Gould Millennium
9508S Microsystem Emulator

Product Description

GOULD

Electronics & Electrical Products

There Is One Fast Way to Develop a Microprocessor-Based System

Gould Millennium Invented It!

You probably realize that you must use in-circuit emulation when you're developing systems around microprocessors. It's the only way that you can look around inside the system. But there are lots of difficult choices involved in selecting the ICE ™ that you need.

Opaque emulation. Almost every manufacturer of in-circuit emulators advertises "fully transparent" emulation. So you need to know the right questions to ask. Is it functionally *and* electrically transparent? Does it support all the operating modes of your target microprocessor? Does it insert artificial wait states? Does it use any of the target system address space? Does it operate at full microprocessor clock speeds? Does it provide the proper input and output impedances? If your emulation isn't transparent, you may spend more time developing an emulator-based system than a microprocessor-based system.

™ *Trademark of Intel Corporation.*

Overbundled system. With many development systems, you must tie up $25,000 worth of computer to use a $5,000 option. Programmers can't edit or assemble code while engineers use the ICE option or vice versa. System development consists of software development (coding, editing, assembly/compilation, debugging), proceeding in parallel with hardware development (designing, prototyping, debugging). The programmers need the development system's text editor, file manager, assemblers, and compilers. The programmers and engineers both need ICE. They should be able to use them separately.

Dedicated Systems. Some development systems are excellent in many respects except that they support the offerings of only one microprocessor manufacturer. What if you want to use one microprocessor as a number cruncher and another as an I/O controller? You could end up with two imcompatible development systems and ICE units if you're not careful. Your development systems and ICE should be able to support all of the microprocessors that you are using now and those you may use in the future. These development systems are called "universal", and we invented the concept.

The Gould Millennium Fast, Efficient Way to System Development

Gould Millennium invented the fast, efficient way to develop microprocessor-based systems, and we have been improving it for years. Our development system philosophy is based on eight years experience in the design and development of microprocessor-based systems. We have developed more than thirty of them, so we understand the design engineer's problems.

Gould Millennium was the first company to produce a universal development system. That was in 1976, and since then we have manufactured and shipped more of them than any other company.

Our previous development systems were sold under other names (names such as Tektronix, Signetics, and Motorola). Our experience with development system design and microprocessor-based product design has taught us just about everything there is to know about development systems. The 9500 family is the result of that experience.

Your interface to a development system is the system's command set. Our engineers improved on the command sets from previous systems. Then they added commands that were lacking. The result is a powerful, 'friendly' command set that is easy to use. You can concentrate on your development task, not on how to use your development system.

9508S Microsystem Emulator. A standalone in-circuit emulator which supports all of the popular 8-bit microprocessors and microcomputers. The 9508S Microsystem Emulator is our second-generation of 8-bit ICE. We have expanded the features available in the 9508; more memory, more sophisticated and versatile host communications, higher speed communications, service requests to and from the console, and command file capability have been added to our high quality emulation to make it even more powerful and easy to use. It is the subject of this Product Description.

Figure 1.

9516 Microsystem Integration Station: a more sophisticated stand-alone in-circuit emulator which will support 16-bit microprocessors as well as 8-bit microprocessors.

9520 Software Development System: a multi-tasking software development system which starts as a single-user system and is upgradable to a dual-user system.

9540 Software Development System: provides multi-user support via the Unix operating system with hard disk.

The Gould Millennium 9500 family of development systems helps you avoid those difficult choices by providing all the development capability you will ever need. Here's why the 9508S Microsystem Emulator is your best choice.

Ultimate Performance. Gould Millennium starts with the best in-circuit emulators available anywhere. They are totally transparent to the system under development. They support all of the operating modes of the target microprocessors. They don't intrude on the target microprocessor's memory or I/O addressing space. They operate at the maximum clock rate of the target microprocessor. They don't introduce artificial wait states.

We start with solid emulation performance. Then we add powerful debug features like real-time trace, complex hardware breakpoints, in-line and mnemonic display. But the important thing is, if your system under development works with the 9508S, it will work with the target microprocessor, and if it works with the target microprocessor, our emulators won't hide the problem.

