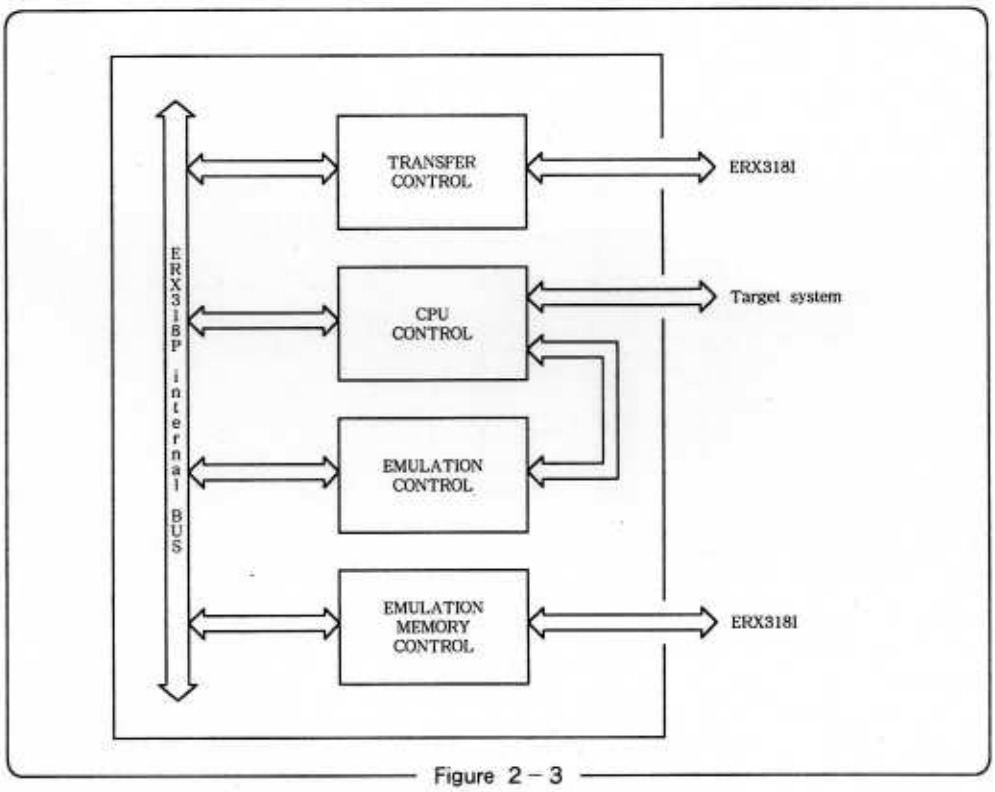


Figure 2 - 3

NOTE

NOTE

3. ERX318 START AND STOP

3.1 Turning power on and off

3.1.1 Turning power on

When the in-circuit operation is not performed

① Turn on the power switch of the host computer.
② Start the ERX software.

When the in-circuit operation is performed

① Turn on the power switch of the host computer.
② Start the ERX software.
③ Turn on the power switch of the target system.

3.1.2 Turning power off

If the in-circuit operation has not been performed

① Terminate the ERX software.
② Turn off the power switch of the host computer.

If the in-circuit operation has been performed

① Turn off the power switch of the target system.
② Terminate the ERX software.
③ Turn off the power switch of the host computer.

NOTE

3.2 Start and stop

3.2.1 File configuration

The basic file configuration contains the following three files :

a. **ERX68K.EXE**
b. **ERX68K.HLP**
c. **ERX68K.MAC**

EXE file	ERX control programs
HLP file	Contains ERX help messages
MAC file	Loads macro when the control program is started. This occurs only when the file name is the same as the control program file name.

File name can be changed. If a file is to be renamed, the other basic files must all be renamed. As a result, prompting default characters are changed automatically.

3.2.2 How to start

```
> program name  [batch_name] < CR >
```

program_name	: Specify the name of a file to be started.
batch_name	: Specify the name of a batch job to be executed after start. If batch name is omitted, batch processing is not executed.

If a macro file with the same name as **program_name** exists, macros are loaded.

NOTE

3.2.3 How to stop

```
>Q<CR>
```

Use the **Quit** command to stop ERX control software.
The [Ctrl] + [C] keys are registered for forceful stop.

NOTE

NOTE

4. HANDLING EACH UNIT

4.1 Interface connector

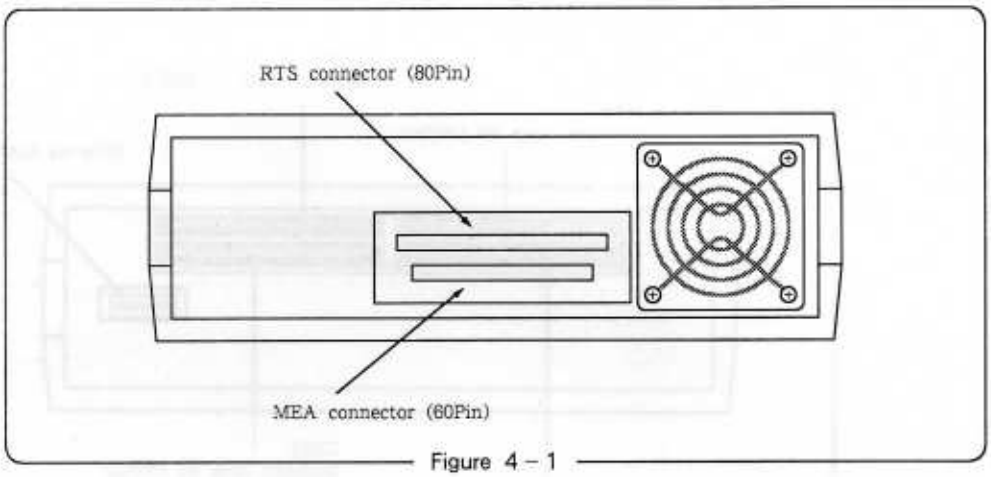


Figure 4 - 1

RTS interface connector — Used for connecting ERX318I and ERX318P.
It must be connected to the ERX318I RTS (R - 901) port.

MEA interface connector — Used for connecting ERX318I and ERX318P.
It must be connected to the ERX318I MEA (R - 902) port.

NOTE

4.2 Incircuit Connector

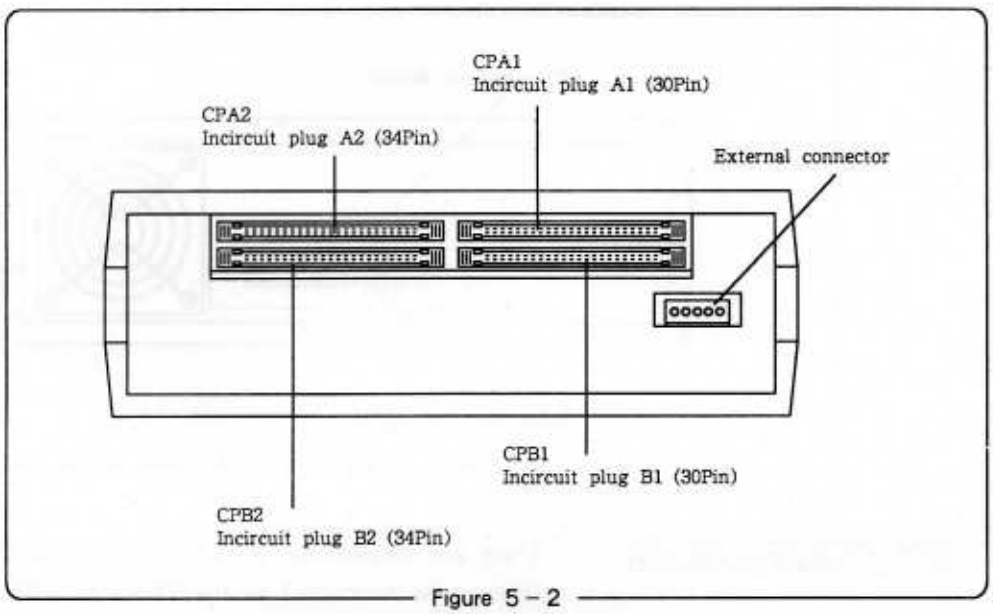


Figure 5 – 2

Incircuit connector	Incircuit socket
CPA1	CSA1
CPA2	CSA2
CPB1	CSB1
CPB2	CSB2

Table 4 – 1

CAP1 CPA2 CPB1 CPB2 — Connectors for connecting the in – circuit probe.Table 4 – 1 indicates the correspondence between the connectors and incircuit probe outlets. If a target system is not connected,the incircuit probe need not be connected.

External connector — Connector for connecting the external probe

NOTE

Connect the in – circuit probe surely. If connection is abnormal, The ERX may be damaged.

4.3 Incircuit Probe

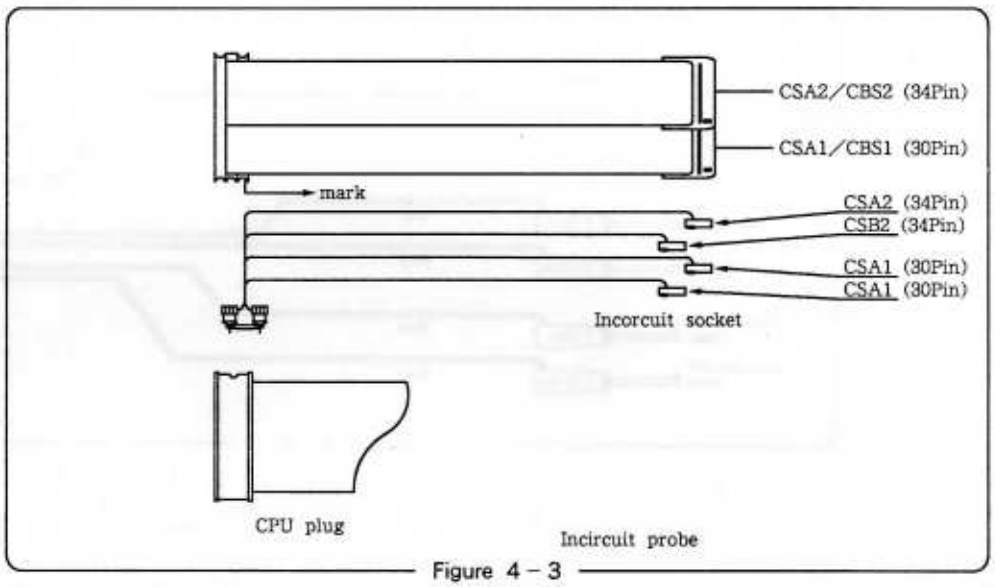


Figure 4 – 3

Incircuit connector	Incircuit socket
CPA1	CSA1
CPA2	CSA2
CPB1	CSB1
CPB2	CSB2

Table 4 – 2

CSA1 CSA2 CSB1 CSB2 Connector for connecting ERX318P for 68000 mainframe. Table 4 – 3 indicates the correspondence between the connectors and incircuit probe outlets.
If the target system is not connected, the incircuit probe need not be connected.

NOTE

Connect surely. If connection is abnormal, The ERX may be damaged.

4.4 External Probe

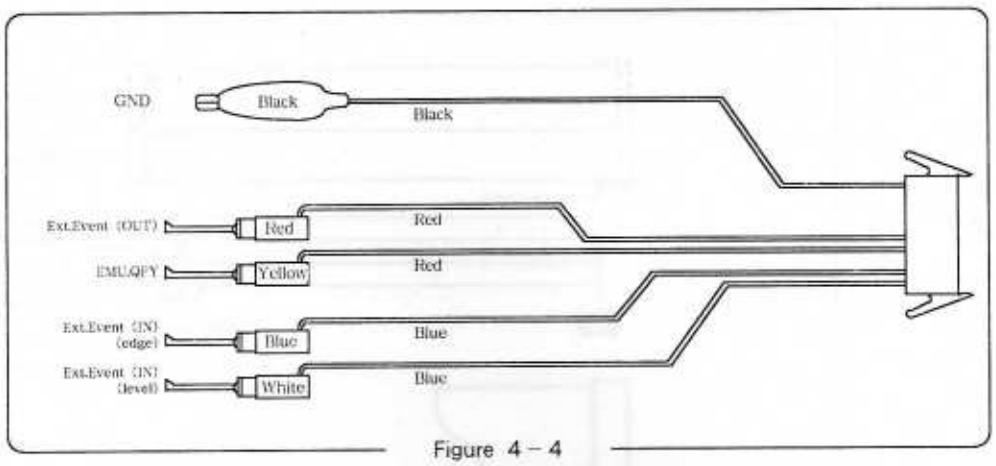


Figure 4 – 4

GND : Ground
Ext.Event (OUT) : Event trigger output
EMU.QFY : Emulation qualification
Ext.Event (IN, edge) : Event edge trigger input
Ext.Event (IN, level) : Event level trigger input

Ground — Connect to the ground line of the target system.

Event trigger output — When the event trigger condition is met, a low TTL level signal is output.
This function can be used for output of triggers of instruments such as a logic analyzer and a synchronous scope.
Trigger output can be controlled by the **Trigger** command.

Emulation qualification — While The ERX is emulating, a high TTL level signal is output. While emulation is stopped, low level signal is output.
Unnecessary signals can be discarded by entering this function in an instrument such as logic analyzer.

NOTE

Event edge trigger input

Event edge trigger can be input in the event function from outside hardware. If event level trigger input is used, this cannot be used.
Input can be controlled by the **Event** command.

Event level trigger input

Event level triggers can be input in the event function from outside hardware.
If event edge trigger input is used, this cannot be used.
Input can be controlled by the **Event** command.

NOTE

NOTE

5. CPU EMULATION FUNCTIONS

5.1 Probing Target System

5.1.1 12.5MHz version

(1) Address bus

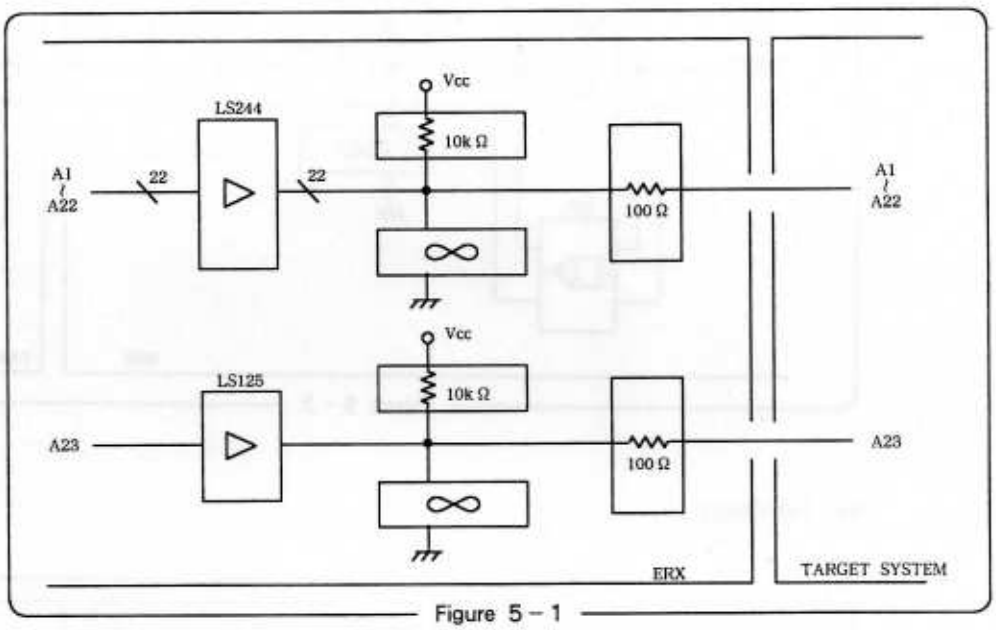


Figure 5 – 1

(2) Data bus

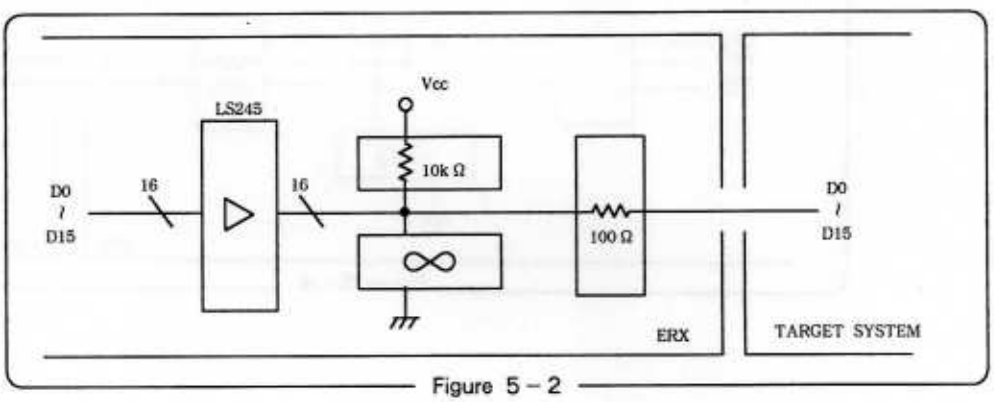


Figure 5 – 2

NOTE

(3) $\overline{\text{RESET}}$,$\overline{\text{HALT}}$

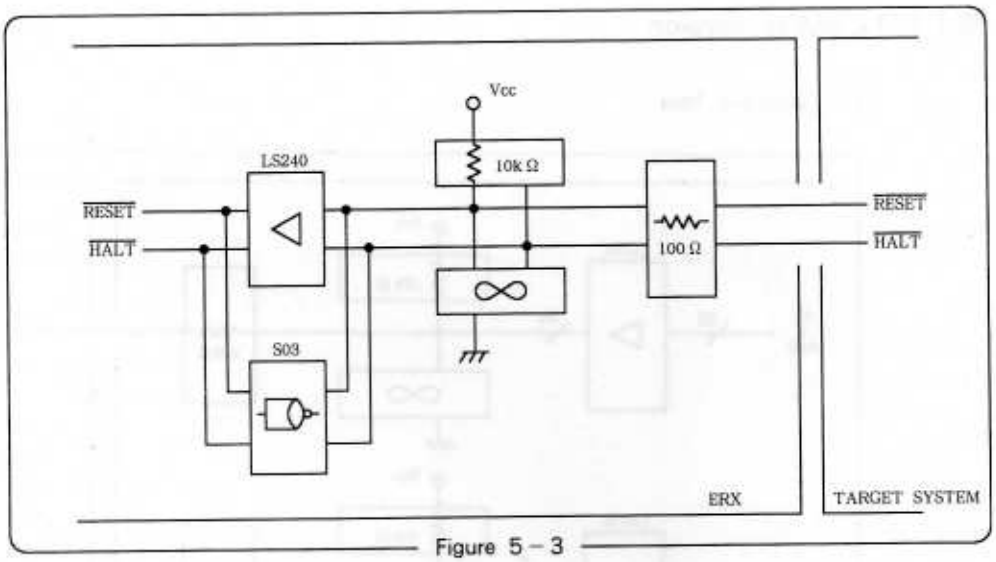


Figure 5 – 3

(4) Interrupt

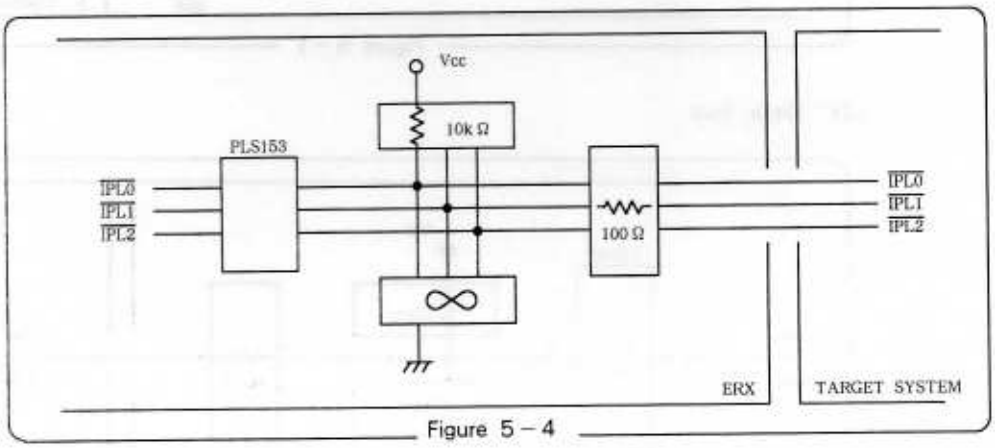


Figure 5 – 4

NOTE

(5) $\overline{BR}$,$\overline{VPA}$,$\overline{BERR}$,$\overline{DTACK}$

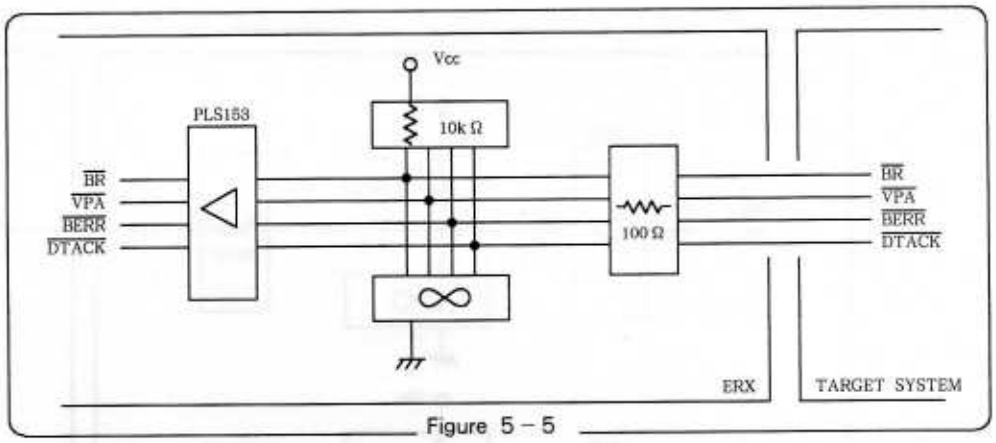


Figure 5 – 5

(6) Status

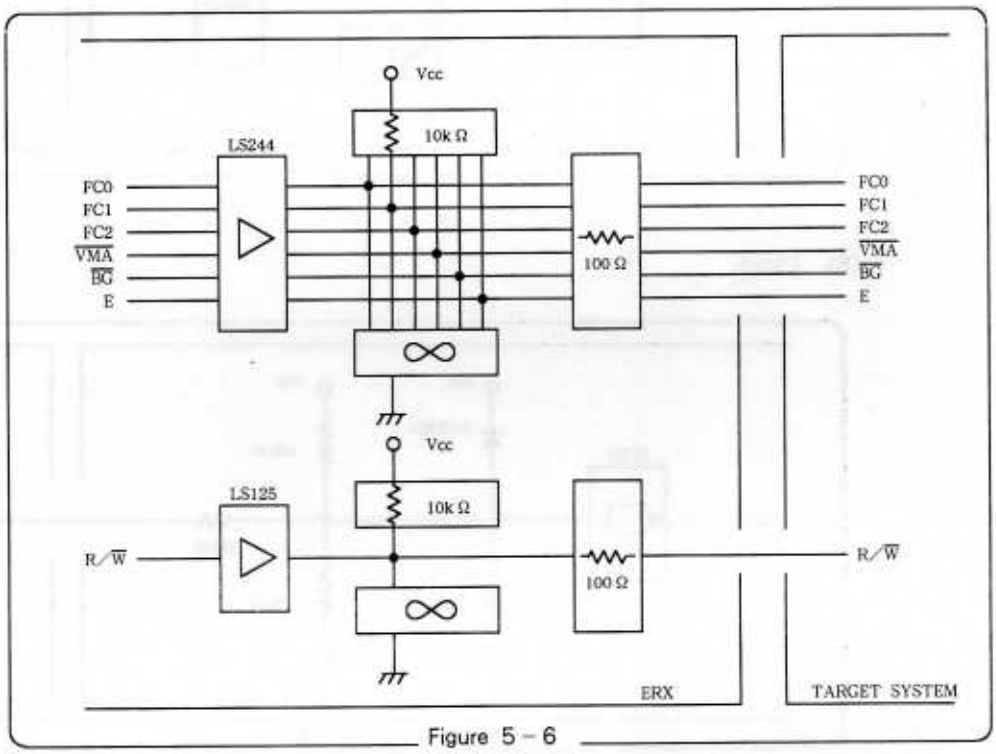


Figure 5 – 6

NOTE

(7) $\overline{AS}$, $\overline{DS}$

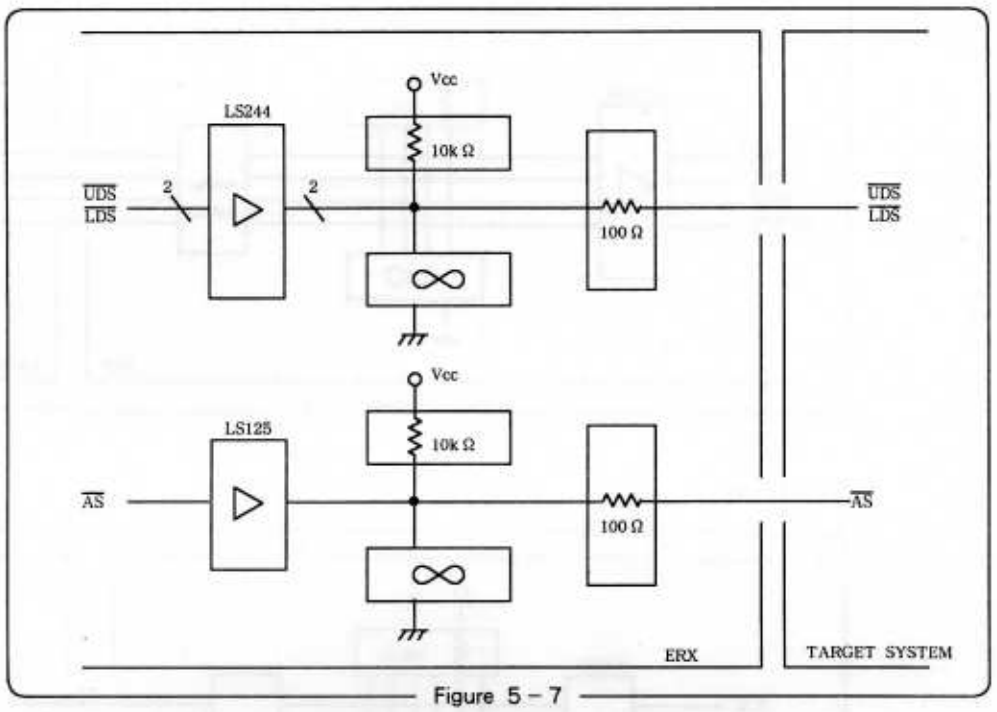


Figure 5 – 7

(8) Clock

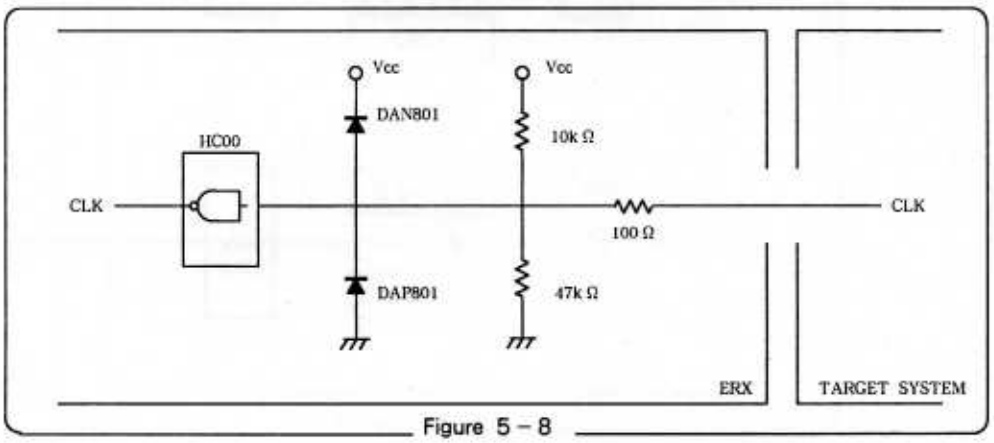


Figure 5 – 8

NOTE

(9) Power supply

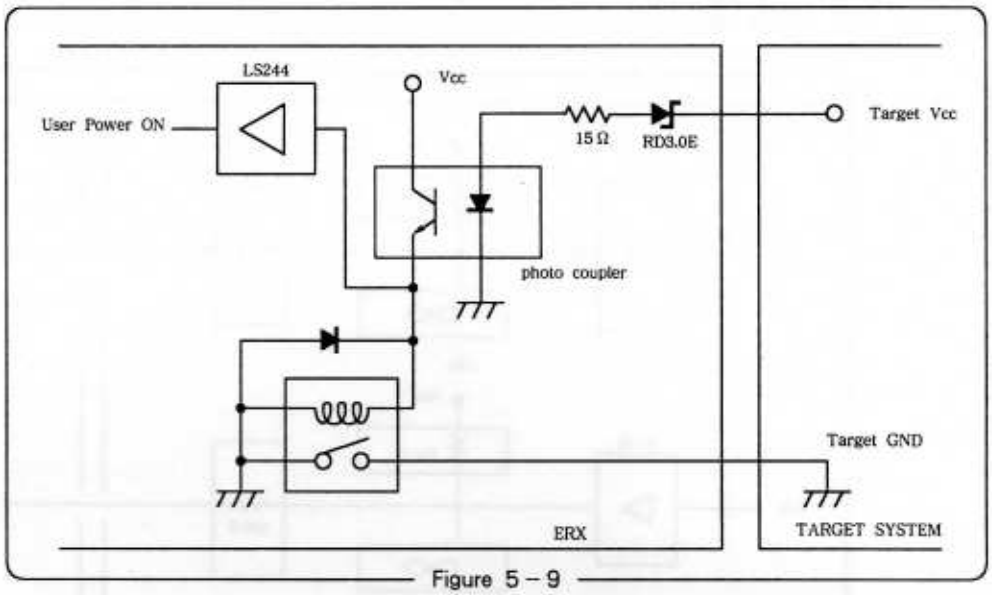


Figure 5 - 9

NOTE

5.1.2 16.67MHz version

(1) Address bus

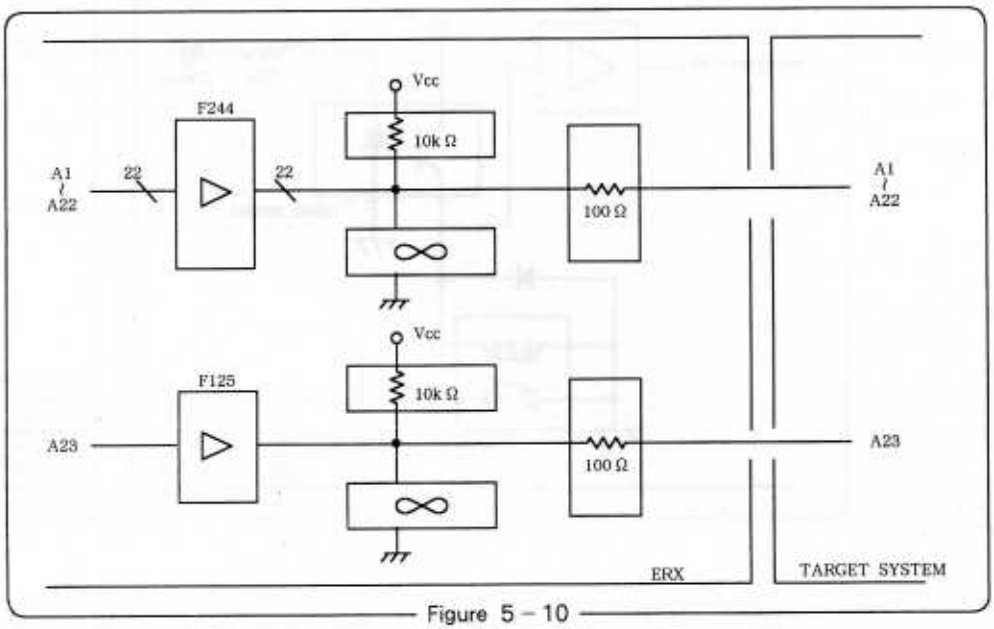


Figure 5 - 10

(2) Data bus

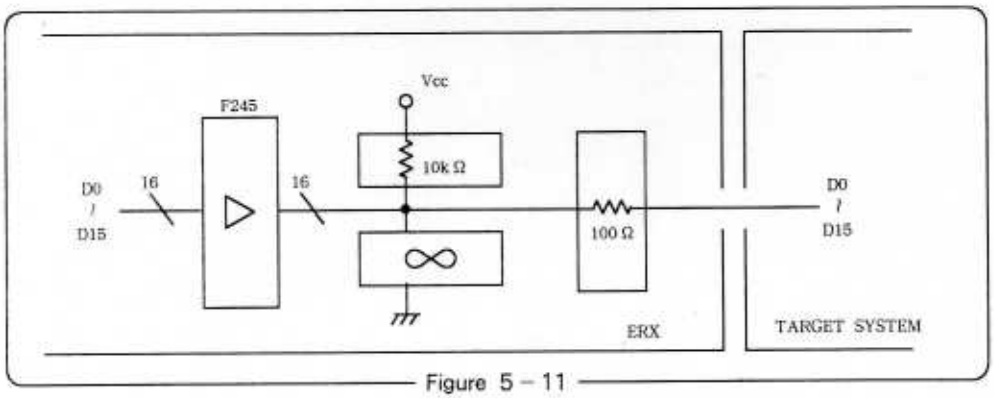


Figure 5 - 11

NOTE

(3) $\overline{\text{RESET}}$,$\overline{\text{HALT}}$

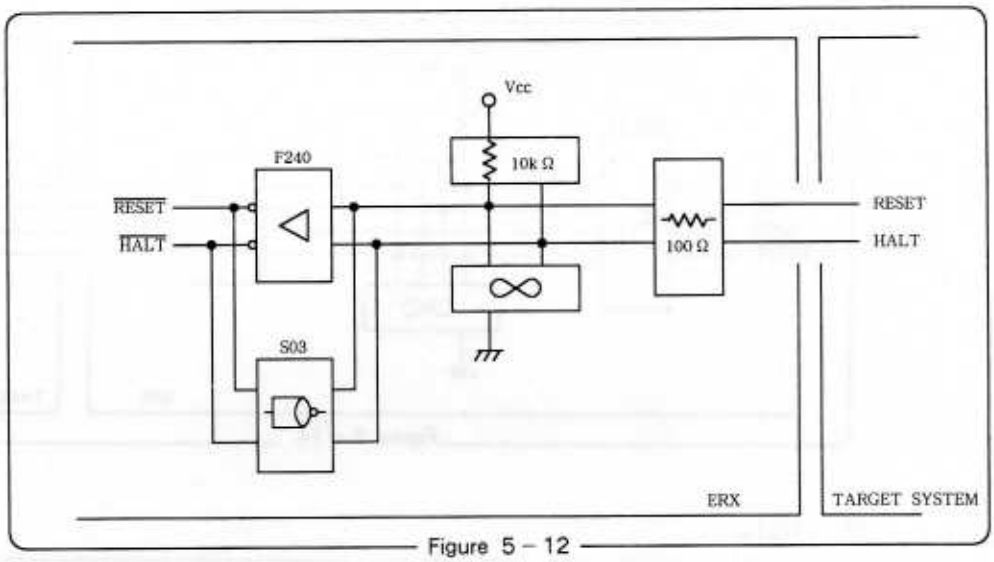


Figure 5 – 12

(4) Interrupt

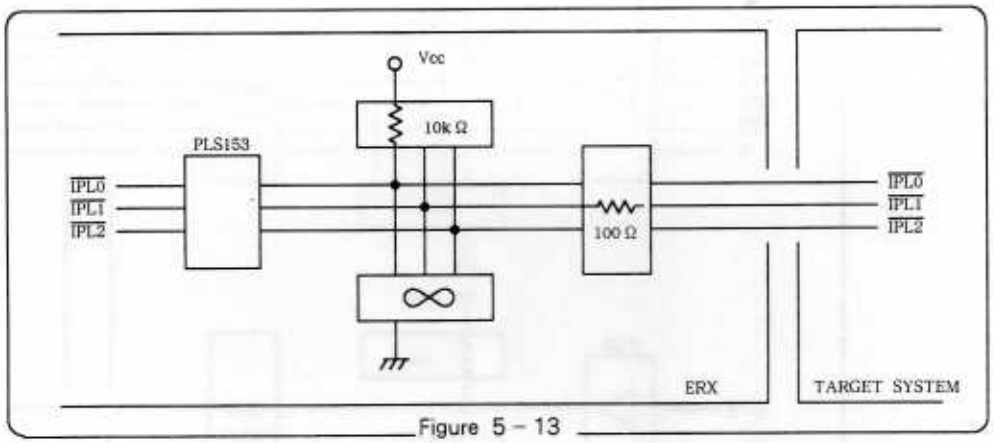


Figure 5 – 13

NOTE

(5) $\overline{BR}$,$\overline{VPA}$,$\overline{BERR}$,$\overline{DTACK}$

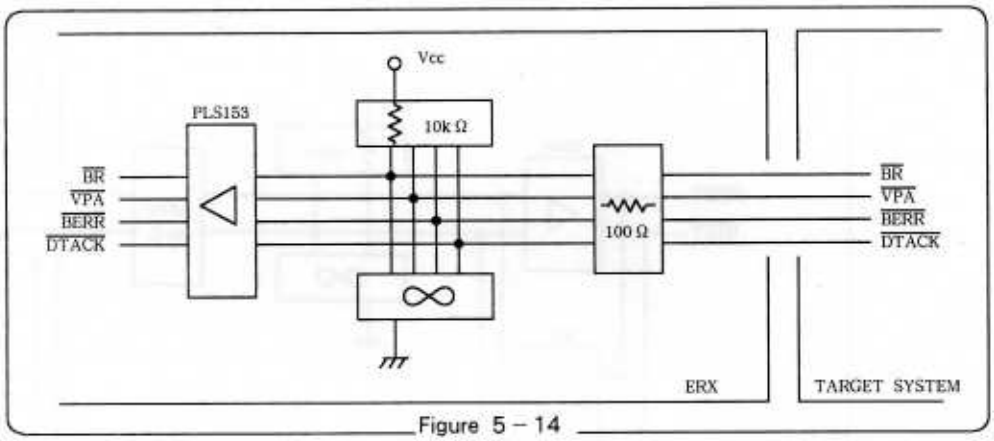


Figure 5 – 14

(6) Status

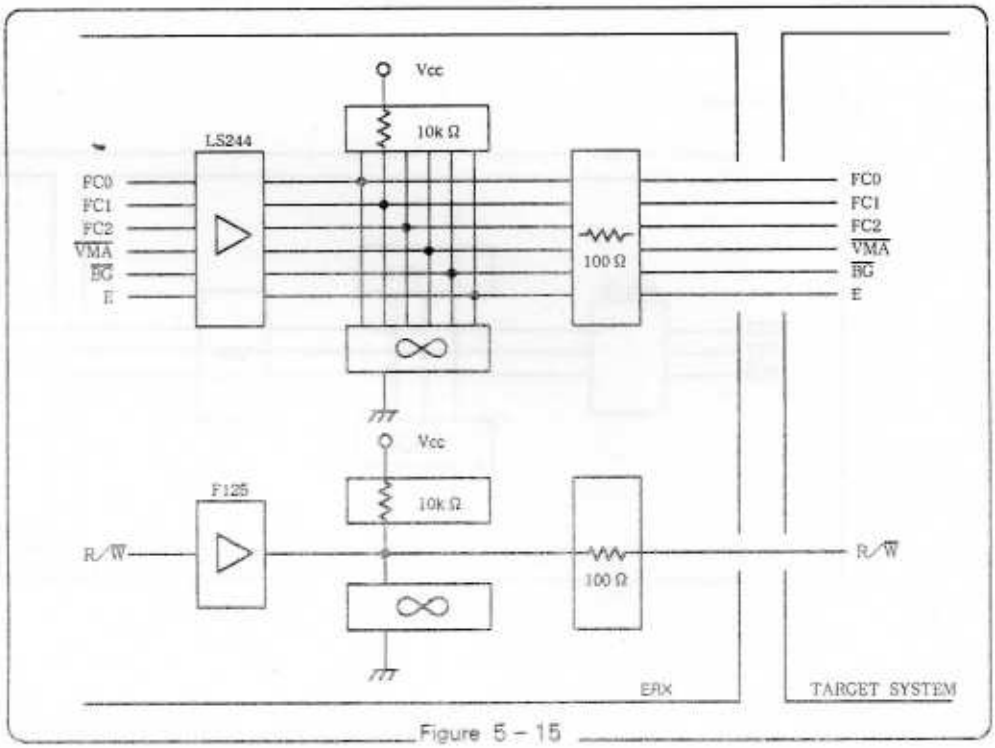


Figure 5 – 15

NOTE

(7) $\overline{AS}$, $\overline{DS}$

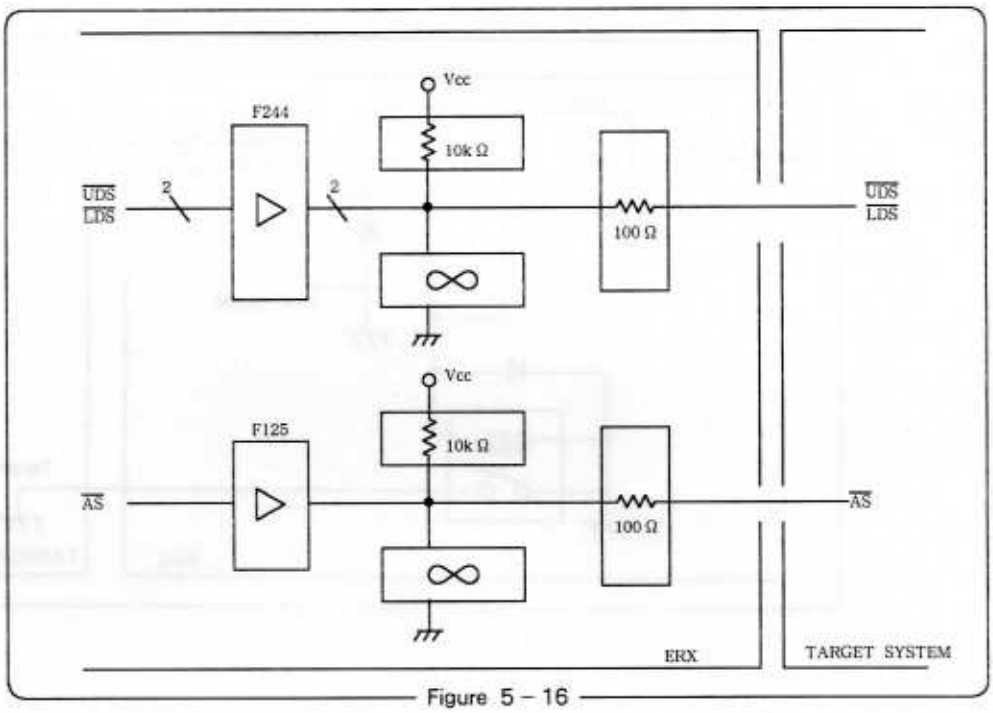


Figure 5 – 16

(8) Clock

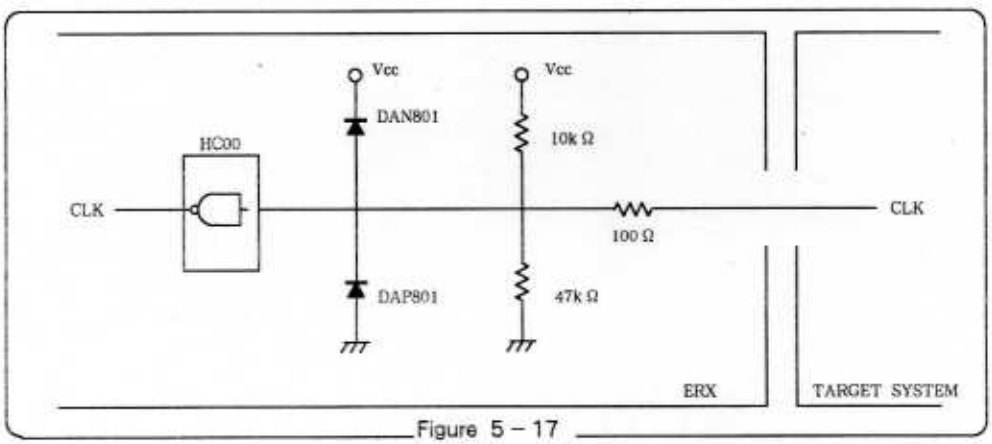


Figure 5 – 17

NOTE

(9) Power supply

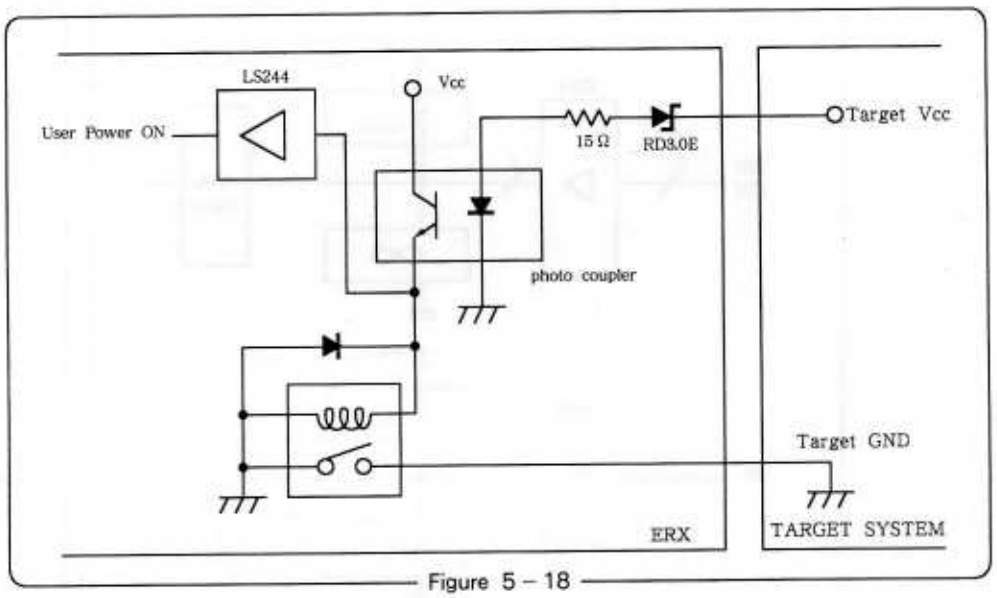


Figure 5 – 18

NOTE

5.2 Pin Control

MACHIN CYCLE	CONTROL				A1-A23	D0-D15	DTACK	VPA	BERR	IPL0~2	BR	HALT	RESET
	UDS/LDS	AS	R/W	FC0-2									
Inerrupt Acknowledge or CPU Space	L	L	H	H	OUT	IN	AC	AC	AC	N	AC	N	AC
Read	L	L	H	X	OUT	IN	AC	AC	AC	AC	AC	AC	AC
Write	L	L	L	X	OUT	OUT	AC	AC	AC	AC	AC	AC	AC
Bus Grant State	TS	TS	TS	TS	TS	TS	N	N	N	N	N	N	AC
HALT State	H	H	H	X	TS	TS	N	N	N	N	AC	AC	AC
RESET State	H	H	H	TS	TS	TS	N	N	N	N	N	AC	AC
Emulation Memory Read	L *1	L	H	X	OUT	TS	N *4	N	N *5	AC	AC	AC	AC
Emulation Memory Write	H *2	L	L	X	OUT	OUT	N *4	N	N *5	AC	AC	AC	AC
Emulation Break	H	X *3	X	X	OUT	TS	N	N	N	N	AC	AC	N
Pin Command	—	—	—	—	—	—	—	—	EN/DI	EN/DI	EN/DI	EN/DI*6	EN/DI

Table 5 – 1

Status : H = High Level
L = Low Level
X = H or L
TS = High Impedance
AC = Accept
N = Not Accept
– = Not Support

* 1 If D = Disable is set by the **EMSELect** command, the LDS/UDS signal is not asserted. The signal state is H.

* 2 If C = Enable is set by the **EMSELect** command, the LDS/UDS signal is asserted. The signal state is L.

* 3 If A = Disable is set by the **EMSELect** command, the AS signal is not asserted. The signal state is H.

* 4 If G = Enable is set by the **EMSELect** command, the DTACK signal is accepted. The signal state is AC.

* 5 If F = Enable is set by the **EMSELect** command, the BERR signal is accepted. The signal state is AC.

* 6 Even if HALT = Disable is set by the **Pin** command, the HALT signal is accepted when the device is reset.

NOTE

5.3 EMSEL Control

(1) EMSEL control functions

(A) Address strobe (AS) signal control
(B) Read/write (R/W) signal control
(C) Data strobe (LDS/UDS) signal control in write cycle
(D) Data strobe (LDS/UDS) signal control in read cycle
(E) Data bus signal control
(F) Bus error (BERR) signal control
(G) User wait (DTACK) signal control
(H) Double bus fault break control
(I) Auto wait signal control
(J) Time-out break signal control
(K) Control of wait time of the auto wait signal
(L) Control of clock count of the time-out break signal

(2) Control of the user wait (DTACK) and auto wait signals

5.3.1 EMSEL control functions

(A) Address strobe (AS) signal control

When emulation is broken, output of the address strobe (AS) signal to the target system can be controlled.

(1) Enable

When emulation is broken, the AS signal is asserted to the target system in a repetitive pattern of idle loop command fetch of the emulation CPU. In emulation, the AS signal is asserted in an access cycle of either the emulation memory (RO/RW mapping is set) or user memory (US mapping is set).

(This is the default at initialization.)

(2) Disable

When emulation is broken, the AS signal is not asserted to the target system (high level).

In emulation, the AS signal is asserted in an access cycle of either the emulation memory or user memory.

NOTE

(B) Read/write (R/W) signal control

When emulation is broken, output of the read/write (R/W) signal to the target system can be controlled.

(1) Enable

When emulation is broken, the R/W signal is asserted to the target system in a repetitive pattern of idle loop command fetch of the emulation CPU. In emulation, the R/W signal is asserted in an access cycle of either the emulation memory (RO/RW mapping is set) or user memory (US mapping is set). (This is the default at initialization.)

(2) Disable

When emulation is broken, the R/W signal is not asserted to the target system (high level).

In emulation, the R/W signal is asserted in an access cycle of either the emulation memory or user memory.

(C) Data strobe (LDS/UDS) signal control in write cycle

In a write cycle of the emulation memory, output of the LDS/ UDS signal to the target system can be controlled.

(1) Enable

In a write cycle of the emulation memory (RO/RW mapping is set), the LDS/UDS signal is asserted to the target system.

(2) Disable

In a write cycle of the emulation memory (RO/RW mapping is set), the LDS/UDS signal is not asserted to the target system (high level). (This is the default at initialization.)