Unbundled Functions. The 9508S is a stand-alone in-circuit emulator. It doesn't need a host development system to perform all of its intended functions. Its real-time trace, in-line assembler/disassembler, complex breakpoints, memory mapping, register display/modification, and debug command set are all self-contained. Software debug and hardware development are unbundled from software generation.

Your 9520 Software Development System (or your host mini or MDS) does the software generation. Download it to the 9508S Microsystem Emulator over a standard RS-232 serial link. Then break the link. The 9520 Software Development System is free to do its job. The 9508S Microsystem Emulator takes over the jobs of software debug and hardware debug. It's like having two complete development systems.

Universal Support. Gould Millennium invented universal development systems, and the 9508S Microsystem Emulator is another step in that concept. It supports all of the popular 8-bit microprocessors and microcomputers.

8021	8048	6800A
8035	8748-4	6801
8035-4	8748-8	6802
8035-8	8049	6803
8039	8049-6	6808
8039-6	8050	6809
8040	8080A	68A09
8041A	8085A-2	68B09
8741A		6809E
8741-6	Z80A	68A09E
		68B09E

The Right Price. The 9508S Microsystem Emulator is an in-circuit emulator that can be used as a stand-alone unit, as a terminal on the 9520 Software Development System, or as an add-on terminal to your existing minicomputer or development system. Its price is competitive, but its real economy is in the flexibility that it offers. You buy just the power you need. You don't scrap your existing development system or computer. When purchased with the 9520 Software Development System, it provides the highest possible productivity and development capability for the price.

Ultimate performance. Unbundled functions. Universal support. Reasonable price. These are the foundations upon which the 9508S Microsystem Emulator is built.

The 9508S Microsystem Emulator

Ultimate Emulation Performance Coupled With Debugging Tools that Speed the Development of Your Microprocessor-Based System

The 9508S consists of a main unit containing common control circuits, real-time trace module, emulation memory, power supply, and the microprocessor in-circuit emulation module. The ICE module is connected to an emulation pod with probe.

A video terminal is connected to the main unit through an RS-232 serial port. A second RS-232 port allows the 9508S to be connected to the 9520 Software Development System or other host computer. Several serial protocols are supported for communications up to 19.2K baud.

Compatible host systems, other than the 9520, include other development systems such as those manufactured by Intel, Motorola, and Zilog, and minicomputers such as those manufactured by Digital Equipment, Data General, and Hewlett-Packard. A sophisticated communications interface supports a variety of protocols and gives you software control of several communications parameters.

Figure 2.