(D) Data strobe (LDS/UDS) signal control in read cycle

In a read cycle of the emulation memory, output of the data strobe (LDS and UDS) signals to the target system can be controlled.

(1) Enable

In a read cycle of the emulation memory (RO/RW mapping is set), the LDS/UDS signal is asserted to the target system.

(2) Disable

In a read cycle of the emulation memory (RO/RW mapping is set), the LDS/UDS signal is not asserted to the target system (high level).

NOTE

(E) Data bus signal control

In a read cycle of the emulation memory, output of the data bus (D0 to D15) signals to the target system can be controlled.

(1) Enable

In a read cycle of the emulation memory, data of the emulation memory is output to the data buses D0 to D15.

(2) Disable

In a read cycle of the emulation memory (RO/RW mapping is set), the data buses D0 to D15 are in high impedance.
(This is the default at initialization.)

*In a write cycle to emulation memory, the data bus D0 to D15 are always in high impedance.

(F) Bus error (BERR) signal control

When the emulation memory is accessed, the bus error (BERR) signal of the target system can be validated.

(1) Enable

The BERR signal of the target system is validated in an access cycle of either the user or emulation memory and BERR processing is emulated, and bus error processing is emulated.

(2) Disable

The BERR signal of the target system is validated in an access cycle of the user memory (US mapping is set) only.
The BERR signal is ignored in an access cycle of the emulation memory (RO/RW mapping is set).
(This is the default at initialization.)

(G) User wait (DTACK) signal control

When the emulation memory is accessed, the wait (DTACK) signal of the target system can be validated.

(1) Enable

The DTACK signal of the target system is validated in an access cycle of either the user or emulation memory and BERR processing is emulated.
In emulation, the access cycle is terminated by the DTACK or VPA signal of the target system.

NOTE

(2) Disable

The DTACK signal of the target system is validated in an access cycle of the user memory (US mapping is set) only.

The DTACK signal is ignored in an access cycle of the emulation memory (RO/RW mapping is set) and emulation starts without wait state.

(This is the default at initialization.)

(H) Double bus fault break signal control

If the processor enters double bus fault state in emulation, the break signal can be generated.

(1) Enable

If the processor enters double bus fault state in emulation, the double bus fault break signal is generated and monitor control is restored automatically.

(This is the default at initialization.)

(2) Disable

The processor stops in double bus fault state.

Initialize the processor by resetting the target system.

Monitor control can be restored by the STOp command.

(I) Auto wait signal control

The auto wait signal inputs the wait signal generated by the ERX wait signal generator in the DTACK pin of the processor.

(1) Enable

In emulation, a two-clock (or one to seven clock) pulse wait state is inserted in each machine cycle. In an access cycle of the user memory (US mapping is set) in this situation, a two clock pulse wait state is generated inside ERX even if the DTACK signal answers in S4 state. After the two-clock pulse wait state, if the DTACK signal of the target system is acknowledged, the machine cycle is terminated.

At this time, one to seven-clock pulse wait state times can be selected by the EMSEL control function (K).

(2) Disable

In emulation, a wait state is not generated inside ERX.

Ordinary real time emulation is performed.

(This is the default at initialization.)

NOTE

(J) Time – out break control

If memory access is executed in emulation state and the DTACK or VPA signal is not answered within 32K clock pulses (the clock value can be changed), the break signal can be generated. If memory access is executed in monitor state, a time – out error can also be generated.

At this time, a clock count value can be selected from 8 to 32K clock pulses by the EMSEL control function (L).

(1) Enable

If a time – out is detected, the break signal and a time – out error are generated.

(This is the default at initialization.)

(2) Disable

Time – out detection is not performed.

If the DTACK or VPA signal is not answered in emulation, monitor control can be restored by the STOp command.

If memory access is executed in monitor state, control can be returned by the ESC key.

(K) Control of wait time of the auto wait signal

Auto wait time of the auto wait signal can be selected from among one to seven clock pulses.

(1) One – clock pulse wait
(2) Two – clock pulse wait (default at initialization)
(3) Three – clock pulse wait
(4) Four – clock pulse wait
(5) Five – clock pulse wait
(6) Six – clock pulse wait
(7) Seven – clock pulse wait

NOTE

(L) Control of clock count of the time – out break signal
The clock count value of the time – out break signal can be selected from among 8 to 32K clock pulses.
(1) 8 clock pulses
(2) 128 clock pulses
(3) 2K (2048) clock pulses
(4) 4K (4096) clock pulses
(5) 8K (8192) clock pulses
(6) 16K (16,384) clock pulses
(7) 32K (32,768) clock pulses (default at initialization)

5.3.2 Control of use of the user wait (DTACK) and auto wait signals

Auto wait	User wait (DTACK)	Auto wait > User wait *1		Auto wait < User wait *2	
EMSEL (I)	EMSEL (G)	User memory access	Emulation memory access	User memory access	Emulation memory access
Disable	Disable	User's $\overline{\text{DTACK}}$	Non Wait	User's $\overline{\text{DTACK}}$	Non Wait
Disable	Enable	User's $\overline{\text{DTACK}}$	User's $\overline{\text{DTACK}}$	User's $\overline{\text{DTACK}}$	User's $\overline{\text{DTACK}}$
Enable	Disable	Auto Wait	Auto Wait	User's $\overline{\text{DTACK}}$	Auto Wait
Enable	Enable	Auto Wait	Auto Wait	User's $\overline{\text{DTACK}}$	User's $\overline{\text{DTACK}}$

Table 5 – 2

* 1 If auto wait time is longer than user wait (DTACK) time
* 2 If auto wait time is shorter than user wait (DTACK) time

NOTE

5.4 Clocks

The clock source can be selected from among the following options by using the **Clock** command :

(0) External clock (TTL level : max 20MHz)
The target system clock is used.
Use this clock when debugging clock - synchronized circuits.

(1) Internal clock (10MHz)

(2) Internal clock (5MHz)

(3) Internal clock (2.5MHz)
This clock is generated inside the ERX.
(If the internal clock is used, the target system will not receive the clock signal.)

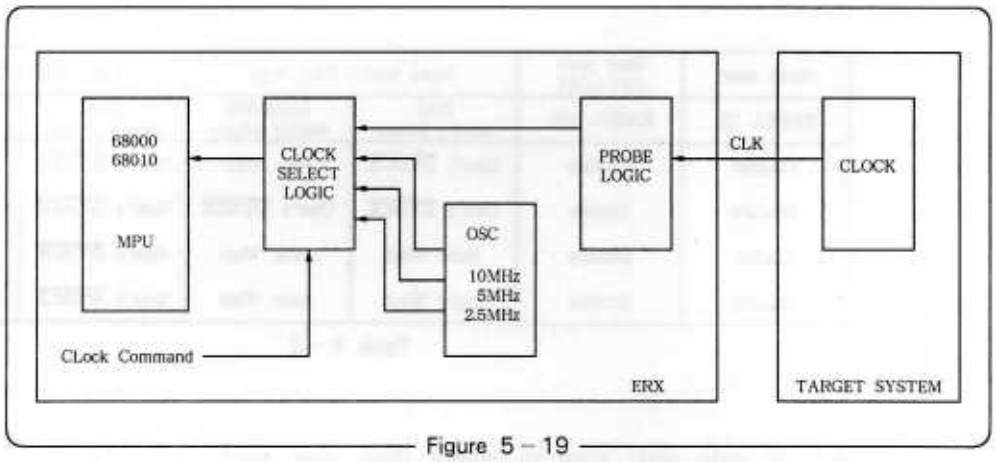


Figure 5 - 19

NOTE

5.5 $\overline{\text{RESET}}/\overline{\text{HALT}}$

While the emulation MPU is emulating the target system, if the $\overline{\text{RESET}}/\overline{\text{HALT}}$ signal is asserted by resetting the target system manually, a hardware reset is caused in the emulation MPU.
This is the same as an ordinary MPU.
While the emulation MPU is breaking emulation, a hardware reset is not caused in the emulation MPU by resetting the target system manually.
The emulation MPU register must be reset by the **Register RESet** command.
The **RESet** command resets the emulation MPU and the **ICERESet** command resets the entire ERX.
The $\overline{\text{RESET}}/\overline{\text{HALT}}$ signal from the target system is enabled/disabled by the **Pin** command.
Thus, emulation can be performed with the $\overline{\text{RESET}}/\overline{\text{HALT}}$ signal line disconnected.

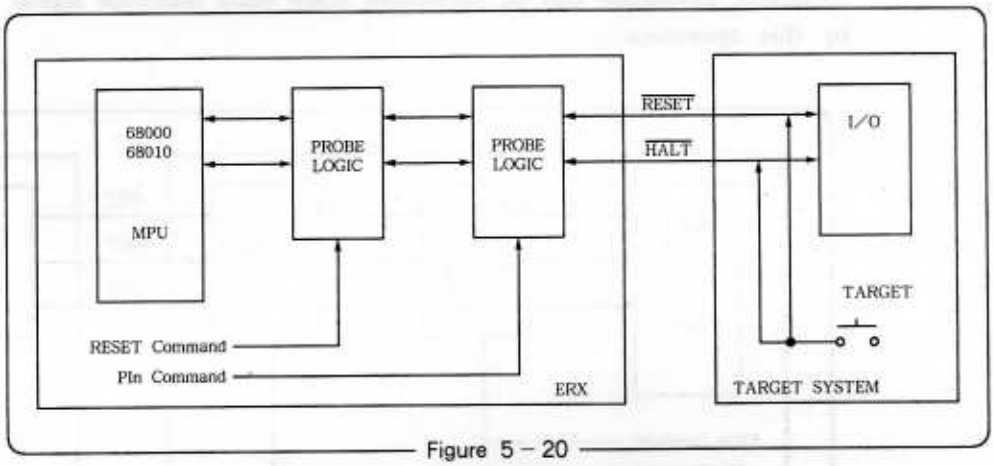


Figure 5 – 20

NOTE

5.6 $\overline{IPL0}$/$\overline{IPL1}$/$\overline{IPL2}$

The ERX generates a **level-7** interrupt in the emulation MPU when using the **STOp** command. This interrupt has priority over the **level-7** interrupt of the target system.

Like the **STOp** command, an emulation break that is set by the **Break** command also generates a **level-7** interrupt. When the emulation MPU is in emulation break state, the **level-7** interrupt signal of the target system is masked.

However, even when the emulation MPU is in emulation break state, **level-7 F.F** of the emulation MPU is set by the **level-7** edge trigger. So, a **level-7** interrupt generated during emulation break generates an interrupt sequence when execution is changed from monitoring to target emulation.

The **level-1** to **6** interrupt signals are masked when the emulation MPU is in emulation break state.

The **level-1** to **7** target interrupt signals can be controlled by using the **Pin** command. Emulation can be performed while each interrupt signal is disconnected by this operation.

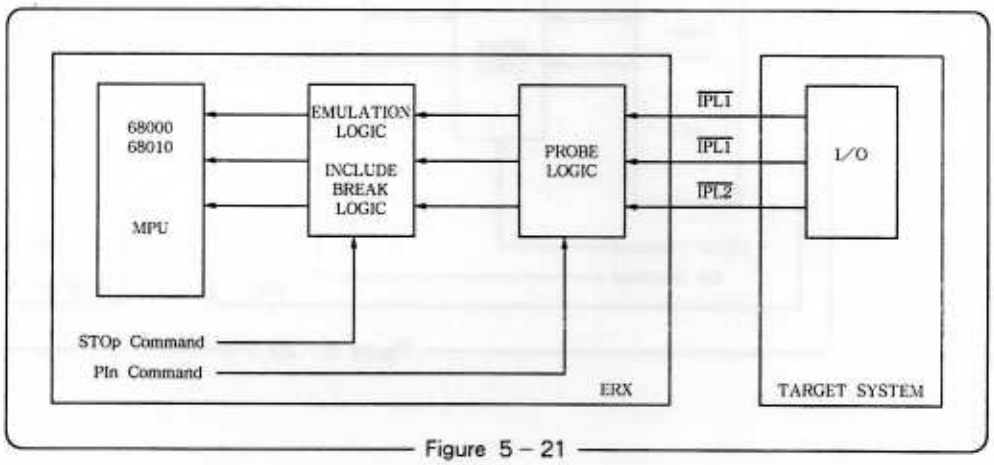


Figure 5-21

NOTE

5.7 $\overline{\text{BERR}}$

The $\overline{\text{BERR}}$ signal is accepted in the machine cycle when the user memory (US mapping setting) is accessed. At this time, bus error handling by combining the $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ signals can also be emulated.
Ordinarily, the $\overline{\text{BERR}}$ signal is ignored in the machine cycle when the emulation memory (RO, RW mapping setting) is accessed.
However, the $\overline{\text{BERR}}$ signal can be accepted by using the **EMSELect** command irregardless of map setting state.
The $\overline{\text{BERR}}$ signal from the target system can be controlled by using the **Pin** command. Emulation can be performed while the $\overline{\text{BERR}}$ signal is disconnected by this operation.

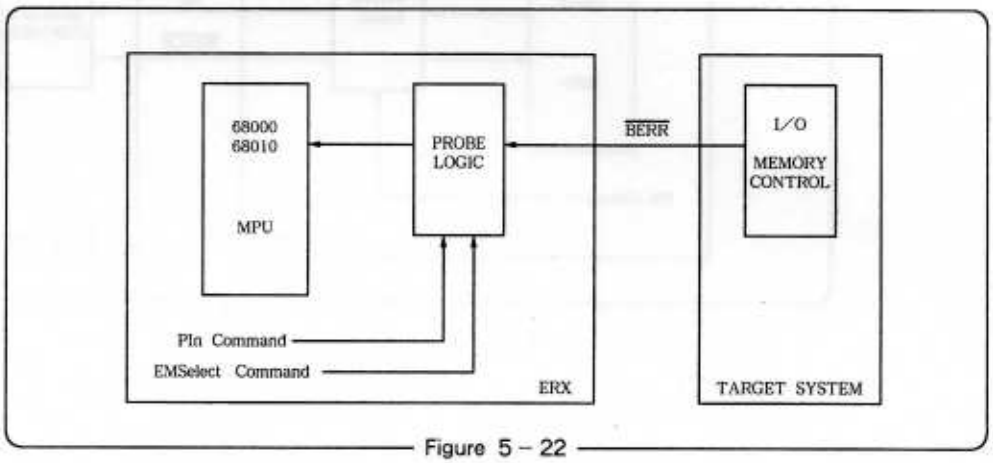


Figure 5 – 22

NOTE

5.8 $\overline{BR}$/$\overline{BG}$/$\overline{BGACK}$

The $\overline{BR}$ and $\overline{BGACK}$ signals are accepted at any time.
Therefore, even if an emulation break occurs while the target system is operating the DMA by the $\overline{BR}$ signal, the DMA can still operate properly.
The $\overline{BR}$ signal from the target system can be controlled by using the **Pin** command. Emulation can be performed while the $\overline{BR}$ signal is disconnected by this operation.

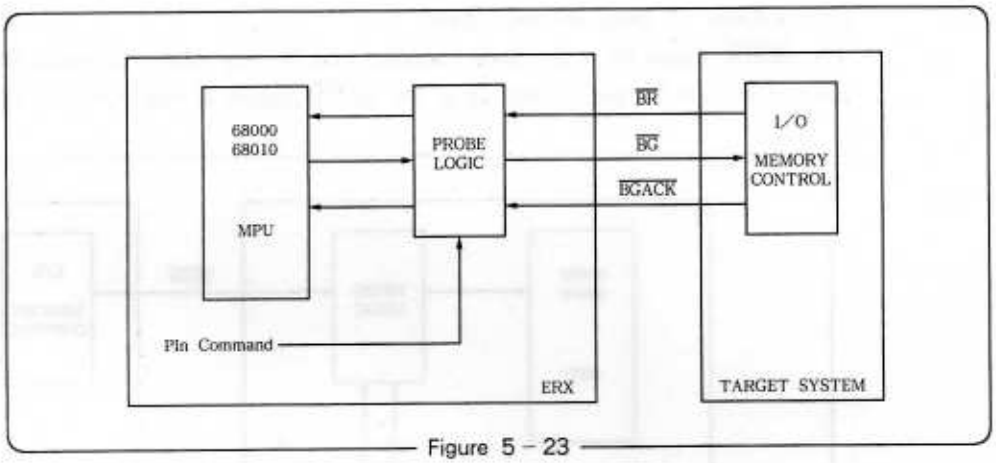


Figure 5 - 23

NOTE

5.9 $\overline{\text{DTACK}}$

When **G = enable** is set by the **EMSELect** command, the $\overline{\text{DTACK}}$ signal is accepted in each machine cycle.

When **G = enable** is set, the $\overline{\text{DTACK}}$ signal is delayed and a 2 – byte clock (or 1 to 7-byte clock) wait state can be genarated in each machine cycle in the ERX.

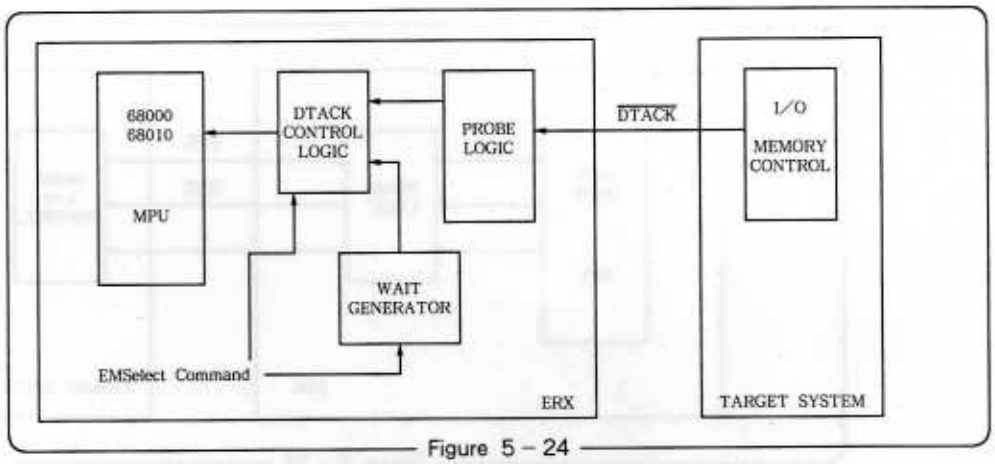


Figure 5 – 24

NOTE

5.10 $\overline{VPA}$/$\overline{VMA}$/E

In a machine cycle in which the user memory or I/O device (US mapping is set) is in access, the $\overline{VPA}$ signal is accepted. In a machine cycle in which the emulation memory (RO/RW mapping is set) is in access, the $\overline{VPA}$ signal is ignored.
The $\overline{VMA}$ signal, which is an answer to the $\overline{VPA}$ signal, does the same function as an ordinary MPU.
The E clock signal is always outputting clock pulses to the target system.

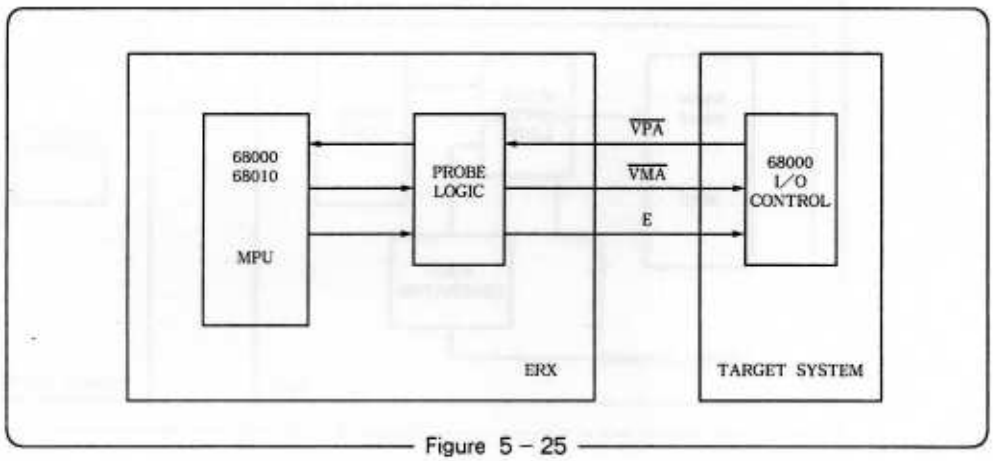


Figure 5 - 25

NOTE

5.11 Power Supply and Ground

The ERX is continuously monitoring the target system power supply.
When the probe is connected normally and the target system power is turned on, the target system and ERX GND are connected by a relay.
If the probe is inserted reversed or the target system power is turned off, the relay is turned off and target system and ERX GND are disconnected. This function avoids destruction of the probe buffers and target system due to improper connection of the probe.

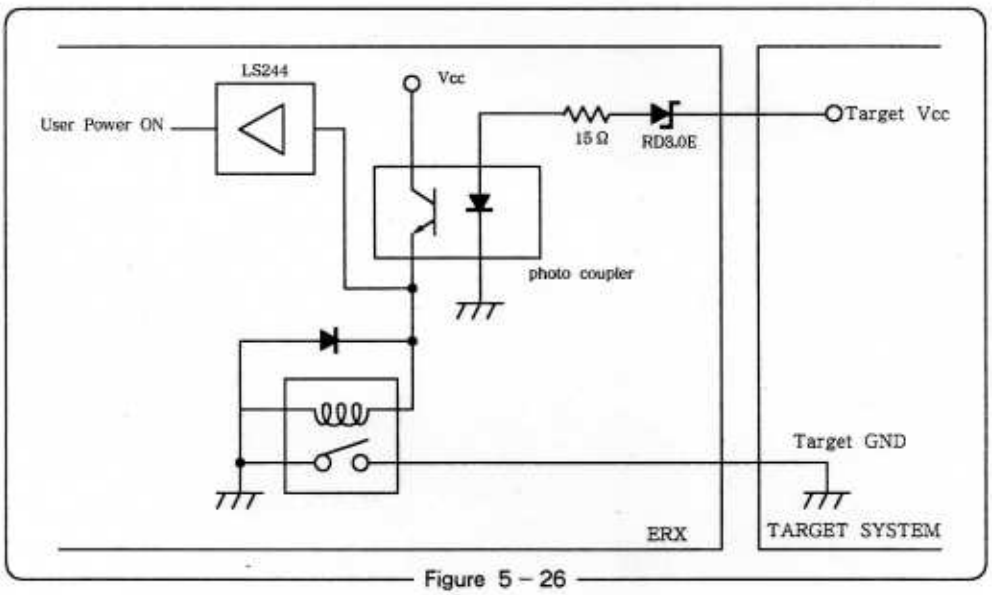


Figure 5 – 26

NOTE

NOTE

6. EMULATION MEMORY FUNCTION

6.1 Emulation Memory

The ERX has built-in 256K-bytes of high-speed static RAM as emulator memory for target programs. It is called emulation memory as compared to a user memory (target system memory).

This emulation memory can substitute for any memory in the 4G-byte memory area. This can be accomplished by allocating a 64K-byte emulation memory block to any memory location in the 4G-byte memory area by using the MAp command. The emulation memory is made up of a high-speed static RAM and can meet all low-speed to high-speed systems requirements.

When a target system views the emulation memory, the memory looks different from ordinary memory because it is built in the ERX probe.

The ERX address, data, and control bus pins must be in high impedance when buses are grounded. Because of the emulation characteristics, DMA transfer between target system and emulation memories is disabled but DMA transfer between the target system memory space and emulation memory is not a problem.

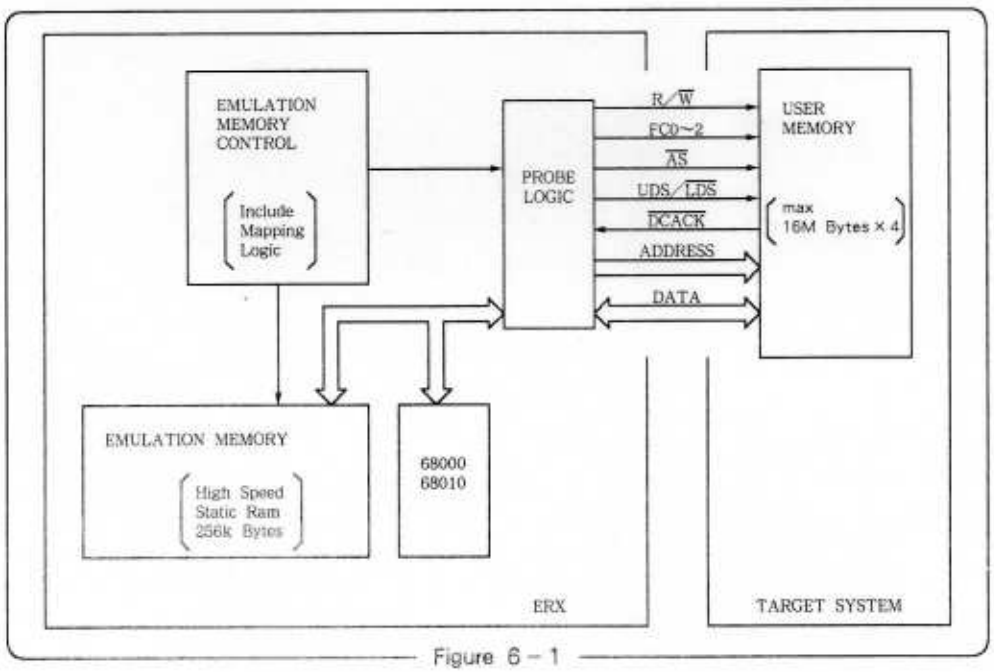


Figure 6-1

NOTE

6.2 User Memory

The ERX can assign a 4G-byte memory space to a user memory in 64K-byte memory space units.

The timing when the ERX writes in the target system memory is equal to the processor access timing. The timing when the ERX reads from the target system memory is a little shorter than the processor access timing. For details, see the AC charcteristics in Appendix.

NOTE

6.3 Mapping

The ERX selects one from among four kinds of memories in 64K – byte units. The following figure shows a block diagram.

(1) Read/write memory (RW)
(2) Read – only memory (RO)
(3) User memory (US)
(4) Nonexistence memory(NO)

The four kinds of memories are explained below.

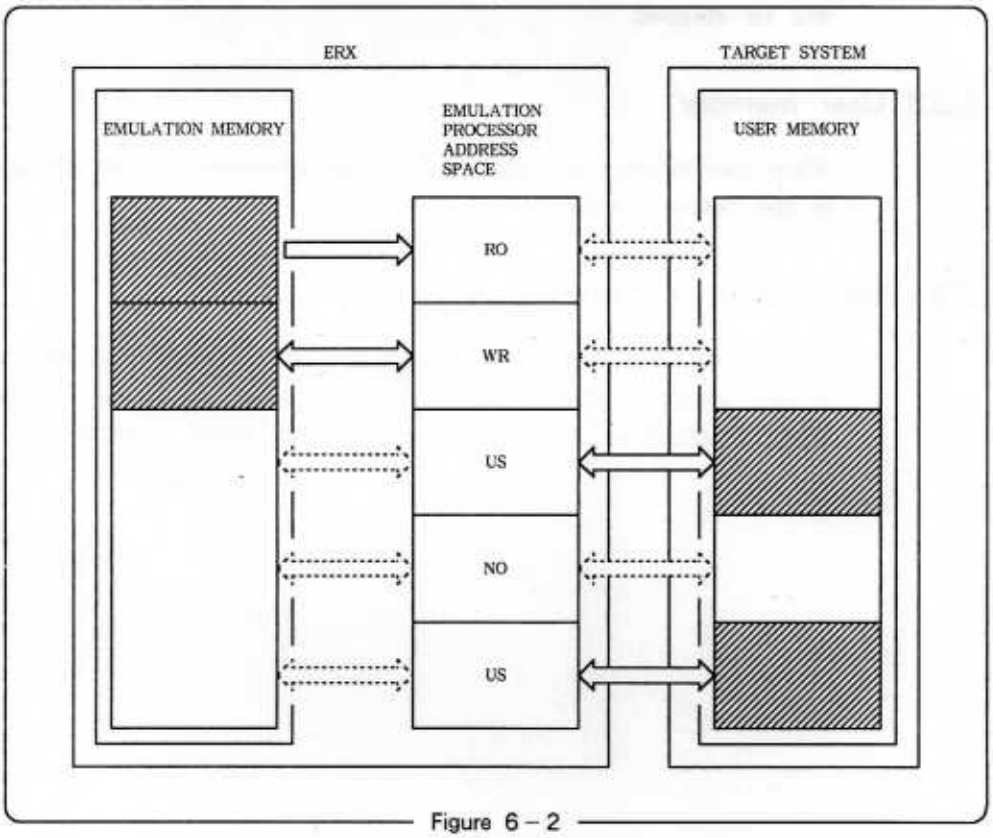


Figure 6 – 2

NOTE

6.3.1 Read/Write memory

If a read/write memory is specified, the target processor uses the emulation memory in the ERX as an accessable memory space instead of the RAM memory of the target system.

6.3.2 Read-only memory

When a read-only memory is specified, the target processor uses the emulation memory in the ERX as an accessable memory space, instead of the ROM device of the target system. The target processor cannot rewrite the memory space defined as a read-only memory. When this area of memory is written to, emulation will be stopped.

6.3.3 User memory

When user memory is specified, the target processor accesses the memory mounted in the target system.

6.3.4 Nonexistent memory

When nonexistent memory is specified, it means that no memory is mounted in the specified memory space. So, if the target processor accesses the memory space specified as nonexistent memory, emulation is stopped.

NOTE

7. REALTIME TRACE FUNCTION

The function that traces the execution contents in real time during emulation is called the real time trace function.

The real time trace function consists of two control functions such as the trace and trigger control functions.

The trace control function traces the execution contents in each machine cycle during emulation.

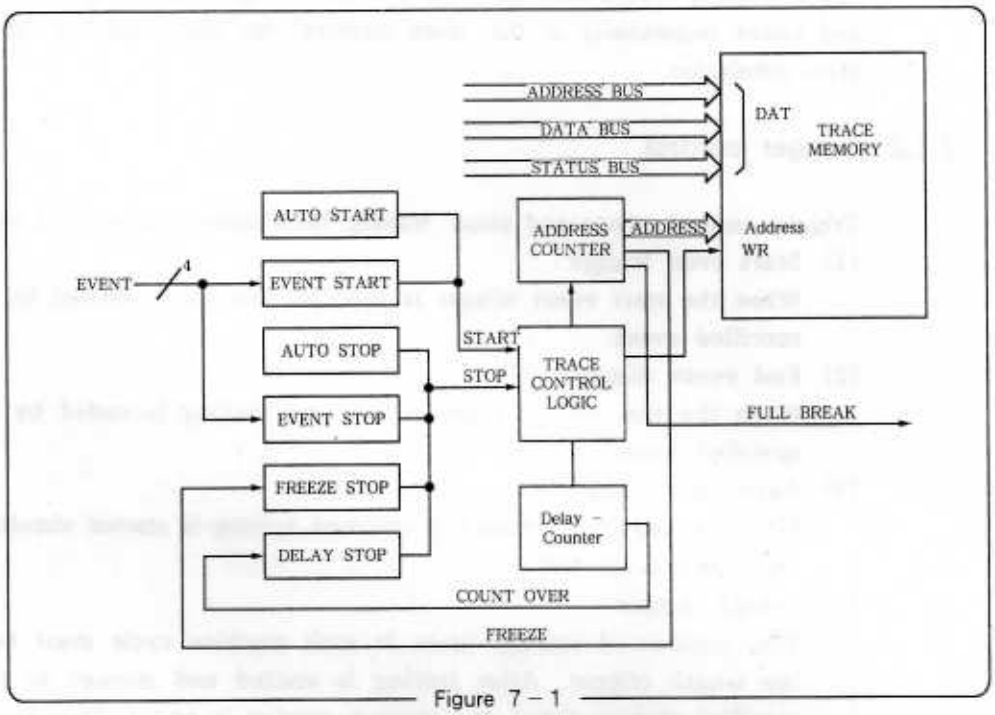


Figure 7 – 1

NOTE

7.1 Real Time Trace Control

7.1.1 Trace control

Trace control traces address, data, and status information in each machine cycle and stores sequentially in the trace memory. So, trace contents can be confirmed after emulation.

7.1.2 Trigger control

Trigger control starts and stops tracing. Each control function is explained below.

(1) Start even trigger
When the start event trigger is specified, tracing is started by output of the specified event.

(2) End event trigger
When the end event trigger is specified, tracing is ended by output of the specified event.

(3) Auto start trigger
When the auto start trigger is specified, tracing is started simultaneously when emulation is started.

(4) Length trigger
The number of storage times in each machine cycle must be specified in the length trigger. After tracing is started and storage to the number of specified storage times is executed, tracing is ended. This function is valid only when the start event trigger is specified.

(5) Multi – trigger
When the multi – trigger is specified, tracing is started each time the start event trigger is generated. If this function is omitted, tracing is started by the first start event trigger generated after emulation and the second and subsequent start event triggers are ignored.

NOTE

(6) Freeze trigger

When the freeze trigger is specified, storage is stopped when the trace memory is full after tracing is started. If this function is omitted, the trace memory overflows after tracing is started. The trace memory, however, rotates and updates old contents.

7.1.3 History full break control

Full break control generates the full break signal when the trace memory is full. So, when the trace memory is full in emulation, emulation can be broken.

NOTE

7.2 How to Use The Real Time Trace Function

The real time trace function is realized by each trigger control function explained in Section 5.1. Varieties of trace modes can be created by combining these trigger control functions.
Some basic modes needed for incircuit emulation are distributed in the standard macro library. For details on how to handle the standard macro library, see Appendix A.
This Section describes the eight modes provided in standard macro library.

(1) End monitor mode (EM)
(2) Begin monitor mode (BM)
(3) End event mode (EE)
(4) Begin event mode (BE)
(5) Center event mode (CE)
(6) Multiple event mode (ME)
(7) Inner event mode (IE)
(8) Outer even mode (OE)

Table 7 – 1 lists the parameters of the History command for realizing these eight modes.

	Start Event	End Event	Auto Start	Length	Multi	Freese
End Monitor	OFF	OFF	ON	OFF	OFF	OFF
Begin Monitor	OFF	OFF	ON	OFF	OFF	ON
Begin Event	*	OFF	OFF	*	OFF	ON
Center Event	*	OFF	ON	2047	OFF	OFF
End Event	OFF	*	ON	OFF	OFF	OFF
Multiple Event	*	OFF	OFF	*	ON	*
Inner Event	*	*	*	OFF	OFF	*
Outor Event	*	*	*	OFF	OFF	*

Table 7 – 1

* The specified value can be changed.

NOTE

7.2.1 End monitor mode

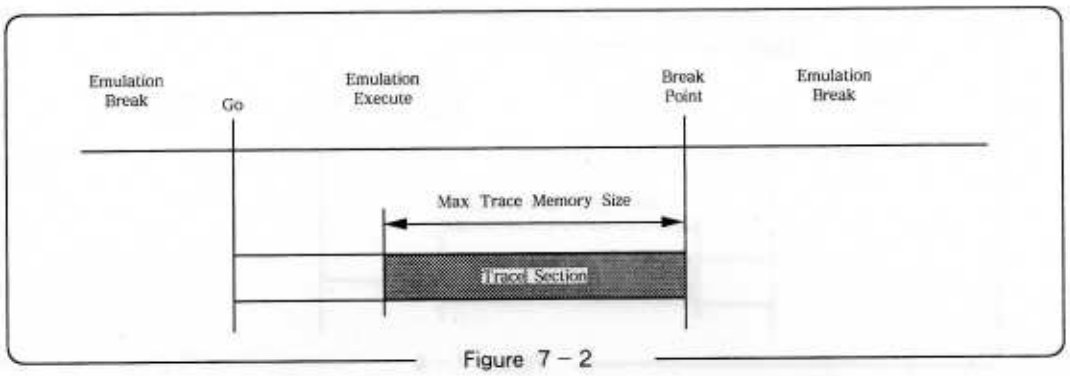


Figure 7 - 2

When emulation is started by the **Go** command, The trace buffer is started. At a breakpoint or when an emulation break is generated by the **STOP** command, the trace is stopped. The trace range ends with the cycle immediately before an emulation break is generated.

7.2.2 Begin monitor mode

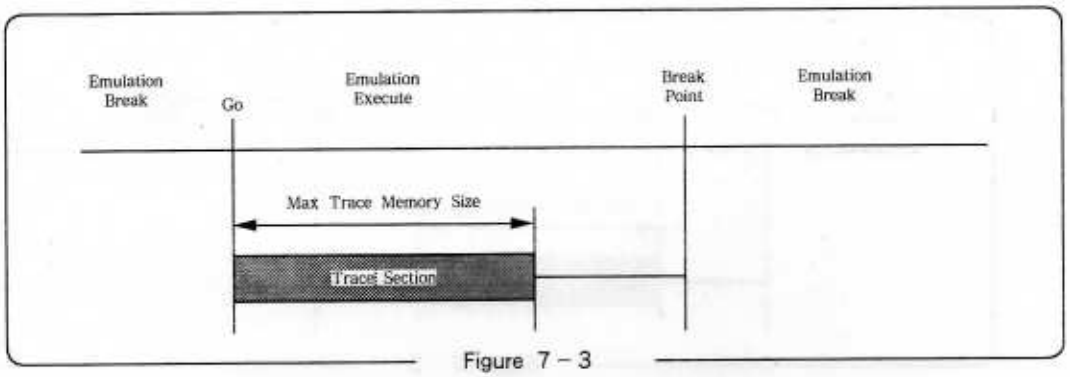


Figure 7 - 3

When emulation is started by the **Go** command, trace is started. When the max trace memory size is traced, the trace is stopped automatically.
The trace range begins immediately after emulation is started by the **Go** command and does not exceed the maximum trace memory cycle.

NOTE

7.2.3 End event mode

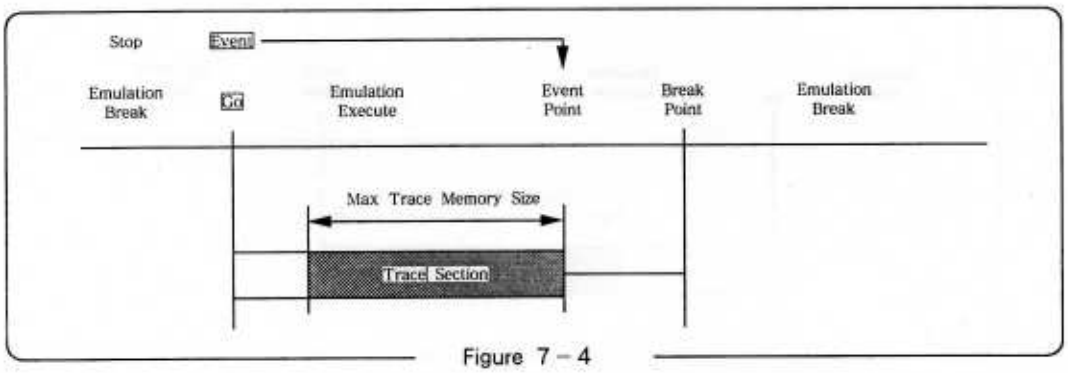


Figure 7 – 4

When emulation is started by the Go command, trace is started.
When the specified event point is passed, the trace is stopped automatically.
The trace range ends with the cycle immediately before the event point is passed and does not exceed the max trace memory cycle.

7.2.4 Begin event mode

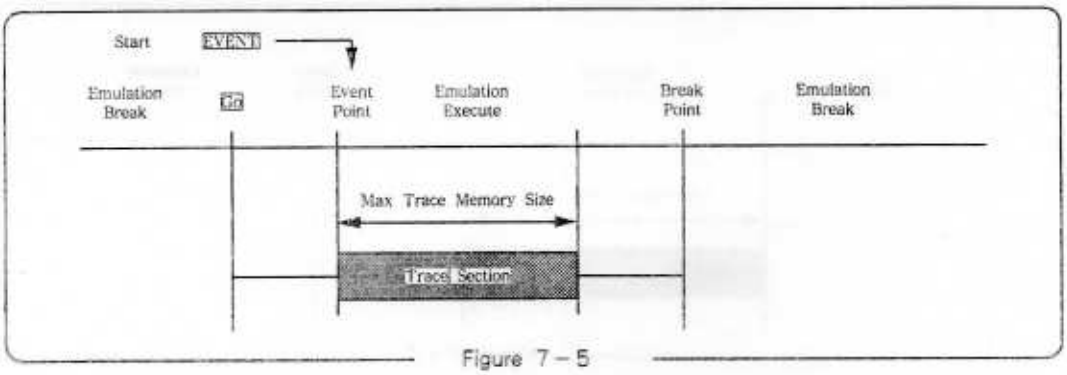


Figure 7 – 5

When the event point specifying start of emulation by the Go command is passed, trace is started. When the maximum trace memory size is stored, the trace is stopped automatically. The trace range begins immediately after the specified event point is passed and does not exceed the max trace memory cycle.

NOTE

7.2.5 Center event mode

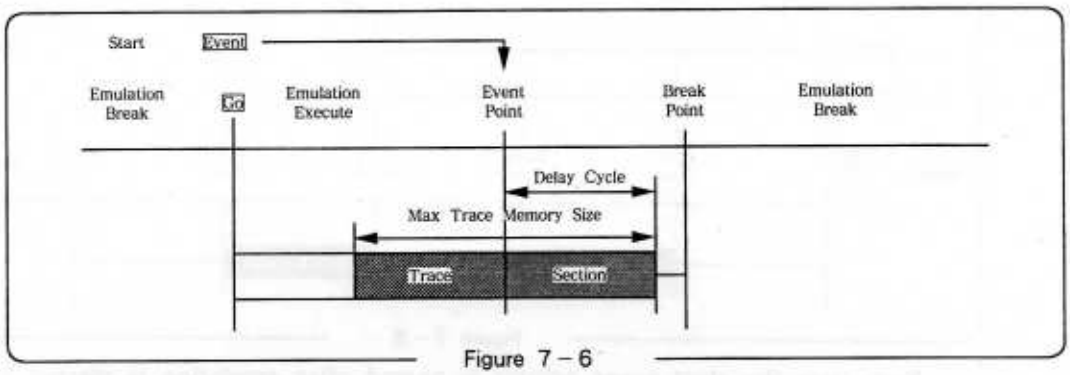


Figure 7 – 6

When emulation is started by the Go command, trace is started. When the max trace memory size is traced, the trace is stopped automatically.
The trace range begins immediately after the specified event point is passed and does not exceed the maximum trace memory cycle.

7.2.6 Multiple event mode

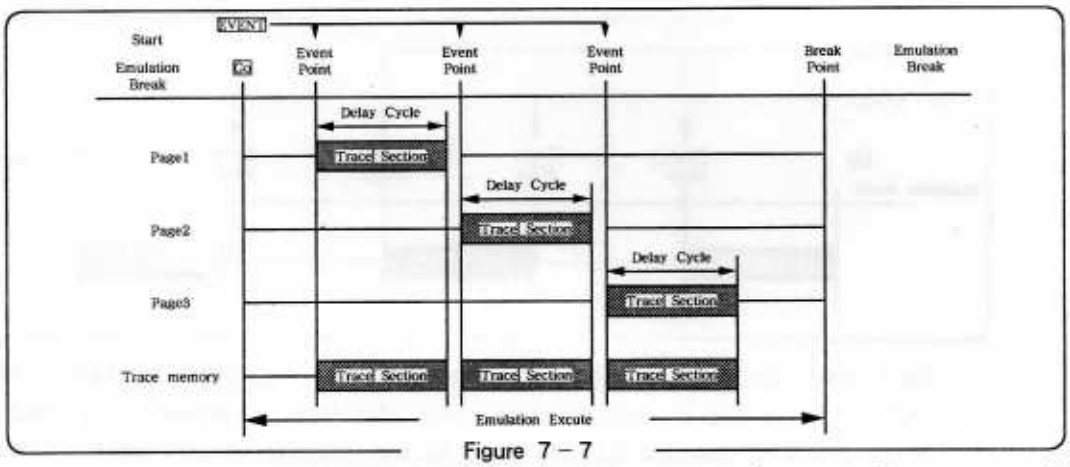


Figure 7 – 7

When traceing is performed by using the same event point more than once, multiple event mode is used. Each time the event point is passed after emulation is started by the Go command, trace is started. The tracing range begins immediately after the event point is passed and does not exceed the delay cycle.

NOTE

7.2.7 Inner event mode

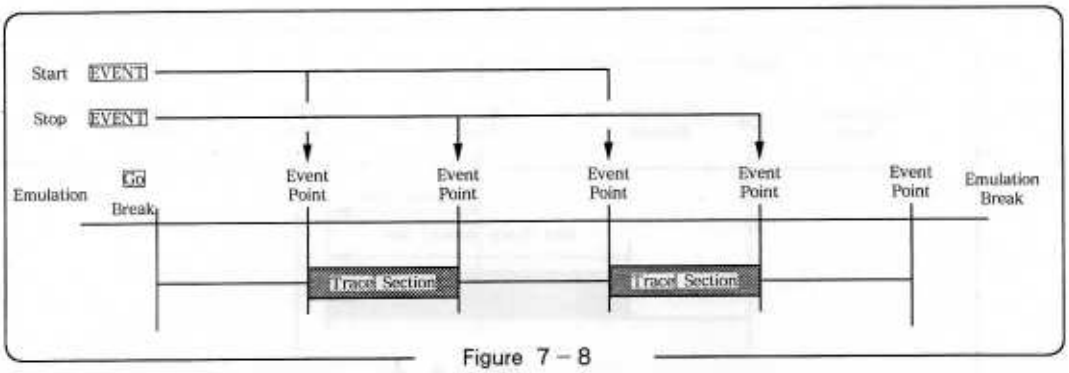


Figure 7 – 8

Each time the start event pointer is passed after emulation is started by the Go command, tracing is started. Each time the stop event pointer is passed, the tracing buffer is stopped. The operation range of the emulation processor seen by the real – time trace can be limited by using this mode. The mode is useful when attention is concentrated on a specific program module and program operation is to be analyzed.

7.2.8 Outer event mode

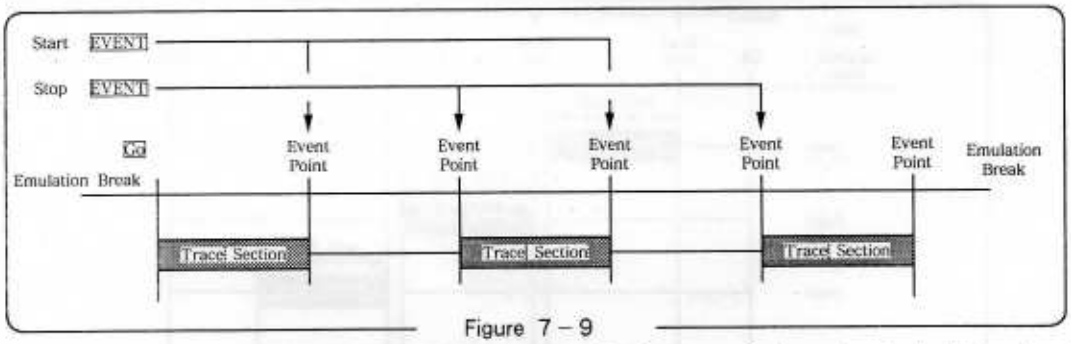


Figure 7 – 9

Each time the start event point is passed after emulation is started by the Go command, tracing is stopped. Each time the stop eventpoint is passed, traceing is started. The function of this mode is the opposite of the function of the inner event mode. The operation range of the emulation processor that need not be seen by real – time trace can be excluded by the outer event mode.
When the interrupt processing module is excluded and application module program operation is to be analyzed, this mode is also useful.

NOTE

8. BREAK FUNCTION

The function that forces the processor to stop during emulation is called the break function. A break occurs when one of the following eight conditions is met. The break function is controlled by NMI (level 7) signal.

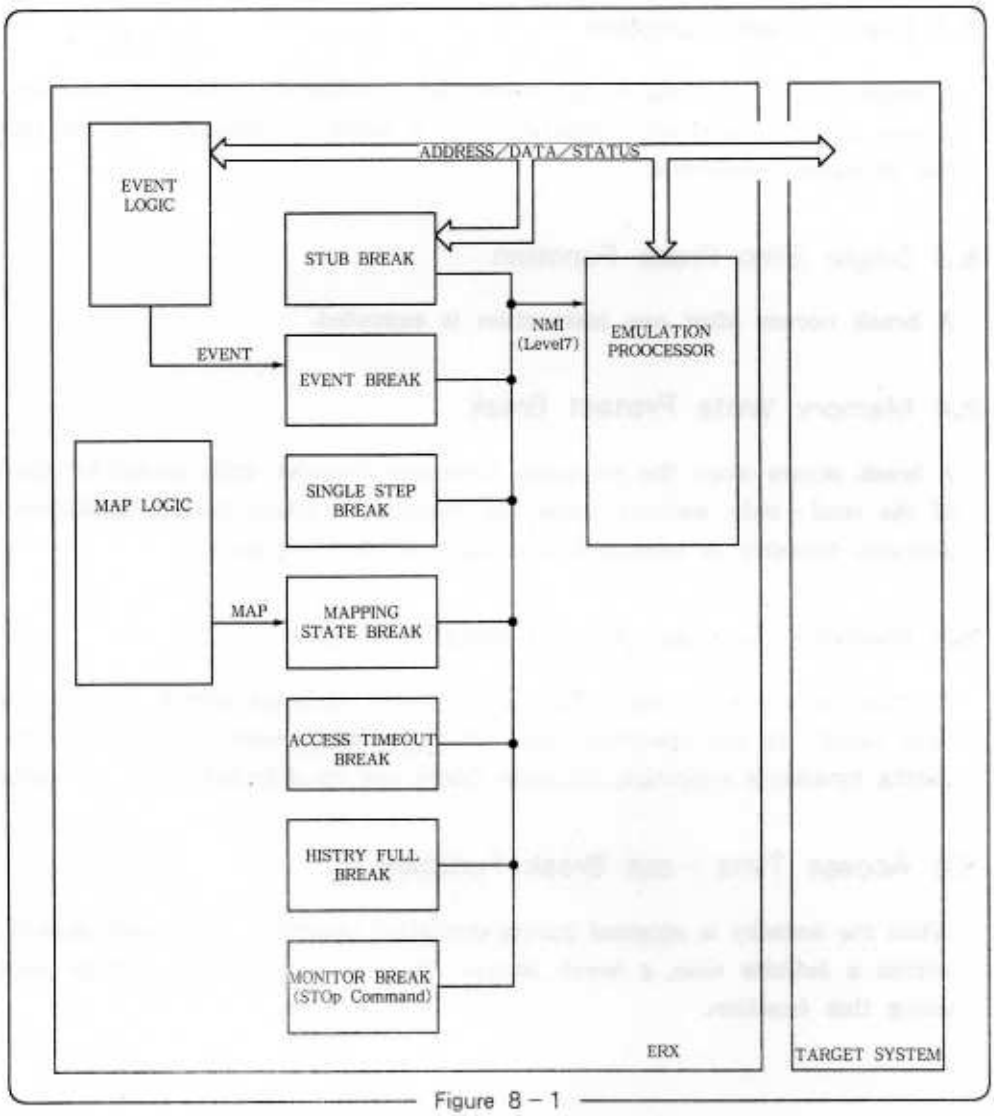


Figure 8 – 1

NOTE

8.1 Monitor Break Function

A break occurs unconditionally according to the operator specification.