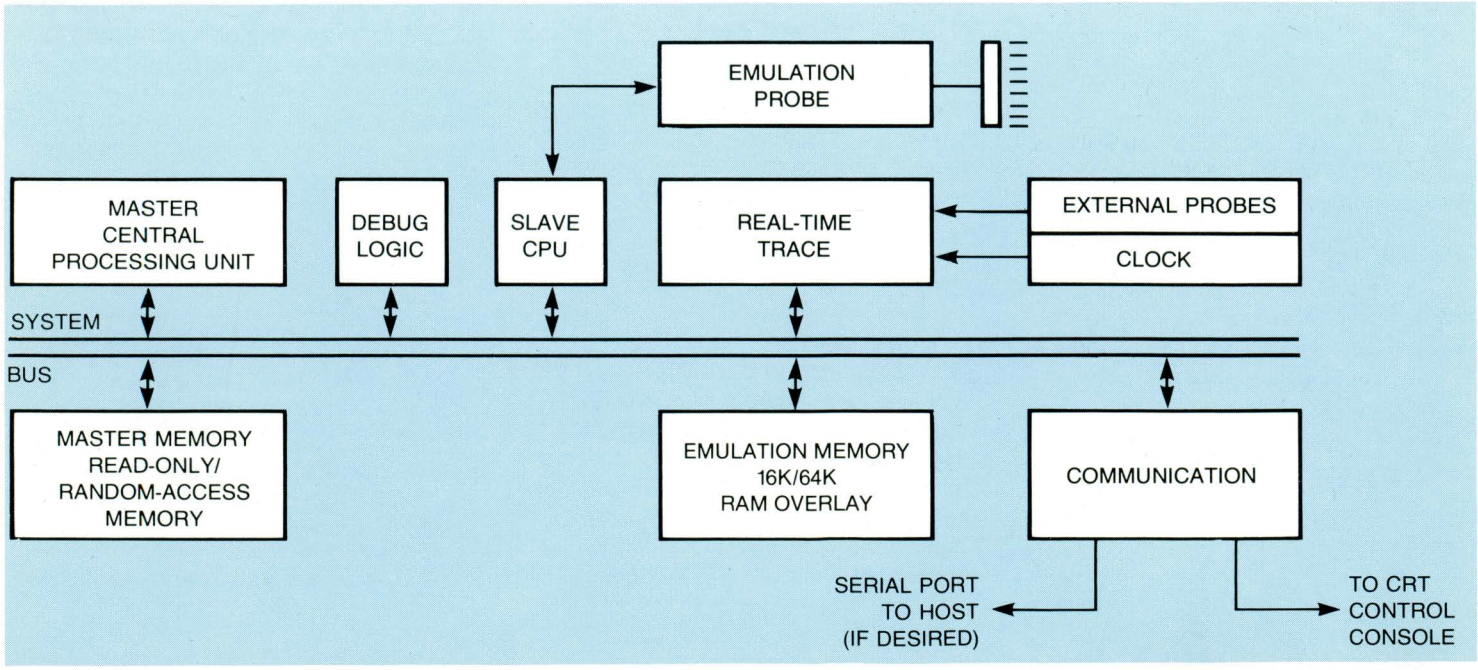


Figure 3. The Gould Millennium 9508S Block Diagram.

Powerful Debug Features Make It Possible

Most operations, however, will be performed in a stand-alone mode. When you are developing hardware, debugging software, or integrating hardware and software, the 9508S provides all the tools you need.

When you are developing software, you will assemble your programs on a host system and download the object code to the 9508S for execution and debug. You can break the link once the download is done, because the host system is needed only for program editing, assembly, and storage.

All of the program execution is accomplished in the 9508S. Your 9520 Software Development System or host minicomputer is free for other parallel activities.

If you want to save partially debugged software, just re-establish the link and upload it to the 9520. When you are ready to continue debugging, download it again and continue where you left off.

Emulation Memory

During software debug and the early stages of system integration, your programs will usually be loaded into the 9508S emulation memory for execution and debug.

Emulation memory is available in 16K and 64K modules. Emulation memory is totally separate from 9508S master ROM and RAM. The 9508S master memory does not intrude on the target microprocessor's addressing space.

Segments of the emulation memory may be mapped into the target microprocessor's address space. Mappable segments may be from 256 bytes to 64K bytes. This allows software modules which are being debugged to interact with previously debugged modules which are resident in the prototype hardware. But no matter where the software is that is being debugged, it is under the control of the 9508S.

Memory Modification and Display

Powerful memory monitoring and control commands are available to control your program and data memory regardless of where it is located.

Base Address Registers enable you to specify up to four base addresses and then address memory locations relative to them. This is especially useful when you are working with relocatable programs since linked modules will seldom be loaded at the addresses shown on your program listings.

A base address register lets you set the actual starting address of a relocatable module in the base address register and then address memory using the addresses contained in your program listings, offset by the base address. You don't have to perform hexadecimal addition or subtraction with all its chances for errors.

In-Line Disassembler converts machine language programs to assembly language mnemonics. This makes your displayed program segments from memory look like your program listings, so you don't have to relate hexadecimal numbers to microprocessor instructions; potential errors are reduced and debugging is speeded up.

In-Line Assembler provides a direct translation from assembly language mnemonics to machine language code. You can patch your programs in assembly language rather than having to translate to machine language manually. This eliminates another potential source of errors, and reduces the amount of time that you need to reassemble or recompile your program on the 9520 Software Development System.

This is a line-by-line assembler which recognizes the mnemonics of the target microprocessor. It's not a powerful macro assembler such as the equivalent assembler on the 9520, but it will translate one instruction at a time and put it at the memory location that you specify. It's a real debugging time-saver and mistake-avoider.

Memory Display Commands provide simple, precise methods for displaying the contents of memory. One command, DISM, displays memory in assembly language mnemonics. The DUMP command displays memory in hexadecimal format with ASCII equivalents at the end of each display line.