8.2 Event Break Function

A break occurs according to the eventpoint specification, when the machine cycle condition is met by the hardware comparator. So, a break is generated in the machine cycle of the emulation processor.

8.3 Single Step Break Function

A break occurs after one instruction is executed.

8.4 Memory Write Protect Break

A break occurs when the emulation processor executes write access for the specified space of the read-only memory using the mapping function during emulation execution. The accessed memory is protected, and may not be written to.

8.5 Memory Guarded Access Break Function

A break occurs when the emulation processor executes access (read, write, or operation code fetch) to the specified space of nonexistent memory using the mapping function during emulation execution. Program errors can be detected easily by using this function.

9.6 Access Time-out Break Function

When the memory is accessed during emulation execution, if acknowledgment is not received within a definite time, a break ocuurs. Memory access errors can be detected easily by using this function.

NOTE

8.7 Double Bus Fault Break Function

When the processor enocounters a double bus fault during emulation, a break occurs and control returns to the monitor.
When the power is turned off, the processor in double bus fault state is stopped.
The processor can be initialized when it is reset by the target system.
Control can be returned to the monitor by using the STOp command.

8.8 History Full – Break Function

A break occurs when in the trace memory becomes full during emulation execution.

8.9 Stub Function

A break occurs when an instruction specifying a software break point is fetched.
This is a before – execution break because the instruction at the break point is not executed and emulation is halted.
Actually, the break function is initiated by an exception generated when an illegal instruction is executed, and control returns to the monitor.
The software break function can be specified only for accessible (read/write) memory when an illegal instruction must be inserted in the program.
When the function is specified for the read – only memory, data is transferred from the read – only memory of the target system to the emulation memory and after that the software break function is enabled using emulation memory (read – only memory).

8.10 On – break Function

The program operation can be changed freely by intervening software.
Software can be intervened by combining operation commands.
Actually, emulation is interrupted by a hardware break generated by the event function and the specified command, macro, or batch job is executed.
So, the on – break function is not a debug function in real – time mode.

NOTE

NOTE

9. EVENT FUNCTION

The function that supplies a trigger signal to each of the ERX emulation execution, analysis, and measurement functions in real time by using the machine cycle comparator is called the event function.

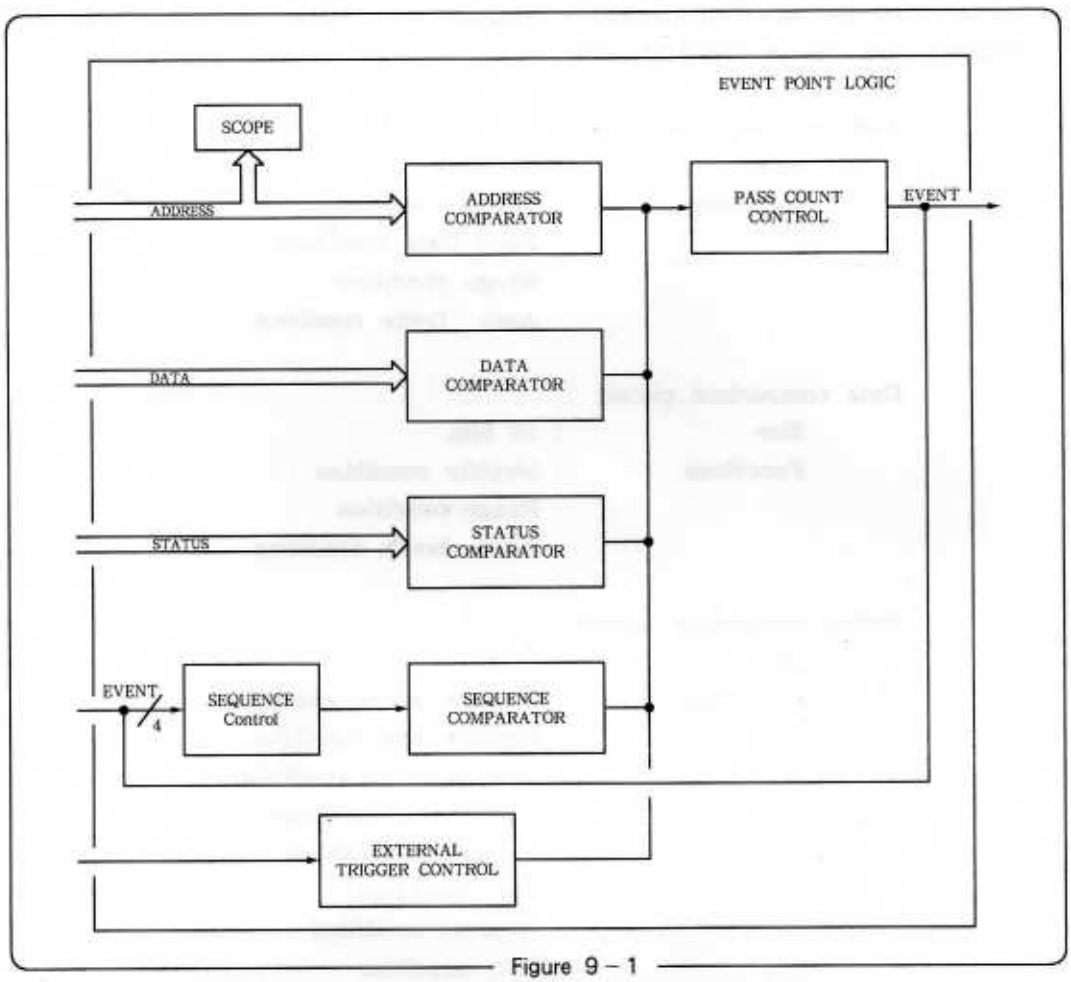


Figure 9 – 1

NOTE

9.1 Event Points

The circuit that monitors emulation state in real time and that outputs an event trigger signal when the specified conditions (address, data, status, bus count sequential, external trigger) are met is called an event point.

Address comparison circuit
- Size : 23 bits
- Functions : Identity condition
  Don't Care condition
  Range condition
  Auto – break condition

Data comparison circuit
- Size : 16 bits
- Functions : Identity condition
  Range condition
  Auto – break condition

Status comparison circuit
- Size : 6 bit
- Functions : Memory access condition
  Memory read condition
  Memory write condition
  Interrupt acknowlidge
  Supervisor condition
  User condition
  Program condition
  Data condition

Passcount function
- Size : 16 bit

NOTE

9.2 Sequential Function

The function that specifies the sequence of the specified event points in more than two channels is called the sequential function. This function cannot control branch. Therefore, sequential function can be specified in serial mode.

9.3 External Input Trigger Function

The function that inputs an asynchronous signal as a condition from an event point is called the external input trigger function. The external signal that has been input is synchronize with the address, data, and status condition in the event point circuit.
An external input trigger signal can be selected from among the following four types :

(1) Level signal : High level active
(2) Level signal : Low level active
(3) Pulse signal : Positive edge
(4) Pulse signal : Negative edge

Actually, an asynchronous signal is input from the external probe.
For details, see Section 4.4

9.4 External Output Trigger Function

This function outputs the identity signal from the event point to the outside.
Actually, an asynchronous signal is input from the external probe.
For details, see Section 4.4

9.5 Scope Function

The function that monitors in real time the target processor program counter location while executing program memory during emulation is called the scope function.

NOTE

NOTE

10. COVERAGE FUNCTION

The function that measures in real time the target program test and that represents the program reliability in numeric is called the coverage function.

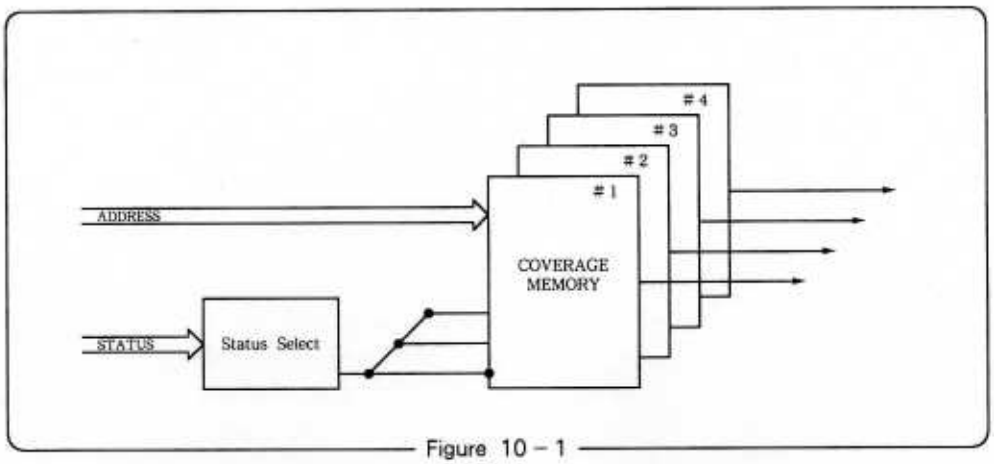


Figure 10 - 1

Actually, when a certain address of the emulation processor is accessed, the function that sets the coverage memory outputs a test result. So, before the coverage function is used, coverage memories must be allocated to the memory spaces to be measured and the coverage memories must be all accessed. After that, when emulation is started, the executed programs and accessed memories are plotted in the coverage memories. When emulation is broken, the distribution of the programs and memory accessed during emulation can be measured. Each measured value is displayed in percentage.

NOTE

NOTE

11. PERFORMANCE FUNCTION

The function that measures execution time between two event points, or that measures the number of execution times at one event point, is called the performance function.

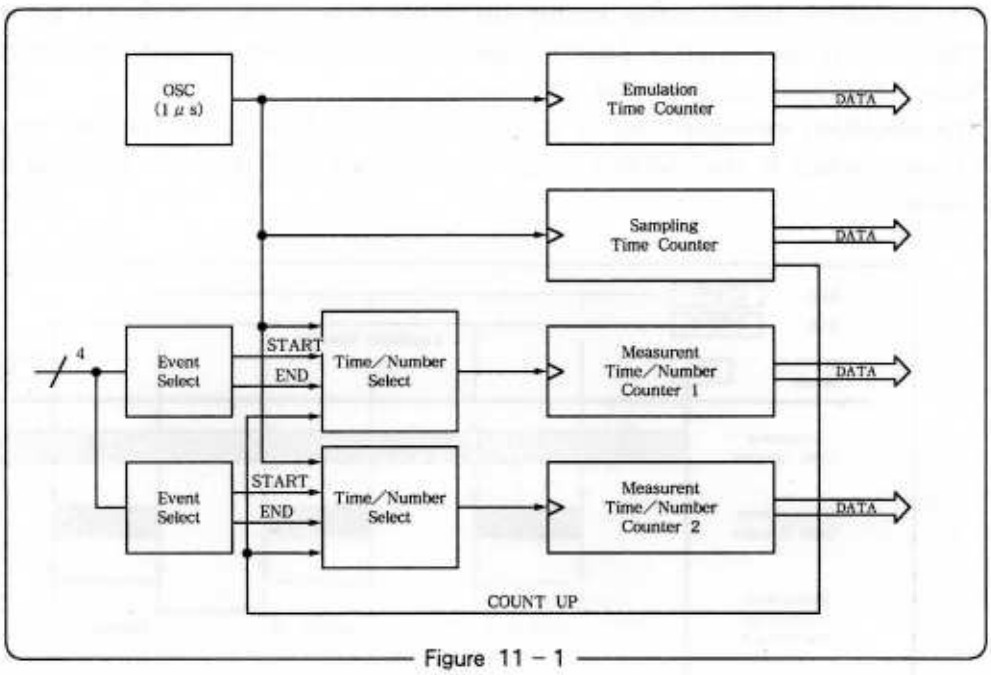


Figure 11 – 1

NOTE

The performance function is made up of an emulation time counter, a measure time counter, and a measurement number counter.
The emulation time counter counts the 1-μs basic clock to measure the execution time. The measurement number counter measures the number of event points that have been passed between the start and end events.
The emulation execution time is the total execution time in the specified range, the number of event points in the specified range, and processor load ratio are displayed as measurement results.

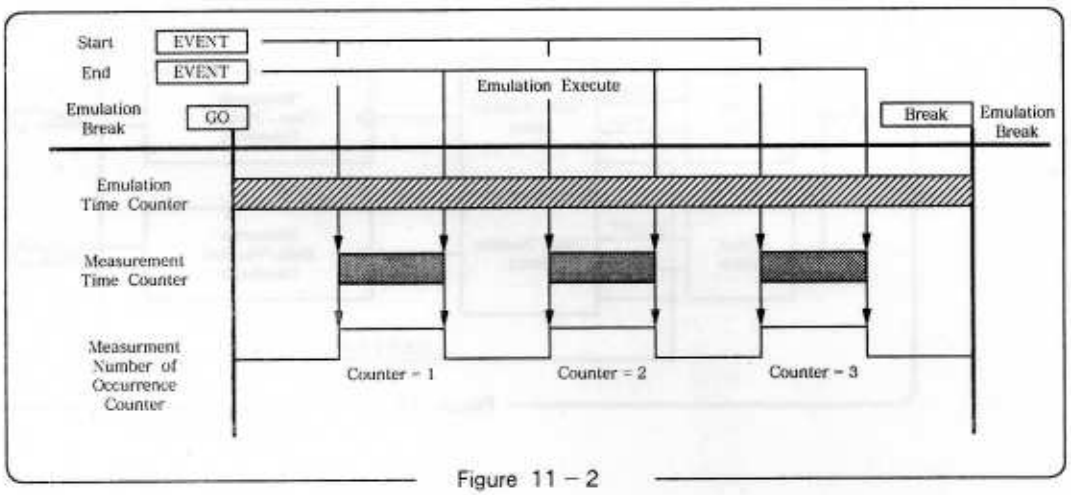


Figure 11 - 2

NOTE

APPENDIX

ERX318 for 68000 • 68010

APPENDIX

Rev. E1.01

ERX318 for 68000・68010 APPENDIX

CONTENTS

A. STANDARD MACRO LIBRARY

A.1 Outline

The standard macro library is a macro file provided by **Zax Corporation.**
When the **ERX** is started, the standard macro library is loaded automatically.
The macro commands, can be executed using the **ERX** commands by using this library named.

File name : **ERX68K.MAC**

The configuration and procedure of the standard macro library are the same as those of a user – created macro file.
The standard macro library loads the macros when the **ERX** is started. The user macro file loads the macros when the **MLoad** command is executed.

NOTE

MH

Function

The MH command displays the command help messages of the standard macro library.

Input format

```
> MH < CR >
> MH  SETUP < CR >
```

Input example

```
ERX>MH
    Help message
        :
        :
ERX>
```

NOTE

DIR

Function

The DIR command displays a list of files on the disk.

Input format

```
> DIR    [path_name]    < CR >
```

path_name : Directory path name. Specify the file name.

Input example

```
ERX>dir *.bat

Volume in drive C has no label
Directory of C: \USER\

SAMPLE  BAT  174   10.10.86  12:00p
DEMO    BAT  2648  12.10.86  12:00p
 2 File(s)   2685418 bytes free

ERX>
```

NOTE

The MS－DOS wild card can be assigned to the file name.

TYPE

Function

The **TYPE** command displays a text file to the screen.

Input format

```
> TYPE File_name  < CR >
```

file_name : Text file name

Input example

```
ERX> type  sample.doc
This is a sample document
          :
          :
ERX>
```

NOTE

MEDIT

Function

The **MEDIT** command modifies the emulation or target memory using the screen editor.

Input format

```
> MEDIT    beg_addr,end_addr   < CR >
```

beg_addr : Specify the modification begin memory address.
end_addr : Specify the modification end memory address.

Input example

```
ERX>MEDIT  1000, 13FF
        :
    Screen editor
        :
ERX>
```

NOTE

SETUP

Function

The **SETUP** command initializes the following when emulation is started :

1. Clock selection
2. Memory mapping
3. Pin control
4. Object program downloading
5. Other setup command settings

The **SETUP** command has two functions. One function specifies the initial condition in an interactive format. The other function initializes the emulator with specified conditions.

When initial conditions are specified in interactive format, a file (CONFIG.CNF) is created automatically.

This file can be used for automatic initialization.

Input format

```
>SETUP  <CR>

Setup by CONFIG.CNF File (Yes or <cr>/No) = N <CR>

Select Clock (0 - 3)  = clock_mode  <CR>

Memory Start Address or  <CR>  = start_address  <CR>
Memory End Address                = end_address  <CR>
Mapping Type (RW/RO/NO/US)  =  map_type  <CR>
Pin Control  (Di/En)  =  DI | EN  <CR>
Load Object File  =  object_file_name <CR>
Key in command ro  <CR>  = command <CR>

>SETUP  <CR>
Setup by CONFIG.CNF File (Yes or <CR>/No)  = Y <CR>
```

auto set

NOTE

Setup by CONFIG.CNF	:	Specifies initialization. Y Initializes by using the CONFIG.CNF file. N Initializes by key input. by this setting, the initial conditions are stored in the CONFIG.CNF file.
clock_mode	:	Selects a target CPU clock. 0 | 1 | 2 | 3
start_address	:	Specifies the mapping starting address. If only <CR> is entered, control is moved from mapping specification to the next specification.
end_address	:	Specifies the mapping end address.
map_type	:	Specifies the mapping type. RO | RW | US | NO
pin_control	:	Controls the mask of the target CPU signal. DI | EN
object_file_name	:	Specifies the object file to be downloaded.
command	:	Specifies other initialization commands.

Input example

```
ERX>setup
Setup by CONFIG.CNF File (Yes or <CR> / No) = n
==== Clock Setup ====
0  External Clock (TTL)
1  Internal Clock (10MHz)
2  Internal Clock (5MHz)
3  Internal Clock (2.5MHz)
Select Clock (0-3)   = 2
```

NOTE

```
==== Map Setup ====
Memory Start Address or <CR>     = 1000
Memory End Address               = OFFFF
Mapping  Type (RW/RO/NO/US)      = us
Memory Start Address or <CR>     = 0
Memory End Address               = 0fff
Mapping  Type (RW/RO/NO/US)      = ro
Memory Start Address or <CR>     =
==== Pin Setup ====
Pin Control (Di/En)    = di
==== Object Setup ====
Load Object File Name  = sample.abs

ERX>setup
Setup by CONFIG.CNF File (Yes or <cr> / No) = y
==== Clock Setup ====
Internal Clock (4MHz)
==== Map Setup ====
 map
0000 - 0FFF = RO
1000 - FFFF = US

==== Pin Setup ====
 Pin
RESET*   = High (Diaable)
NMI*     = High (Disable)
IWT*     = High (Disable)
BUSRQ*   = High (Disable)
WAIT*    = High (Disable)

==== Object Setup ====
Load Object File Name  = sample.abs
ERX>
```

NOTE

EM

Function

The EM command sets history trigger mode as end monitor mode.

Input format

```
> EM   < CR >
```

Input example

```
ERX>EM
ERX>
```

NOTE

BM

Function

The BM command sets history trigger mode as begin monitor mode.

Input format

```
> BM   < CR >
```

Input example

```
ERX>BM
ERX>
```

NOTE

BE

Function

The **BE** command sets history trigger mode as begin event mode.

Input format

```
> BE    event_symbol   < CR >
```

event_symbol : Specify the defined event symbol name.

Input example

```
ERX>BE MAIN.LOOP1
ERX>
```

NOTE

CE

Function

The **CE** command sets history trigger mode as center event mode.

Input format

```
> CE    event_symbol    < CR >
```

event_symbol : Specify the defined event symbol name.

Input example

```
ERX>CE MAIN.LOOP1
ERX>
```

NOTE

EE

Function

The **EE** command sets history trigger mode as end event mode.

Input format

```
> EE    event_symbol    < CR >
```

event_symbol : Specify the defined event symbol name.

Input example

```
ERX>EE MAIN.LOOP1
ERX>
```

NOTE

ME

Function

The ME function sets history trigger mode as multiple event mode.

Input format

```
> ME    event_symbol,length   < CR >
```

event_symbol : Specify the defined event symbol name.
length : Specify the amount of one – time storage.

Input example

```
ERX>ME MAIN, LOOP1, 20
ERX>
```

NOTE

IE

Function

The IE command sets history trigger mode as inner event mode.

Input format

```
> IE    start_event, end_event   < CR >
```

start_event : Specify the symbol name of the event that becomes the trace start trigger.

end_event : Specify the symbol name of the event that becomes the trace end trigger.

Input example

```
ERX>IE MAIN.START, MAIN.END
ERX>
```

NOTE

OE

Function

The **OE** command sets history trigger mode as outer event mode.

Input format

```
> OE    start_event, end_event   < CR >
```

start_event : Specify the symbol name of the event that becomes the trace interrupt trigger.

end_event : Specify the symbol name of the event that becomes the trace restart trigger.

Input example

```
ERX>OE SUB. START, SUB. RETURN
ERX>
```

NOTE

AIE

Function

The AIE command adds inner event mode triggers.

Input format

```
> AIE    start_event, end_event   < CR >
```

start_event : Specify the symbol name of the event that becomes the trace start trigger.

end_event : Specify the symbol name of the event that becomes the trace end trigger.

Input example

```
ERX>AIE  SUB1. START, SUB1, RETURN
ERX>
```

NOTE

AOE

Function

The **AOE** adds outer event mode triggers.

Input format

```
> AOE    start_event, end_event   < CR >
```

start_event : Specify the symbol name of the event that becomes the trace interrupt trigger.

end_event : Specify the symbol name of the event that becomes the trace restart trigger.

Input example

```
ERX>AOE  SUB2. START, SUB2. RETURN
ERX>
```

NOTE

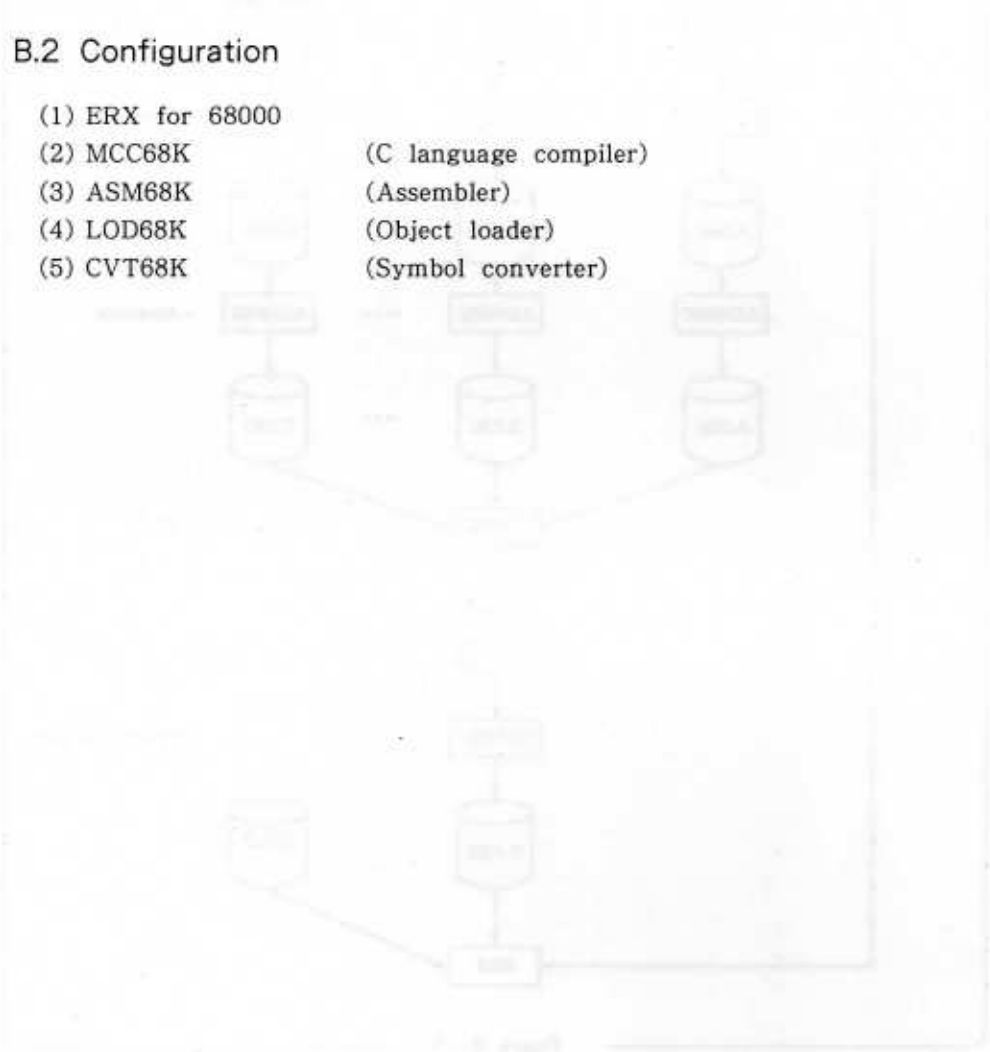
B. HLLD MACRO LIBRARY

B.1 General Description

The ERX for 68000 high - level language debugger is a real time high - level language debugger realized in ERX on the basis of the symbols and objects output by the user generated source program and compiler.

B.2 Configuration

(1) ERX for 68000
(2) MCC68K (C language compiler)
(3) ASM68K (Assembler)
(4) LOD68K (Object loader)
(5) CVT68K (Symbol converter)



NOTE

B.3 SYSTEM FLOW

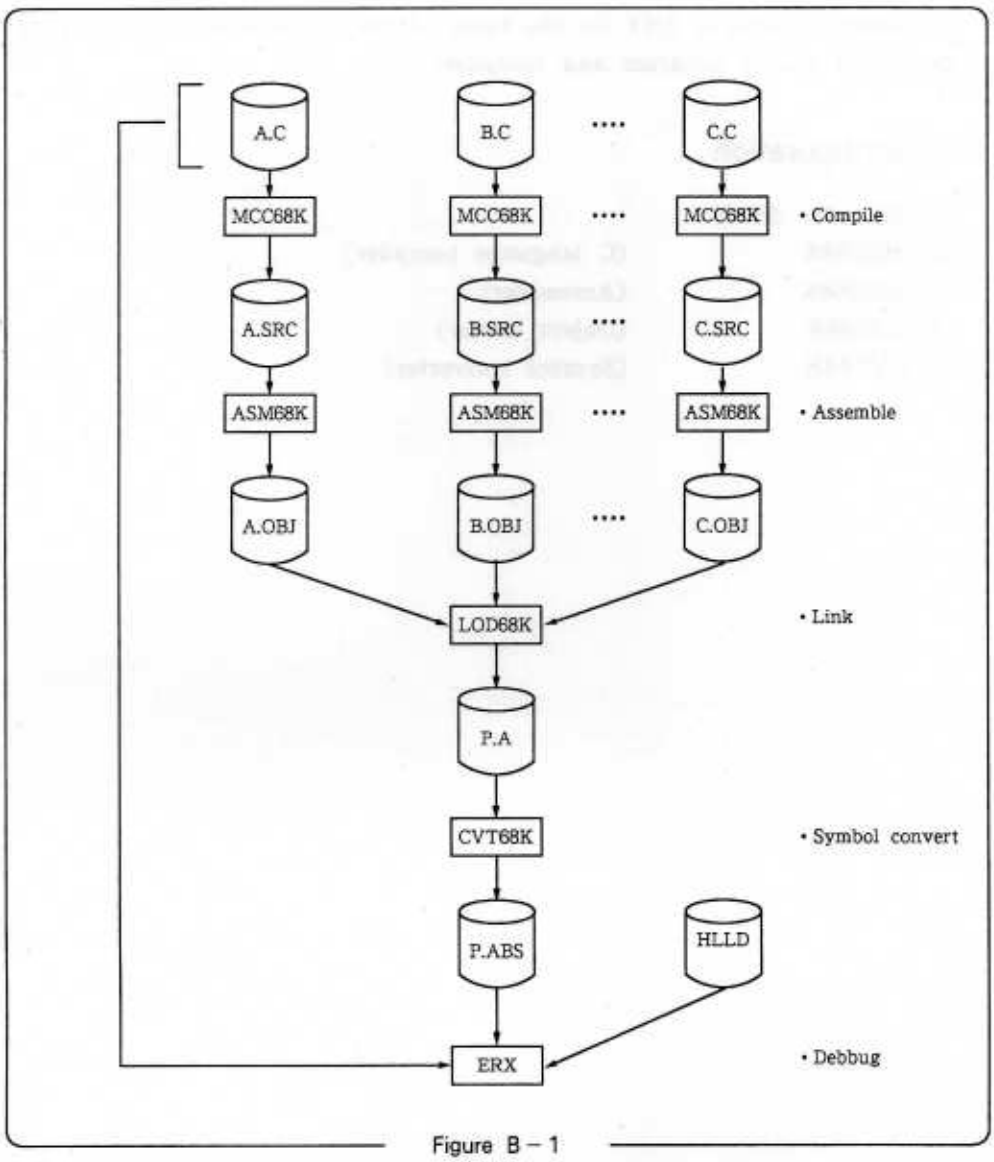


Figure B – 1

NOTE

B.4 Explanation of Functions

The ERX high level language debugger has the following functions in addition to the ERX functions.

B.4.1 Command help

The command help function displays the high level language debugging command syntax.

B.4.2 Break point line setting

The breakpoint line setting function sets a break point in a desired source line number.

B.4.3 Break point line release

The break point line release function releases the break point set in the source line number.

B.4.4 Source line step execution

The source line step execution function executes the specified lines of the source program.

B.4.5 Break source line display

The break source line display function displays the source of the line that caused emulation break.

B.4.6 Break source line dump display

The break source line dump display function displays dump of the memory corresponding to the line that caused emulation break.

B.4.7 Character string variable display

The character - string variable display function displays character - string variables by the NULL code or up to 128 bytes by the ASCII code.

B.4.8 Local area dump display

The local area dump display function displays dump of the local area used by the current functions.

NOTE

B.4.9 Stack frame display

The stack frame display function displays function names and arguments from the current position to route function.

B.4.10 History source display

The history source display function displays the history contents by the source line.

B.4.11 Source program display

The source program display function displays the specified source program.

B.5 Explanation of Operations

B.5.1 Compiler and assembler option specification

The compiler and assembler options of the source file (s) to be debugged must be specified as follows :

```
>MCC68K {file_name} /debug=lines;
>ASM68K {file_name} /d/case;
>LOD68K {file_name} ;
```

For details, see the MCC68K User's Guide and ASM68K User's Guide.

B.5.2 Program execution setup

B.5.2.1 Start ERX68K and prepare for debugging.

(1) Mapping (MAp command)
(2) Register reset (R RES command)
(3) Program loading (Load command)
(4) High – level language debug macro loading (HLLD command)
(5) Register reset (R RES command)
(6) Source file directory specification (View command)

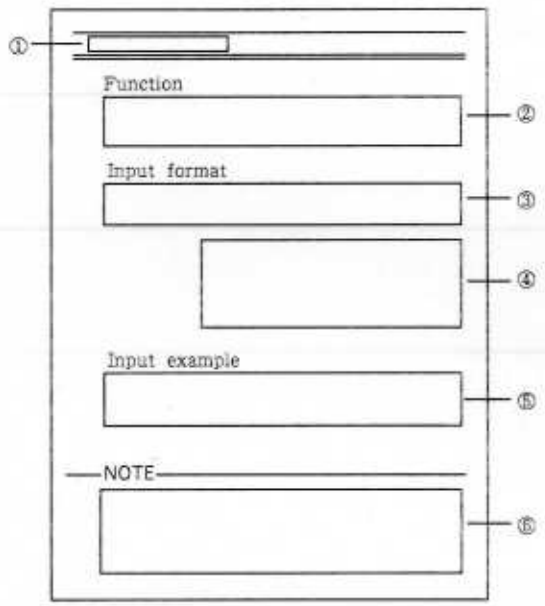
Execution setup is needed each time ERX is started. This operation can be done easily by using the SETUP command (see the standard macro library).

NOTE

B.6 Command Functions

This section explains the commands in alphabetical order.
The commands are explained in the following format.

B.6.1 Command explanation format



① Command name	:	Name of the command to be input. Uppercase characters cannot be omitted. (Lowercase characters can be omitted.)
② Function	:	Command function
③ Input format	:	How to activate the command The command is indicated in abbreviation. A lowercase character string indicates a parameter.
④ Input parameters	:	Lowercase characters in the input format. Required parameters are explained.
⑤ Input example	:	Actual input example
⑥ NOTE	:	Note (s) on use

NOTE

HLLD

Function

The **HLLD** command defines a command for debugging varieties of high level languages.

Input format

```
> HLLD   < CR >
```

Input example

```
ERX>HLLD
ERX>
```

NOTE

MH

Function

The MH command displays the command syntax of the high level language debugger.

Input format

```
> MH  HLLD < CR >
```

HLLD : Specify the high – level language debugger.

Input example

```
ERX>MH HLLD

     Help messages

ERX>
```

NOTE

BL

Function

The BL command sets a break－point in the desired source line number.

Input format

```
> BL source_line   < CR >
```

source_line : Specify the source program line number.

Input example

```
ERX>BL 52
ERX>BL *
```

NOTE

NBL

Function

The NBL command releases the break - point set in the source line number.

Input format

```
> NBL source_line   < CR >
```

source_line : Specify the source program line number.

Input example

```
ERX>NBL 52
ERX>NBL *
```

NOTE

T

Function

The **T** command executes the specified lines of the source program.

Input format

```
>T [step]   <CR>
```

step : Specify the number of source steps to be executed in a decimal number.
The default is one.

Input example

```
ERX>T
ERX>T 10
```

NOTE

BX

Function

The BX command displays the source of the line that caused emulation break.

Input format

```
> BX   < CR >
```

Input example

```
ERX>BX

    Break source line

ERX>
```

NOTE

DX

Function

The DX command displays dump of the memory corresponding to the line that caused emulation break.

Input format

```
> DX    [Length]    < CR >
```

Length : Specify the display byte count in a hexadecimal number. If this parameter is omitted, dump up to the next symbol address is displayed.

Input example

```
ERX>DX
ERX>DX 100
```

NOTE

DS

Function

The **DS** command displays a character – string variable by NULL (' \0') code or up to 128 bytes by ASCII characters.

Input format

```
> DS    address < CR >
> DS    event_symbol < CR >
```

address : Specify the display start address in a hexadecimal number.
event_symbol : Specify the display start symbol.

Input example

```
>DS 1000
>DS MAIN.MESSAGE
```

NOTE

DL

Function

The **DL** command displays dump of the local area used by the current functions.

Input format

```
> DL   < CR >
```

Input example

```
ERX>DL
```

NOTE

H/V

Function

The H/V displays the history file contents by the source lines.

Input format

```
> H/V  [beg_point]  [,end_point]  < CR >
```

beg_point : Specify the search start point in a decimal number from 1 to 8191.
If this parameter is omitted, the size of the storage traced currently is used.

end_point : Specify the search end point in a decimal from 1 to 8191.
The default is one.

Input example

```
ERX>H/V
ERX>H/V  100
ERX>H/V  2000, 1900
```

NOTE

View

Function

The View command displays the directory path of the View command.

Input format

```
>V/P  <CR>
```

Input example

```
>V/P
```

NOTE

View

Function

The **View** command sets the directory path of the **View** command.

Input format

```
> V/P  path   < CR >
```

path : Specify the directory path.

Input example

```
>V/P .
>V/P \ZAX
>V/P \ZAX \SOURCE
```

NOTE

View

Function

The View command displays the file extension of the View command.

Input format

```
> V/E < CR >
```

Input example

```
>V/E
```

NOTE

View

Function

The **View** command sets the file extension of the **View** command.

Input format

```
> V/E extention   < CR >
```

extension : Specify the file extension.

Input example

```
>V/E C
>V/E PLM
>V/E FOR
```

NOTE

View

Function

The View command displays the source corresponding to the current program counter.

Input format

```
>V  <CR>
```

Input example

```
>V
```

NOTE

View

Function

The View command displays the source file in MS – DOS.

Input format

```
> V filename [,start_line] [,end_line] < CR >
```

file_name : Specify the file name.
The directory path must be specified by VIEW/P.
If the extension is omitted, the default is the value specified by VIEW/P.

start_line : Specify the display start line in a decimal number.
The default is the first line.

end_line : Specify the display end line in a decimal number.
The number of lines can also be specified.
The lines must be specified by (+ (the number of lines)).
If this parameter is omitted, ten lines are displayed.

Input example

```
>V sievex.c, 10, 20
```

NOTE

View

Function

The **View** command displays the source starting at the specified symbol.

Input format

```
> V/S  [symbol]  [,line]  < CR >
```

symbol : Specify the symbol for displaying source data.
Only the line number symbol can be specified for source display.
If this parameter is omitted, display starts at the current program counter value.

line : Specify the number of lines of the source to be displayed.
The default is one.

Input example

```
>V/S
>V/S main.#10
>V/S main.#10,5
```

NOTE

View

Function

The View command sets color mode.

Input format

```
> V/C switch   < CR >
```

switch : ON Validates color mode
OFF Invalidates color mode

Input example

```
>V/C ON
>V/C OFF
```

NOTE

View

Function

The View command displays color mode.

Input format

```
> V/C   < CR >
```

Input example

```
>V/C
```

NOTE

C. AC CHARACTERISTICS

C.1 Signal Timing

(1) Signal Timings

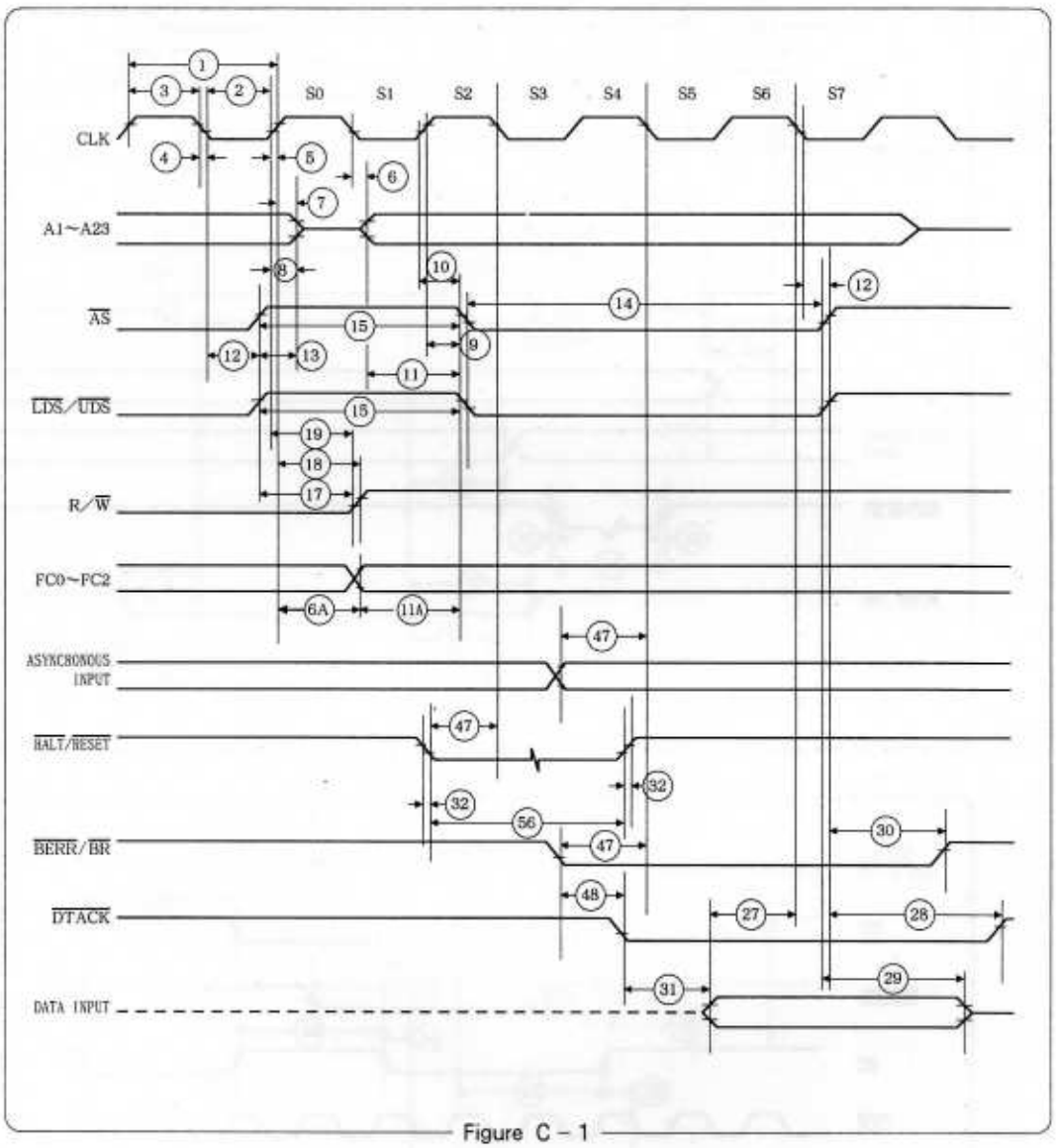


Figure C - 1

NOTE

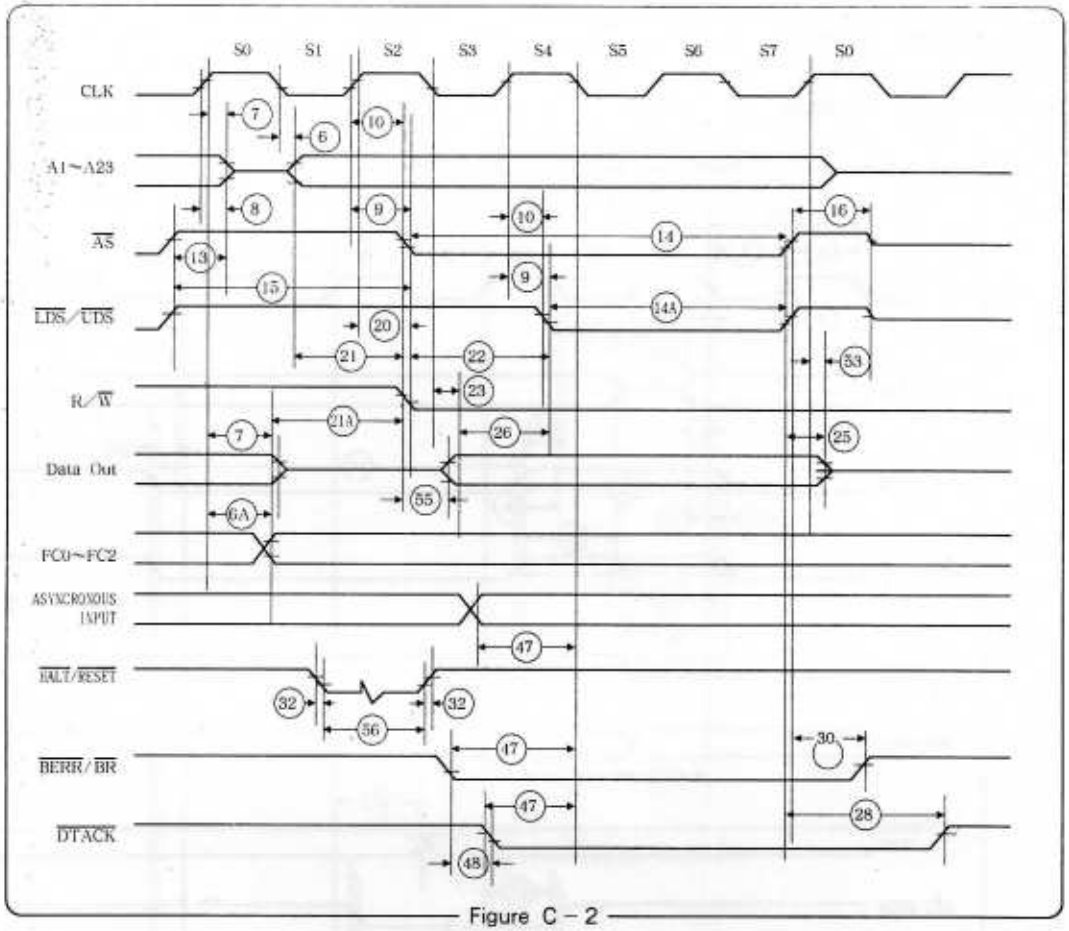


Figure C - 2

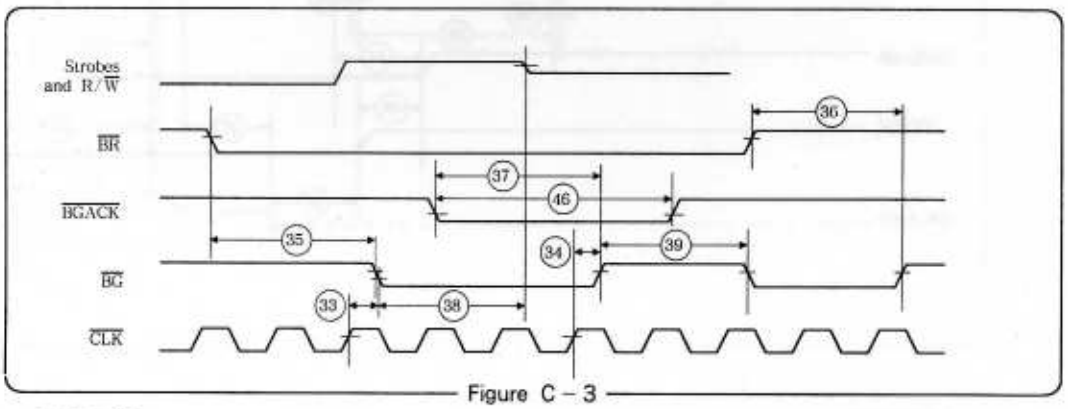


Figure C - 3

NOTE

(2) AC electric characteristics

Num.	Characteristic	Mark	12.5MHz MC68000L12		16.67MHz MC68000P12F		ERX		Unit
			Min	Max	Min	Max	Min	Max	
1	Clock Period	'cyc	80	250	60	125	80	250	ns
2	Clock Width Low	'CL	35	125	27	62.5	35	125	ns
3	Clock Width High	'CH	35	125	27	62.5	35	125	ns
4	Clock Fall Time	'Cf	—	5	—	5	—	5	ns
5	Clock Rise Time	'Cr	—	5	—	5	—	5	ns
6	Clock Low to Address	'CLAV	—	55	—	50	—	70	ns
6A	Clock High to FC Valid	'CHFCV	—	55	—	45	—	70	ns
7	Clock High to Address Data High Impedance	'CHAZx	—	60	—	50	—	75	ns
8	Clock High to Address/FC Invalid	'CHAZn	0	—	0	—	15	—	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low (Maximum)	'CHSLx	—	55	—	40	—	70	ns
10	Clock High to $\overline{AS}$, $\overline{DS}$ Low (Minimum)	'CHSLn	0	—	3	—	15	—	ns
11[2]	Address to $\overline{AS}$, $\overline{DS}$ (Read) Low/$\overline{AS}$ Write	'AVSL	0	—	15	—	0	—	ns
11A[2]	FC Valid to $\overline{AS}$, $\overline{DS}$ (Read)Low/$\overline{AS}$ Write	'FCVSL	40	—	30	—	40	—	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	'CLSH	—	50	—	40	—	65	ns
13[2]	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	'SHAZ	10	—	10	—	10	—	ns
14[2,5]	$\overline{AS}$, $\overline{DS}$ Witth to (Read)/$\overline{AS}$ Write	'SL	160	—	120	—	160	—	ns
14A[2]	$\overline{DS}$ Width Low (Write)	—	80	—	60	—	80	—	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width High	'SH	65	—	60	—	65	—	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$ High Impedace	'CHSZ	—	60	—	50	—	60	ns
17[2]	$\overline{AS}$, $\overline{DS}$ High to R/$\overline{W}$ High	'SHRH	10	—	10	—	10	—	ns
18[1]	Clock High to R/$\overline{W}$ High (Maximum)	'CHRHx	—	60	—	40	—	75	ns
19	Clock High to R/$\overline{W}$ High (Minimum)	'CHLHn	0	—	0	—	15	—	ns
20[1]	Clock High to R/$\overline{W}$ Low	'CHRL	—	60	—	40	—	75	ns
21[2]	Address Valid to R/$\overline{W}$ Low	'AVRL	0	—	0	—	0	—	ns
21A[2]	FC Valid to R/$\overline{W}$ Low	'FCVRL	30	—	20	—	30	—	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Low (Write)	'RLSL	30	—	20	—	30	—	ns
23	Clock Low Data Out Valid	'CLDO	—	55	—	50		65	ns
25[2]	$\overline{DS}$ High to Data Out Invalid	'SHDO	15	—	15	—	15	—	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	'DOSL	15	—	15	—	15	—	ns
27[6]	Data in to Clock Low (Setup Time)	'DICL	15	—	7	—	15	—	ns
28[2]	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High	'SHDAH	0	70	0	110	0	70	ns
29	DS High to Data Invalid (Hold Time)	'SHDI	0	—	0	—	0	—	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	'SHBEH	0	—	0	—	0	—	ns
31[2,6]	DTACK Low to Data In (Setup)	'DALDI	—	50	—	40	—	50	ns

Table C－1

NOTE

Num.	Characteristic	Mark	12.5MHz MC68000L12		16.67MHz MC68000P12F		ERX		Unit
			Min	Max	Min	Max	Min	Max	
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	tcyc	0	200	—	150	0	200	ns
33	Clock High to $\overline{BG}$ Low	tCL	—	50	—	40	—	65	ns
34	Clock High to $\overline{BG}$ High	tCH	—	50	—	40	—	65	ns
35	$\overline{BR}$ Low $\overline{BG}$ Low	tCf	1.5	3.0	1.5	3.5	1.5	3.0	Clk. Per
36	$\overline{BR}$ High $\overline{BG}$ High	tCr	1.5	3.0	1.5	3.5	1.5	3.0	Clk. Per
37	$\overline{BTACK}$ Low to $\overline{BG}$ High	tCLAV	1.5	3.0	1.5	3.5	1.5	3.0	Clk. Per
38	$\overline{BG}$ Low to Bus High Impedance (With $\overline{AS}$	tCHFCV	—	60	—	50	—	60	ns
39	$\overline{BG}$ Width High(Maximum)	tCHAZx	1.5	—	1.5	—	1.5	—	ns
46	BGACK Width	tCHAZn	1.5	—	1.5	—	1.5	—	ns
47[6]	Asynchronous Input Setup Time	tCHSLx	20	—	10	—	60	—	ns
48	$\overline{BERR}$ Low to $\overline{DTACK}$ Low (Note 3)	tCHSLn	50	—	10	—	50	—	ns
53	Data Hold from Clock High	tAVSL	0	—	0	—	15	—	ns
55	R/$\overline{W}$ to Data Bus Impedance Change	tFCVSL	10	—	0	—	40	—	ns
56	Halt?RESET Pulse Width (Note 4)	tCLSH	10	—	10	—	10	—	Clk. Per

Table C-2

NOTE

F. EMULATION CPU SOCKET

F.1 Replacing Emulation CPU

If emulation CPU of ERX318P for 68000 is replaced, MC68000 or MC68010 can be emulated.

Replace emulation CPU according to the following procedure :

(1) Remove the top cover from ERX318P for 68000.
Figure F – 1 shows the screw positions.

Figure F – 1

NOTE

(2) Under the top cover, there is an emulation CPU socket on the top substrate.

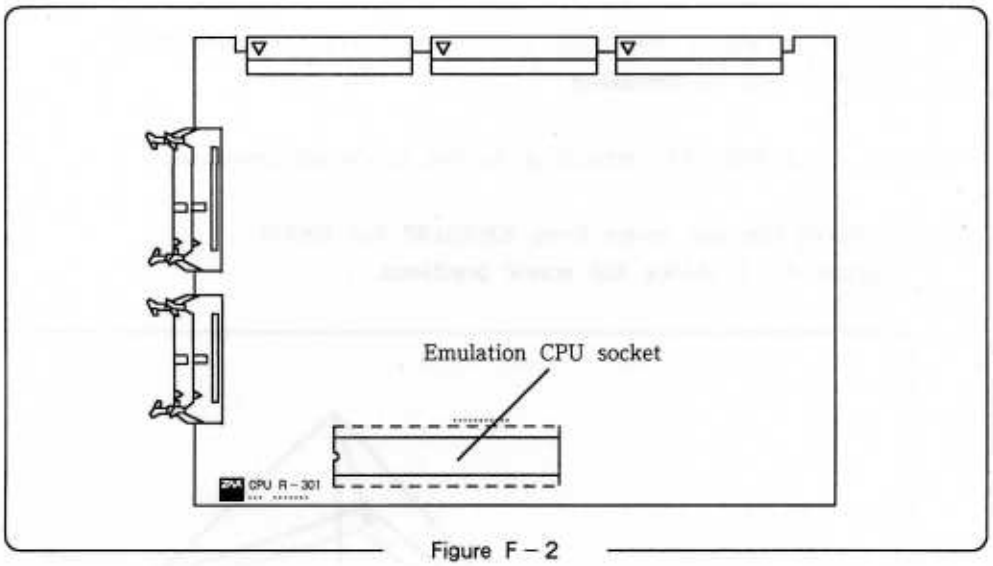


Figure F-2

(3) Insert the plug with mark "OPEN" into the emulation CPU socket with the driver. Mark "ACT" appears on the opposite and emulation CPU can be replaced. After that, insert the plug with mark "ACT" into the emulation CPU socket with the driver. Mark "OPEN" appears on the opposite. Thus, emulation CPU can be replaced. Figure F-3 shows how to use a screwdriver.

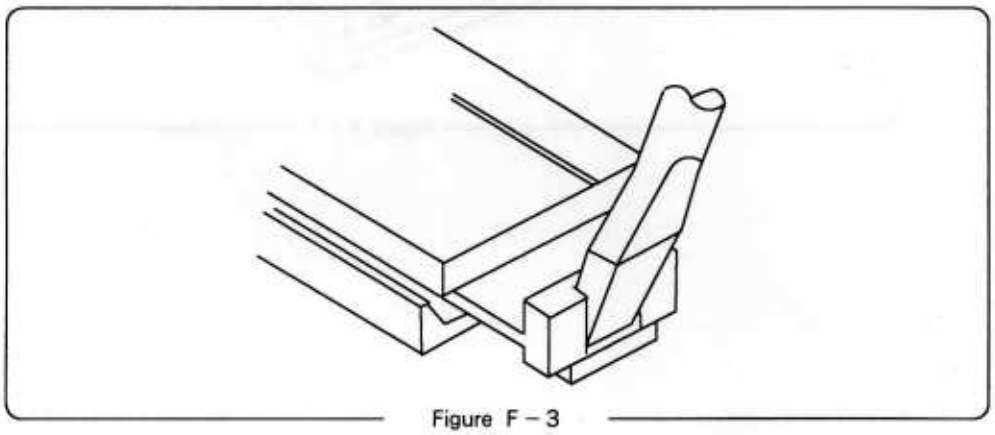


Figure F-3

NOTE

(4) Attach the top cover to ERX318P for 68000.
Figure F – 4 shows the screw positions.

Figure F – 4

NOTE

NOTE

G. CVT68K

Function

CVT68K converts Microtec ASM68K symbol/absolute fole (.ABS) to the emulator standard symbol/absolute file.

Input format

```
> CVT68K  [-s]  infile  [outfile]  <CR>
```

infile	:	The infile is ASM68K symbol/absolute file.
outfile	:	The outfile is emulator standard symbole/absolute file. If outfile is omited, infile will be replaced.
-s	:	If -s option specified, symbol file will be generated. The extesion of symbol file is .sym. The contents of symbol file is sorted by address order. With sorted symbol files, emulator software can access sy mboles quickly.

Input example

```
ERX68K>CVT68K SAMPLE.ABS SAMPLE2.ABS

ERX68K>CVT68K SAMPLE.ABS

ERX68K>CVT68K -s SAMPLE.ABS SAMPLE2.ABS
```

NOTE

TECHNICAL REPORT

XYZ

TECHNICAL REPORT

ERX318 for 68000 • 68010

TECHNICAL REPORT

Rev. E1.01

ERX318 for 68000 · 68010 TECHNICAL REPORT

CONTENTS

A. HOW TO USE THE BREAK FUNCTION

A.1 Setting Auto Break

The auto break function uses character strings (symbols) in the symbol definition used in the program source file.
Each symbol has specific address.
Set each symbol name in an event point by using the **Load** command. The auto break function is based on event break point setting. The auto break function defines all events (symbols) as break – points using the wild card function.

An example for using the auto break function is shown below.

```
ERX>b *=on
ERX>g/w

D0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E002    SSP = 0000B000    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E002          CLR.L  D1
<Event "MAIN.MAIN" Break>

ERX>g/w

D0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E008    SSP = 0000B000    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E008          MOVE.W  D_CMDREG,D0
<Event "MAIN.CMDIN" Break>

ERX>g/w

D0-7 00000021 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E012    SSP = 0000B000    USP = 00000000     SR = -S7----- = 2700
SP:00E012          BNE.L  MAIN_JP1
<Event "MAIN.D CMDREG" Break>
```

NOTE

```
ERX>g/w

D0-7 00000021 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E022    SSP = 0000B000    USP = 00000000    SR = -S7--Z-- = 2704
SP:00e022          BNE.L  MAIN_JP2
<Event "MAIN.MAIN _JP1" Break>

ERX>g/w

D0-7 00000021 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D002 00000000 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E040    SSP = 0000AFFC    USP = 00000000    SR = -S7--Z-- = 2704
SP:00E040          MOVEA.L #$0000D102, A1
<Event "MAIN.MEMWR" Break>

ERX>g/w

D0-7 00000021 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D002 00000000 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E040    SSP = 0000AFFC    USP = 00000000    SR = -S7--Z-- = 2704
SP:00E04E  MEMWR_LOP    MOVE.B (A0)+, D0
<Event "MAIN.D SIZE" Break>

ERX>g/w

D0-7 00000041 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D003 0000D102 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E050    SSP = 0000AFFC    USP = 00000000    SR = -S7----- = 2700
SP:00E050          NOP
<Event "MAIN.MEMWR_LOP" Break>

ERX>g/w

D0-7 00000041 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D003 0000D102 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E056    SSP = 0000AFFC    USP = 00000000    SR = -S7----- = 2700
SP:00E056          SUBQ.B #1. D1
```

NOTE

```
ERX>g/w

D0-7 00000041 000000FF 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000003 0000D103 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E04E    SSP = 0000AFFC    USP = 00000000    SR = -S7XN--C = 2719
SP:00E04E  MEMWR_lop    MOVE.B (A0)+,D0
<Event "MAIN.MEMWR LOPE" Break>

ERX>b MAIN.MEMWR L#=off
ERX>g/w

D0-7 00000042 000000FF 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D004 0000D103 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E052    SSP = 0000AFFC    USP = 00000000    SR = -S7X---- = 2710
SP:00E 052         MOVE.B D0,(A1)+
<Event "MAIN.D_SRCBUF" Break>

ERX>g/w

D0-7 00000042 000000FF 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D004 0000D104 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E056    SSP = 0000AFFC    USP = 00000000    SR = -S7X---- = 2710
SP:00E056          SUBQ.B #1,D1
<Event "MAIN.D_DESBUF" Break>
ERX>g/w

D0-7 00000048 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D102 0000D202 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E02A    SSP = 0000B000    USP = 00000000    SR = -S7--Z-- = 2704
SP:00E02A          BRA.LB MAIN_END

<Event MAIN.MEMWR _END" Break>
ERX>g/w

D0-7 00000048 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D102 0000D202 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E032    SSP = 0000B000    USP = 00000000    SR = -S7--Z-- = 2704
SP:00E032  MAIN.MAIN_END" Break>
```

NOTE

A.2 Setting Break – Points in All Commands between Specified Symbols

The range break function sets break – points in all commands between the specified symbols.

An example for using the range break function is shown below.

```
ERX>
ERX>reset
ERX>r reset
ERX>b *=off
ERX>b/r MAIN.MEMWR, MAIN.MEMWR_LOP
ERX>b
 No.  Module  Stmbol      FA Address St Size Data Count EX SEQ B C H P T
&25  $$   _25          -- 00E03A  -- ---- ---- ----  --  -- * - - - -
00E04E

ERX>g/w MAIN.MAIN

D0-7 00000021 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D002 00000000 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E040    SSP = 0000AFFC    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E040          MOVEA.L #$0000D102, A1
<Event "_25" Break>

ERX>g/w

D0-7 00000021 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D002 0000D102 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E046    SSP = 0000AFFC    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E046          MOVE.W D_SIZE, D0
<Event "_25" Break>
```

NOTE

A.3 Setting Break – Points in All Accesses between Specified Memory

The event break function is used when breakpoints are set in all accessed spaces in the specified memory. The range addresses can be specified by using the **Event** command, conditions outside the range may also be specified.
An example is shown below.

```
ERX>
ERX>reset
ERX>r reset
ERX>b *=off
ERX>ev MAIN.D_DESBUF
Symbol     : MAIN.D_DESBUF
Faddress   : --          =
Address    : 00D102      = 0D102, 0D201
Status     : --          =
Size       : ----        =
Data       : ----        =
Passcount  : ----        =
External   : --          =
Sequential : --          =
ERX>
ERX>b MAIN.D_DESBUF
ERX>b
 No.  Module  Stmbol     FA Address St Size Data Count EX SEQ B C H P T
&13  MAIN    D_DESBUF    -- 00D102  -- ---- ---- ----  --  -- * - - - -
                           00D201

ERX>g/w MAIN.MAIN

D0-7 00000041 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000003 0000D103 00000000 00000000 00000000 00000000 00000000 00000000
PC = 0000E056    SSP = 0000AFFC    USP = 00000000    SR = -S7----- = 2700
SP:00E056                SUBQ.B #1,D1
<Event "MAIN.D_DESBUF" Break>
```

NOTE

A.4 Setting Break – Points by Using The Sequential Function

Break – points can be set by using the sequential conditions of the **Event** command. ERX can set sequential conditions of four levels. Actually, the trigger of each function can be generated by combining four event points. An example for setting break – points using the two – level sequential conditions is shown below.

```
ERX>
ERX>b *=off
ERX>ev MAIN.D_DESBUF
Symbol     : MAIN.D_DESBUF
Faddress   : --          =
Address    : 00D102, 00D201 =
Status     : --          =
Size       : ----        =
Data       : ----        =
Passcount  : ----        =
External   : --          =
Sequential : --          = #1
ERX>ev MAIN.MEMWR_LOP
Symbol     : MAIN.MEMWR_LOP
Faddress   : --          =
Address    : 00E04E      =
Status     : --          =
Size       : ----        =
Data       : ----        =
Passcount  : ----        =
External   : --          =
Sequential : --          = #2
ERX>ev MAIN.MEMWR_LOP
ERX>b
 No.  Module  Symbol      FA Address St Size Data Count EX SEQ B C H P T
&14  MAIN    MEMWR_LOP    -- 00E04E  -- ---- ---- ----  --  #2 * - - - -
ERX>g/w MAIN.MAIN

D0-7 00000042 0000D103 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D004 0000D103 00000000 00000000 00000000 00000000 00000000 0000AFF8
PC = 0000E050    SSP = 0000AFF8    USP = 00000000    SR = -S7X---- = 2710
SP:00E050          NOP
<Event "MAIN.MEMWR_LOP" Break>
```

NOTE

```
ERX>g/w

D0-7 00000043 000000FE 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000D05 0000D104 00000000 00000000 00000000 00000000 00000000 0000AFF8
PC = 0000E050    SSP = 0000AFF8    USP = 00000000     SR = -S7----- = 2700
SP:00E050          NOP
<Event "MAIN.MEMWR_LOP" Break>
```

NOTE

A.5 Setting Break – Points by Address, Status, and Pass Count

Break – points can be set automatically by the **Break** command.
Actually, the event function is used, so event points are registered automatically.
Default module name (specified in the **DEFM** command) or (serial number) is selected as an event name.

```
ERX>
ERX>reset
ERX>r reset
ERX>b *=off
ERX>b MAIN.MEMWR_LOP, MR, 5
ERX>g/w MAIN.MAIN

D0-7 00000045 000000FC 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D005 0000D104 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E050    SSP = 0000AFFC    USP = 00000000     SR = -S7----- = 2700
SP:00E050          NDP
<Event "__29" Break>

ERX>
```

NOTE

The event function registers event points automatically but does not delete, so the event points must be deleted by the **EDelete** command.

B. HOW TO USE THE HISTORY

B.1 Setting End Moniter

The end monitor mode is the ERX default setting. When the ERX is started, if the history mode is not specified, it will default to the end monitor mode setting.
When emulation is broken during debugging, history can be displayed.
This mode has the highest activity ratio.
End monitor mode is set by using the EM command. Storing history information with in the specified storage range starts when emulaion is broken.

```
ERX>
ERX>B MAIN.MAIN _END
ERX>EM
ERX>G MAIN.MAIN

D0-7 00000048 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D102 0000D202 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E032    SSP = 0000B000    USP = 00000000    SR = -S7--Z-- = 2704
SP:00E032 MAIN_END     JMP  MAIN_ END
<Event "MAIN.MAIN_END" Break>
```

NOTE

```
ERX>H/D 30
Point E FA Addr Data St              Opecode  Operand
 0030   SD 00D100  47 MR
 0029   SP 00E052                    MOVE.B   D0, (A1)+
 0028   SP 00E054        MEMWR_LOPM   NOP
 0027   SD 00D200  47 MW
 0026   SP 00E056                    SUBQ.B   #1, D1
 0025   SP 00E058        MEMWR_LOPE   BNE.S    MEMWR_LOP
 0023   SP 00E04E        MEMWR_LOP    MOVE.B   (A0)+, D0
 0022   SP 00E050                    NOP
 0021   SD 00D101  48 MR
 0020   SP 00E052                    MOVE.B   D0, (A1)+
 0019   SP 00E054        MEMWR_lopm   NOP
 0018   SD 00D201  48 MW
 0017   SP 00E056                    SUBQ.B   #1, D1
 0016   SP 00E058        MEMWR_LOPE   BNE.B    MEMWR_LOP
 0015   SP 00E05A                    NOP
 0014   SP 00E05C        MEMWR_END    RTS
 0012   SD 00AFFC 0000 MR
 0011   SD 00AFFE E02A MR
 0010   SP 00E02A                    BRA.L    MAIN_END
 0008 > SP 00E032        MAIN_END     JMP      MAIN_END
 0003 > SD 00AFFE E032 MW
 0002 > IA FFFFFE E032 IA
 0001   SD 00AFFA 2704 MW
 **** Pause ****
ERX>
```

NOTE

B.2 Setting Center Events

The center event function stores the contents of 8191 – time storage including before and after the trigger point by using the event signal as a storage trigger. An example is shown below.

```
ERX>
ERX>B MAIN.MAIN_END
ERX>EV MAIN.D_DESBUF
Symbol    : MAIN.D_DESBUF
Faddress  : --          =
Address   : 00D102      = 0D180
Status    : --          = MW
Size      : ----        = /
ERX>CE MAIN.D_DESBUF
em
h s=MAIN.D_DESBUF a=on l=4095

Macro CE completed.
ERX>H/S
 No.  Module  Symbol     FA Address St Size Date Count EX SEQ B C H P T
&13  MAIN    D_DESBUF    -- 00D180  MW ---- ---- ----  -- --  - - S - -
Auto Start : ON
Length     : 4095
Multi      : OFF
Freeze     : OFF
Break      : OFF
ERX>G MAIN.MAIN

D0-7 00000048 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D102 0000D202 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E032    SSP = 0000B000    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E032  MAIN_END         JMP    MAIN_END
<Event "MAIN.MAIN_END" Break>
```

NOTE

```
ERX>H/D 1190, 1170
Point E FA Addr Data St                    Opcode  Operand
 1190   SP 00E052                          MOVE. B  D0, (A1)+
 1189   SP 00E054          MEMWR_LOPM      NOP
 1188   SD 00D17F  46 MW
 1187   SP 00E056                          SUBQ. B  #1, D1
 1186   SP 00E058          MEMWR_LOPE      BNE. S  MEMWR_LOP
 1184   SP 00E04E          MEMWR_LOP       MOVE. B  (A0)+, D0
 1183   SP 00E050                          NOP
 1182   SD 00D080  47 MR
 1181   SP 00E052                          MOVE. B  D0, (A1)+
 1180   SP 00E054          MEMWR_LOPM      NOP
 1179 * SD 00D180  47 MW  D_DESBUF
 1178   SP 00E056                          SUBQ. B  #1, D1
 1177   SP 00E058          MEMWR_LOPE      BNE. S  MEMWR _LOP
 1175   SP 00E04E          MEMWR_LOP       MOVE. B  (A0)+, D0
 1174   SP 00E050                          NOP
 1173   SD 00D081  48 MR
 1172   SP 00E052                          MOVE. B  D0, (A1)+
 1174   SP 00E054          MEMWR_LOPM      NOP
 1170 * SD 00D181  48 MW
ERX>
```

NOTE

B.3 Setting Inner Events

The inner event function sets event points and executes the history storage function by using the event signal as the history start/stop trigger. The storage function stores the contents in the specified range.

```
ERX>
ERX>B MAIN.MAIN_END
ERX>IE MAIN.MEMWR, MAIN.MEMWR_LOP
em
H s=MAIN.MEMWR e=MAIN.MEMWR_LOP a=off

Macro IE completed.
ERX>H/S
 No.  Module  Symbol     FA Address St Size Date Count EX SEQ B C H P T
&12  MAIN    MEMWR       -- 00E03A  -- ---- ---- ----  -- --  - - S - -
&14  MAIN    MEMWR_LOP   -- 00E04E  -- ---- ---- ----  -- --  - - E - -
Auto Start : OFF
Length     : $
Muiti      : OFF
Freeze     : OFF
Break      : OFF
ERX>G MAIN.MAIN

D0-7 00000048 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D102 0000D202 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E032    SSP = 0000B000    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E032  MAIN_END    JMP   MAIN_END
<Event "MAIN.MAIN_END" Break>

ERX>H/D
Point E FA Addr Data St              Opcode  Operand
 0012 * SP 00E03A       MEMWR        MOVEA.L #$0000D002, A0
 0009   SP 00E040                    MOVEA.L #$0000D102, A1
 0006   SP 00E046                    MOVE.W  D_SIZE, D0
 0003   SP 00E04C                    MOVE.W  D_D0, D1
 0002   SD 00D202 0000 MR  D_SIZE
 0001 * SP 00E04E       EMWR_LOP     MOVE.B  (A0)+, D0
 **** Pause ****
ERX>
```

NOTE

NOTE

C. HOW TO USE THE STUB FUNCTION

C.1 How to Use Program Patch

Program operations can be changed freely through ERX.
ERX intervention can be realized by combining operation commands.
Actually, NMI is generated by executing the illegal instruction command, emulation is interrupted, and the specified command is executed. The stub function is not a real time debug function. Program modification, addition, and deletion can be checked before the program source file is modified.
Because the illegal instruction command must be inserted in the program, the stub function can be specified only in a read, write, and access memory. If the stub function is to be specified in a read only memory, after required data is transferred from the read only memory of the target system to the emulation memory, the stub function can patch the program using the emulation memory (state : read only memory).
An example is shown below.

```
ERX>ex MAIN.D_SIZE=3
ERX>di MAIN.MEMWR_LOP, +0b
SP:00E04E 1018          MEMWR_LOP     MOVE.B  (A0)+, D0
SP:00E050 4E71                        NOP
SP:00E052 12C0                        MOVE.B  D0, (A1)+
SP:00E054 4E71          MEMWR_LOPM    NOP
SP:00E056 5301                        SUBQ.B  #1, D1
SP:00E058 66F4          MEMWR_LOPE    BNE.S   MEMWR_LOP
SP:00E05A 4E71                        NOP
SP:00E05C 4E75          MEMWR_END     RTS
SP:00E05E 4E75          CMDIN         RTS
SP:00E060 0010FFF7                    ORI.B   #$F7, (A0)
SP:00E064 00FF                        ****
```

NOTE

```
ERX>stub MAIN.MEMWR_LOP,"PATCH"
ERX>macro PATCH
MACRO>echo off
MACRO>cput "PATCH /n"
MACRO>r d0=mb(reg(a0))
MACRO>r a0=reg(a0)+1
MACRO>g/w
ERX>b MAIN.MAIN_END
ERX>g/w MAIN.MAIN
PATCH
PATCH
PATCH

D0-7 00000043 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D105 00000000 00000000 00000000 00000000 00000000 00000000 00000000
PC = 0000E032    SSP = 0000B000    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E032  MAIN_END         JMP     MAIN_END
<Event "MAIN.MAIN_END" Break>

ERX>
```

NOTE

D. HOW TO USE ON BREAK

D.1 How to Use Program Patch

Program operations can be changed freely through ERX.
ERX intervention can be realized by combining operation commands.
In actual processing, NMI is generated by executing the illegal instruction command, emulation is interrupted, and the specified command is executed. The on break function is not a real time debug function. Program modification, addition, and deletion can be checked before the program source file is modified.
Any memory type (RO, RW, US) can be used because the event signal is used.
A use example is shown below.

```
ERX>ex MAIN.D_SIZE=3
ERX>di MAIN.MEMWR_LOP,+0b
SP:00E04E 1018          MEMWR_LOP     MOVE.B  (A0)+,D0
SP:00E050 4E71                        NOP
SP:00E052 12C0                        MOVE.B  D0,(A1)+
SP:00E054 4E71          MEMWR_LOPM    NOP
SP:00E056 5301                        SUBQ.B  #1,D1
SP:00E058 66F4          MEMWR_LOPE    BNE.S   MEMWR_LOP
SP:00E05A 4E71                        NOP
SP:00E05C 4E75          MEMWR_END     RTS
SP:00E05E 4E75          CMDIN         RTS
SP:00E060 0010FFF7                    ORI.B   #$F7,(A0)
SP:00E064 00FF                        ****
```

NOTE

```
ERX>onbreak MAIN.MEMWR_LOP, "PATCH"
ERX>macro PATCH
MACRO>echo off
MACRO>cput "PATCH/n"
MACRO>r d0=mb(reg(a0))
MACRO>r a0=reg(a0)+1
MACRO>g/w
MACRO>
ERX>b MAIN.MAIN_END
ERX>g/w MAIN.MAIN
PATCH
PATCH
PATCH

D0-7 00000046 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D008 00000000 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E032    SSP = 0000B000    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E032  MAIN_END          JMP    MAIN_END
<Event "MAIN.MAIN_END" Break>

ERX>
Batch file ERX completed.
ERX68K>q
exiting ERX68K.
```

NOTE

E. HOW TO USE THE COVERAGE

E.1 How to Use The Coverage Function

The coverage function measures to see whether the target program has been tested enough and represents program reliability by numeric values.

In actual measurement, a result is obtained by a function that sets the coverage memory when emulation processor accesses an address. When the coverage function is to be used, allocate coverage memories in the memory space to be measured and clear all coverage memories.

After that, when the processor is put in emulation state, it executes the coverage function and plots the accessed memory addresses in the coverage memory. Regardless of whether emulation is being executed or broken, the distribution of executed programs and accessed memories can be measured.

The measured values are represented by percentage.

A use example is shown below.

```
ERX>
ERX>ev MAIN.MEMWR_LOPM
Symbol     : MAIN.MEMWR_LOPM
Faddress   : --        =
Address    : 00E054    =
Status     : --        =
Size       : ----      =
Data       : ----      =
Passcount  : ----      = 80
External   : --        = /
ERX>b MAIN.MEMWR_LOPM
ERX>ev MAIN.D_SRCBUF
Symbol     : MAIN.D_SRCBUF
Faddress   : --        =
Address    : 00D002    = 0D002, 0D101
Status     : --        = /
```

NOTE

```
ERX>ev MAIN.D_DESBUF
Symbol    : MAIN.D_DESBUF
Faddress  : --          =
Address   : 00D102      = 0D102, 0D201
Status    : --          = /
ERX>cov *=on
ERX>cov/c1
ERX>g MAIN.MAIN

D0-7 00000048 00000081 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D082 0000D182 00000000 00000000 00000000 00000000 00000000 0000AFFC
PC = 0000E056    SSP = 0000AFFC    USP = 00000000    SR = -S7----- = 2700
SP:00E056                  SUBQ.B #1,D1
<Event "MAIN.MEMWR_LOPM" Break>

ERX>cov/p
 No.  Module  Symbol     FA  Start  End    St  P/U   Cal
&7    MAIN    D_CMDREG   --  00D000 00D000 --   *  100.00%
&17   MAIN    D_SIZE     --  00D202 00D202 --   *  100.00%
&18   MAIN    MAIN       --  00E01E 00E01E --   *  100.00%
&12   MAIN    MAIN_JP1   --  00E01E 00E01E --   *  100.00%
&10   MAIN    MEMWR      --  00E03A 00E03A --   *  100.00%
&14   MAIN    MEMWR_LOP  --  00E04E 00E04E --   *  100.00%
&16   MAIN    MEMWR_LOPM --  00E054 00E054 --   *  100.00%
&15   MAIN    MEMWR_LOPE --  00E058 00E058 --   *  100.00%
&8    MAIN    CMDIN      --  00E05E 00E05E --   *  100.00%
```

NOTE

```
ERX>cov/u
 No.   Module  Symbol      FA  Start  End    St  P/U   Cal
&1     MAIN    V_MEMRD     --  000020 000020 --   -    0.00%
&24    MAIN    V_MEMWR     --  000021 000021 --   -    0.00%
&5     MAIN    PER_START   --  009000 009000 --   -    0.00%
&20    MAIN    PER_LOP1    --  009004 009004 --   -    0.00%
&21    MAIN    PER_LOP2    --  00900A 00900A --   -    0.00%
&22    MAIN    PER_LOP3    --  009014 009014 --   -    0.00%
&23    MAIN    PER_LOP4    --  009018 009018 --   -    0.00%
&4     MAIN    PER_END     --  009026 009026 --   -    0.00%
&6     MAIN    V_STACK     --  00B000 00B000 --   -    0.00%
&9     MAIN    D_SRCBUF    --  00D002 00D102 --   -   50.00%
&13    MAIN    D_DESBUF    --  00D102 00D201 --   -   50.00%
&11    MAIN    MAIN_JP2    --  00E02E 00E02E --   -    0.00%
&19    MAIN    MAIN_END    --  00E032 00E032 --   -    0.00%
&3     MAIN    MEMRD       --  00E038 00E038 --   -    0.00%
&2     MAIN    MEMWR_END   --  00E05C 00E05C --   -    0.00%
```

NOTE

```
ERX>cov/d
 No.   Module  Symbol      FA  Start  End    St     Pass              Unpass
&1     MAIN    V_MEMRD     -- 000020 000020 --                       000020-000020
                                                   (  0.00%)          (100.00%)
&24    MAIN    V_MEMWR     -- 000021 000021 --                       000021-000021
                                                   (  0.00%)          (100.00%)
&5     MAIN    PER_START   -- 009000 009000 --                       009000-009000
                                                   (  0.00%)          (100.00%)
&20    MAIN    PER_LOP1    -- 009004 009004 --                       009004-009004
                                                   (  0.00%)          (100.00%)
&21    MAIN    PER_LOP2    -- 00900A 00900A --                       00900A-00900A
                                                   (  0.00%)          (100.00%)
&22    MAIN    MAIN _LOP3  -- 009014 009014 --                       009014-009014
                                                   (  0.00%)          (100.00%)
&23    MAIN    PER_LOP4    -- 009018 009018 --                       009018-009018
                                                   (  0.00%)          (100.00%)
&4     MAIN    PER_END     -- 009026 009026 --                       009026-009620
                                                   (  0.00%)          (100.00%)
&6     MAIN    V_STACK     -- 00B000 00B000 --                       00B000-00B000
                                                   (  0.00%)          (100.00%)
&7     MAIN    D_CMDREG    -- 00D000 00D000 --                       00D000-00D000
                                                   (100.00%)          (  0.00%)
&9     MAIN    D_SRCBUF    -- 00D002 00D101 -- 00D002-00D081 00D082-00D101
                                                   ( 50.00%)          ( 50.00%)
&13    MAIN    D_DESBUF    -- 00D102 00D201 -- 00D102-00D181 00D182-00D201
                                                   ( 50.00%)          ( 50.00%)
```

NOTE

```
&17  MAIN   D_SIZE       -- 00D202 00D202 -- 00D202-00D202
                                              (100.00%)      (  0.00%)
&18  MAIN   MAIN         -- 00E000 00E000 -- 00E000-00E000
                                              (100.00%)      (  0.00%)
&10  MAIN   MAIN_JP1     -- 00E01E 00E01E -- 00E01E-00E01E
                                              (100.00%)      (  0.00%)
&11  MAIN   MAIN_JP2     -- 00E02E 00E02E --                 00E02E-00E02E
                                              (  0.00%)      (100.00%)
&19  MAIN   MAIN_END     -- 00E032 00E032 --                 00E032 00E032
                                              (  0.00%)      (100.00%)
&3   MAIN   MEMRD        -- 00E038 00E038 --                 00E038 00E038
                                              (  0.00%)      (100.00%)
&12  MAIN   MEMWR        -- 00E03A 00E03A -- 00E03A-00E03A
                                              (100.00%)      (  0.00%)
&14  MAIN   MEMWR_LOP    -- 00E04E 00E04E -- 00E04E-00E04E
                                              (100.00%)      (  0.00%)
&16  MAIN   MEMWR_LOPM   -- 00E054 00E054 -- 00E054-00E054
                                              (100.00%)      (  0.00%)
&15  MAIN   MEMWR_LOPE   -- 00E 58 00E058 -- 00E058-00E058
                                              (100.00%)      (  0.00%)
&2   MAIN   MEMWR_END    -- 00E05C 00E05C --                 00E05C-00E05C
                                              (  0.00%)      (100.00%)
&8   MAIN   CMDIN        -- 00E05E 00E05E -- 00E05E-00E05E
                                              (100.00%)      (  0.00%)
```

NOTE

NOTE

F. HOW TO USE THE PERFORMANCE

F.1 Measuring The Inside of Specified Module

Basically, the performance function consists of the three counters; emulation time counter, measurement time counter, and measurement number of occurrence counter.
The emulation time counter counts the clock pulses of the target processor and measures emulation time.
The measurement number of occurrence counter measures the number of times of measurement from the start to end event point.
Emulation time, total time of executing the specified range, number of times of measuring the specified range, average time of executing the specified range, and processor load factor are displayed as a result.
A use example is shown below.

```
ERX>
ERX>b MAIN.MAIN_END
ERX>pe s=MAIN.MEMWR   LOP e=MAIN.MEMWR_LOPE
ERX>pe
 No.  Module  Symbol      FA Address St Size Data Count EX SEQ B C H P T
&14  MAIN    MEMWR_LOP    -- 00E04E  -- ---- ---- ----  --  -- - - - - S -
&15  MAIN    MEMWR_LOPE   -- 00E058  -- ---- ---- ----  --  -- - - - - E -
ERX>g MAIN.MAIN

D0-7 00000048 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 0000D102 0000D202 00000000 00000000 00000000 00000000 00000000 0000B000
PC = 0000E032    SSP = 0000B000    USP = 00000000     SR = -S7--Z-- = 2704
SP:00E032  MAIN_END         JMP   MAIN_END
<Event "MAIN.MAIN   END" Break>

ERX>pe/d
Total Emulation Time      =   0H: 0M:  0S: 000995US
Number of Event Occurence =                   256
Time  of Event Duration   =   0H: 0M:  0S: 000973US
Average Time              =   0H: 0M:  0S: 000004US
Time Percentage           =                 97.79%
ERX>
```

NOTE

NOTE

G. HOW TO USE THE HLLD FUNCTION

G.1 Outline of High – Level Language Debug (HLLD)

G.1.1 Functions

(1) Source line display (**View.Disassemble**)
(2) Line – unit execution (**T**)
(3) Display of the source line at break time (**Onbreak**)
(4) Real time trace (**History**)
(5) Stack trace (**STK**)

G.1.2 Commands

The commands added for HLLD are listed below.

The command added for HLLD are listed below.

> HLLD	(Macro)	Initializes HLLD
> T	(Macro)	Executes each line
> ONABORT	(Macro)	Performs postprocessing after interrupt of the T command
> MH HLLD	(Macro)	Displays the HLLD command menu
> View	(Command)	Displays the source file
> H/V	(Command)	Source level real time trace
> ONBREAK	(Command)	Defines on break

The commands whose functions have been enhanced for HLLD are listed below.

> H/D	(Command)	Displays real time trace reverse assembly
> DI	(Command)	Displays memory reverse assembly

NOTE

The above two commands display source reverse assembly and source together.

G.2 Examples for Using High – Level Language Debug (HLLD)

The HLLD function is used for the following :

- Program file
- Vecter file
- Character output routine
- Compile patch file
- Link loader command file
- Operating example

NOTE

Program File

```
/* sieve2.c -- scaled down sieve with MAXPRIME_2 instead of 8091 */
/* Eratoshenes Sieve prime number calculation */

#define MAXITER 1
#define MAXPRIME_2 9

        char    flags[MAXPRIME_2];

main ()
{
        register int i,k;
        int     prime,count,iter;

        for (iter = 1;iter(=MAXITER;iter++) {
             count = 0;
             for(i = 0; i(MAXPRIME_2; i++)
                 flags[i] = 1;
             for(i = 0; i(MAXPRIME_2; i++) {
                  if(flags[i]) {
                                prime = i + i + 3;
                                k = i + prime;
                                while (k ( MAXPRIME_2) {
                                      flags[k] = 0;
                                     k+ = prime;
                                }
                                printf(" \n%d",prime);
                                count++;
                   }
              }
        }
        printf(" \n%d primes\n",count);
}
```

NOTE

Vecter File

```
            XREF    ????STACKTOP
VECTOR      IDNT
            ORG     $0
            DC.L    ????STACKTOP
            DC.L    $2000
            END
```

Character Output Routine

```
/*************************************************************************
*                                                                        *
*  char *OUTCHR(c)                                                       *
*                                                                        *
*   outchr output one cher to port 3                                     *
*          use OFA * 2 to direct the output                              *
*                                                                        *
*************************************************************************/

OUTCHR(c)
char c;
{
char *output_port;
            output port = (char *) 0x0000ffff;
           *output port = c;
}
```

NOTE

Compile Patch File

```
ECHO off
REM command file to compile, assemble and link modules of test program
ECHO on
REM compile test program
mcc68k sievex /debug=lines /opt=none;
REM assemble test program, sample startup and level 2 I/O routines
asm68k sievex/d/case;
REM link test program
lod68k @sievex.cmd, sievex.map, sievex.ab
cvt68k sievex.ab sievex.abs
```

Link Loader Command File

```
*****************************************
* linking loader command file for SIEVEX *
*****************************************
*
LIST   C, D, O, P, X, S, T
PUBLIC ????STACKTOP $9000     * Initializes stack pointer (SP)
ORDER  9, 13, 14, 15          * Position sections
sect   9=$2000                * code address
sect   13=$9000               * data address (13:common, 14:data, 15:hea
load   vector
load   entry
LOAD   SIEVEX                 * Test program
load outchr
load csys68k
load inchrw
load sbrk
load\mcc68k\ 68000\mcc68kab.lib
END
```

NOTE

Operating Example

```
demo>on/cl
demo>ed *
demo>rem *** HLLD DEMO
demo>hlld
demo>setup
Setup by CONFIG.CNF File (Yes or <cr> / No) = y
==== Clock Setup ====
Erice : Command parameter error"".
Clock Mode is 1
 0. External (TTL)
 1. Internal (10MHz)
 2. Internal (5MHz)
 3. Internal (2.5MHz)
Select (0-3) ?
==== Map Setup ====
setup68k>  map
SP:000000 - 007FFF = RO
SP:008000 - 00FFFF = RW
SP:010000 - FFFFFF = US
SD:000000 - 007FFF = RO
SD:008000 - 00FFFF = RW
SD:010000 - FFFFFF = US
UP:000000 - 007FFF = RO
UP:008000 - 00FFFF = RW
UP:010000 - FFFFFF = US
UD:000000 - 007FFF = RO
UD:008000 - 00FFFF = RW
UD:010000 - FFFFFF = US
Available Memory Size = 448Kbyte(S)
==== Pin Setup ====
setup68k>  pin
BR*    = High  (Disable)
HALT*  = High  (Disable)
BERR*  = High  (Disable)
RESET* = High  (Disable)
```

NOTE

```
L7     = --  (Disable)
L6     = --  (Disable)
L5     = --  (Disable)
L4     = --  (Disable)
L3     = --  (Disable)
L2     = --  (Disable)
L1     = --  (Disable)
==== Emselect Setup ====
setup68k>  ems
A (Assert AS into monitor)                                = Enable
B (Assert R/W into monitor)                               = Enable
C (Assert DS into emulation memory write cycle)           = Disable
D (Assert DS into emulation memory read cycle)            = Enable
E (Output data bus into emulation memory read cycle)      = Disable
F (Sample BERR into emulation memory access)              = Disable
G (Sample DTACK into emulation memory access)             = Disable
H (Generate fault break into emulation)                   = Enable
I (Generata auto wait into all memory access)             = Disable
J (Generata timeout into all memory access)               = Enable
K (Number of auto wait clock [1-7])                       = 2clock
L (Number of timeout clock [8, 128, 2K, 4K, 8K, 16K, 32K]) = 32Kclock
==== Object Setup ====
Load Object File Name= = sievex
Loading Object File "sievex.ABS"
Completed Loading.  Start Address: SP:002000
==== Other Setup commands ====
demo>es
 No.   Module  Symbol    FA Address St Size Data Count EX SEQ B C H P T
&1    ENTRY   ENTRY     -- 002000  -- ---- ---- ----  --  -- - - - - -
&2    ENTRY   LSTOP     -- 002020  -- ---- ---- ----  --  -- - - - - -
&15   sievex  .main     -- 002028  -- ---- ---- ----  --  -- - - - - -
&9    sievex  #14       -- 002034  -- ---- ---- ----  --  -- - - - - -
&10   sievex  #15       -- 00203A  -- ---- ---- ----  --  -- - - - - -
&11   sievex  #16       -- 00203E  -- ---- ---- ----  --  -- - - - - -
&12   sievex  #17       -- 002042  -- ---- ---- ----  --  -- - - - - -
&13   sievex  #18       -- 002074  -- ---- ---- ----  --  -- - - - - -
&14   sievex  #19       -- 002078  -- ---- ---- ----  --  -- - - - - -
```

NOTE

```
&3   sievex  #20          -- 002098  -- ---- ---- ----  -- --  - - - - -
&4   sievex  #21          -- 0020AE  -- ---- ---- ----  -- --  - - - - -
&5   sievex  #23          -- 0020BE  -- ---- ---- ----  -- --  - - - - -
&6   sievex  #24          -- 0020DA  -- ---- ---- ----  -- --  - - - - -
&7   sievex  #26          -- 0020F6  -- ---- ---- ----  -- --  - - - - -
&8   sievex  #27          -- 002106  -- ---- ---- ----  -- --  - - - - -
&16  sievex  #31          -- 002138  -- ---- ---- ----  -- --  - - - - -
&18  sievex  #32          -- 002148  -- ---- ---- ----  -- --  - - - - -
&22  outchr  .OUTCHR      -- 00214E  -- ---- ---- ----  -- --  - - - - -
&19  outchr  #14          -- 002158  -- ---- ---- ----  -- --  - - - - -
&20  outchr  #15          -- 002160  -- ---- ---- ----  -- --  - - - - -
&21  outchr  #16          -- 002168  -- ---- ---- ----  -- --  - - - - -
&23  inchrw  #13          -- 0023DE  -- ---- ---- ----  -- --  - - - - -
 No.  Module  Symbol      FA Address St Size Data Count EX SEQ B C H P T
&26  inchrw  .INCHRW      -- 0023DE  -- ---- ---- ----  -- --  - - - - -
&24  inchrw  #14          -- 0023E2  -- ---- ---- ----  -- --  - - - - -
&25  inchrw  #15          -- 0023E4  -- ---- ---- ----  -- --  - - - - -
&27  SBRK    .sbrk        -- 0023EA  -- ---- ---- ----  -- --  - - - - -
&17  sievex  .flags       -- 009122  -- ---- ---- ----  -- --  - - - - -
demo>di
SP:002000 2E7C00009000    ENTRY          MOVEA.L #$00009000,A7
SP:002006 41F9000094AE                   LEA.L  $0094AE.L,A0
SP:00200C 5888                           ADDQ.L  #4,A0
SP:00200E 23C8000094AE                   MOVE.L  A0,$0094AE.L
SP:002014 2C7C00000000                   MOVEA.L #$00000000,A6
SP:00201A 4EB90000216C                   JSR    $00216C.L
SP:002020 4E722700        LSTOP          STOP    #$2700
SP:002024 60FA                           BRA.S  LSTOP
SP:002026 4E75                           RTS
SP:002028 4E56FFD8        .main          LINK    A6,$#FFD8
demo>di sievex..main
SP:002028 4E56FFD8        .main          LINK    A6,#$FFD8
SP:00202C 2D7C00009122FFE6               MOVE.L  #$00009122,$FFE6(A6)
sievex #14 :        for (iter = 1;iter(=MAXITER;iter++) {
SP:002034 7001            #14            MOVEQ.L #$01,D0
SP:002036 2D40FFE2                       MOVE.L  D0,$FFE2(A6)
```

NOTE

```
sievex #15 :          count = 0;
SP:00203A 42AEFFDA          #15          CLR.L  $FFDA(A6)
sievex #16 :          for(i = 0; i<MAXPRIME_2; i++)
SP:00203E 42AEFFF4          #16          CLR.L  $FFF4(A6)
sievex #17 :          flags[i] = 1;
SP:002042 3D6EFFF6FFEA      #17          MOVE.W  $FFF6(A6),$FFEA(A6)
SP:002048 206EFFE6                       MOVEA.L $FFE6(A6),A0
SP:00204C 43D0                           LEA.L  (A0),A1
SP:00204E 2D49FFFC                       MOVE.L  A1,$FFFC(A6)
demo>defm sievex
demo>b #20
demo>g/w
sievex #20 :                    prime = i + i + 3;
demo>h/v
Point  Executed Source Line
 0469
sievex #14 :       for (iter = 1;iter<=MAXITER;iter++) {
 0464
sievex #15 :          count = 0;
 0460
sievex #16 :          for(i = 0; i<MAXPRIME_2; i++)
 0456
sievex #17 :              flags[i] = 1;
 0408
sievex #17 :              flags[i] = 1;
 0362
sievex #17 :              flags[i] = 1;
 0316
sievex #17 :              flags[i] = 1;
 0270
sievex #17 :              flags[i] = 1;
 0224
sievex #17 :              flags[i] = 1;
 0178
sievex #17 :              flags[i] = 1;
 0132
sievex #17 :              flags[i] = 1;
```

NOTE

```
sievex #17 :          flags[i] = 1;
 0041
sievex #18 :       for(i = 0; i<MAXPRIME_2; i++) {
 0039
sievex #19 :                if(flags[i]) {
 0011
sievex #20 :                          prime = i + i + 3;
demo>h/d 44
Point E FA  Addr  Data St                  Opcode Operand
 0044   SP 002074        #18               BGT.S  #17
 0043  SD 008FB0 0000 MR
 0042  SD 008FB2 0009 MR
sievex #18 :           for(i = 0; i<MAXPRIME_2; i++) {
 0041  SP 002074          #18              CLR.L  $FFF4(A6)
 0037  SD 008FB2 0000 MR
 0036  SD 008FB0 0000 MW
sievex #19 :                if(fiags[i]) {
 0039  SP 002078          #19              MOVEA.W $FFE6(A6), $FFEC(A6)
 0034  SD 008FB2 0000 MR
 0032  SD 008FA8 0000 MW
 0033  SP 00207E                           MOVEA.L $FFFC(A6), A0
 0030  SP 002082                           LEA.L  (A0), A1
 0029  SD 008FA2 0000 MR
 0028  SD 008FA4 9122 MR
 0027  SP 002084                           MOVE.L A1, $FFFC(A6)
 0024  SD 008FB8 0000 MW
 0023  SD 008FBA 9122 MW
 0025  SP 002088                           MOVEA.L $FFFC(A6), A0
 0020  SD 008FB8 0000 MR
 0019  SD 008FBA 9122 MR
 0021  SP 00208C                           MOVE.W $FFEC(A6), D0
 0016  SD 008FA8 0000 MR
 0017  SP 002090                           TST.B  $00(A0, D0.W)
 0013  SD 009122  01 MR  .flags
 0014  SP 002094                           BEQ.L  $00210C
```

NOTE

```
sievex #20 :                                prime = i + i + 3;
 0011 > SP 002098          #20                MOVE.L $FFF4(A6),$FFDE(A6)
 0008 > SD 008FB0 0000 MR
 0007 > SD 008FB2 0000 MR
 0005 > SD 008F9A 0000 MW
 0004 > SD 008F9C 0000 MW
 0006 > SP 00209E                             MOVE.L $FFDE(A6),D0
 0002 > IA FFFFFE FFDE IA
 0001  SD 008F90 209E MW
 **** Pause ****
demo>v
sievex #20 :                                prime = i + i + 3;
sievex #21 :                                k = i + prime;
sievex #22 :                                while (K < MAXPRIME __2) {
sievex #23 :                                      flags[k] = 0;
sievex #24 :                                      k+ = prime;
sievex #25 :                                }
sievex #26 :                                printf("\n%d",prime) ;
sievex #27 :                                count++;
sievex #28 :                  }
sievex #29 :           }
demo>t 20
~sievex.#21
sievex #21 :                                k = i + prime;
~sievex.#23
~sievex.#26
sievex #23 :                                      flags[k] = 0;
sievex #24 :                                      k+ = prime;
~sievex.#26
sievex #23 :                                      flags[k] = 0;
sievex #24 :                                      k+ = prime;
sievex #26 :                                printf("\n%d",prime) ;
~outchr.#14
~outchr.#16
~outchr.#14
~outchr.#16
demo>ed outchr.*
demo>on ~*.* "g/w"
```

NOTE

```
demo>t 20
sievex #27 :                        count++;
sievex #19 :                if(flags[i]) {
sievex #20 :                        prime = i + i + 3;
sievex #21 :                        k = i + prime;
sievex #23 :                            flags[k] = 0;
sievex #24 :                            K+ = prime;
sievex #26 :                        printf("\n%d", prime);
sievex #27 :                        count++;
sievex #19 :                if(flags[i]) {
sievex #20 :                        prime = i + i + 3;
sievex #21 :                        k = i + prime;
sievex #26 :                        printf("\n%d", prime);
sievex #27 :                        count++;
sievex #19 :                if(flags[i]) {
sievex #19 :                if(flags[i]) {
sievex #20 :                        prime = i + i + 3;
sievex #21 :                        k = i + prime;
sievex #26 :                        printf("\n%d", prime);
sievex #27 :                        count++;
sievex #19 :                if(flags[i]) {
demo>rem *** covarage demo
demo>ev stack a=8000 9000
demo>cov *=on
demo>cov/cl
demo>r reset
demo>g/w
sievex #20 :                        prime = i + i + 3;
demo>cov
 No.  Module  Symbol      FA  Start  End    St P/U   Cal
&1    ENTRY   ENTRY       -- 002000 002000 --  *  100.00%
&2    ENTRY   LSTOP       -- 002020 002020 --  -    0.00%
&15   sievex  .main       -- 002028 002028 --  *  100.00%
&9    sievex  #14         -- 002034 002034 --  *  100.00%
&10   sievex  #15         -- 00203A 00203A --  *  100.00%
&11   sievex  #16         -- 00203E 00203E --  *  100.00%
&12   sievex  #17         -- 002042 002042 --  *  100.00%
```

NOTE

```
&13  sievex  #18          -- 002074 002074 --  *  100.00%
&14  sievex  #19          -- 002078 002078 --  *  100.00%
&3   sievex  #20          -- 002098 002098 --  *  100.00%
&4   sievex  #21          -- 0020AE 0020AE --  -    0.00%
&5   sievex  #23          -- 0020BE 0020BE --  -    0.00%
&6   sievex  #24          -- 0020DA 0020DA --  -    0.00%
&7   sievex  #26          -- 0020F6 0020F6 --  -    0.00%
&8   sievex  #27          -- 002106 002106 --  -    0.00%
&16  sievex  #31          -- 002138 002138 --  -    0.00%
&18  sievex  #32          -- 002148 002148 --  -    0.00%
&23  inchrw  #13          -- 0023DE 0023DE --  -    0.00%
&23  inchrw  .INCHRW      -- 0023DE 0023DE --  -    0.00%
&24  inchrw  #14          -- 0023E2 0023E2 --  -    0.00%
&25  inchrw  #15          -- 0023E4 0023E4 --  -    0.00%
&27  SBRX    .sbrk        -- 0023EA 0023EA --  *  100.00%
&30  sievex  stack        -- 008000 009000 --  -    3.44%
&17  sievex  .flags       -- 009122 009122 --  *  100.00%
demo>cov/d stack
 No.  Module  Symbol     FA  Start  End   St     Pass        Unpass
&30   sievex  stack      -- 008000 009000 --              008000-008F73
                                                 008F74-009000
                                                   ( 3.44%)     ( 96.55%)
demo>
```

NOTE