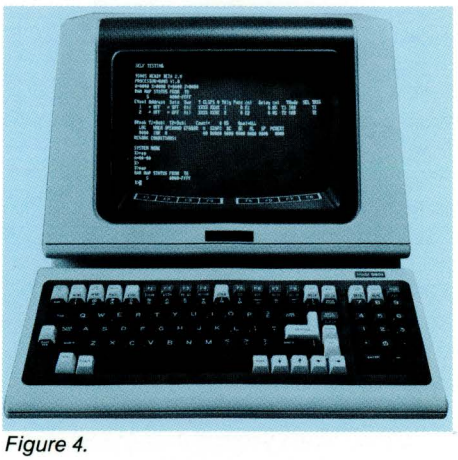


Figure 4.

Memory Modification Commands allow you to change memory quickly and accurately. The ASM command is used to make changes in assembly language mnemonics. The EXAM command lets you examine one memory location at a time and either modify it or leave it unchanged. The PATCH command lets you modify contiguous memory locations (up to 64 bytes) with an ASCII or hexadecimal string. The FILL command lets you load a range of memory with an ASCII or hexadecimal string; for example, loading all of unused memory with HALT instructions so that program execution will stop if your program jumps outside of its intended bounds. The MOVE command is used to move blocks of memory from one location to another.

Register Modification and Display

In all cases you can display and modify any register which can be accessed by your program. You can display all registers with a single command on your video terminal. The REG command is used to control register display and modification.

Service Requests

The ability to transmit characters or messages to and from the console is supplied by the service request command, SVC. SVC allows you to utilize the system console from within your user program.

Real-Time Trace

Next to ICE itself, real-time trace is probably the most powerful tool available to designers of microprocessor-based systems. It is used in all phases of system development: in hardware development when you want to see what's happening between your microprocessor and your complex peripherals; in software debug when you need to trace program execution paths to determine where a program is coming from and where it is going; in system integration when you must trace the interaction of unproven hardware and unproven software.

The 9508S real-time trace has a high speed memory which records the last 128 events on the target microprocessor's bus and on the eight external data probes. It can be qualified to record all events or only specific types of events such as instruction fetches or memory writes.

Real-time trace is used in conjunction with a powerful hardware breakpoint capability. The breakpoint logic is used to stop program execution in the area in which you are interested, and the real-time trace buffer lets you see what happened there.

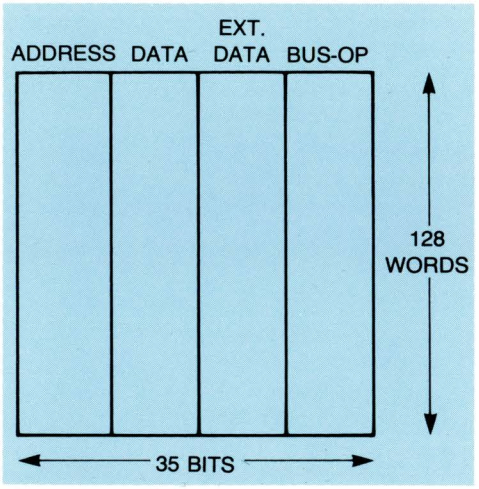


Figure 5.

Here are some specifics about the 9508S real-time trace capabilities.

Trace Buffer is 128 locations deep and 35 bits wide. Information recorded in each location consists of 16 bits of address, 8 bits of data, 8 bits of external probe data, and bus transaction type. Trace buffer stores can be qualified so that only specific types of transactions will be recorded. You can select Memory Read, Memory Write, all Memory, I/O Read, I/O Write, all I/O, Instruction Fetch cycles, all Reads, or all Writes.

External Probes consist of 8 data probes, a clock probe, and a ground probe. They can be used to monitor I/O ports during program execution, external status indicators, or anything else of a digital nature which may be going on outside of the target microprocessor. They can be recorded synchronously by the microprocessor clock or sampled by an asynchronous clock for storage at microprocessor clock time. There is also a latch mode for detecting multiple transitions occurring between microprocessor clocks. They can also be included in the breakpoint equations, so that breaks and triggers can be conditioned by external events as well as internal bus transactions.

Breakpoints

The 9508S Microsystem Emulator has four breakpoints:

One simple hardware breakpoint on addresses. (See GO command on page 16.)

One software breakpoint on microprocessor register contents. (See REGBRK command on page 16.)

Two complex hardware breakpoints.

The complex hardware breakpoints provide the greatest flexibility and debugging power and will be covered in considerable detail in the following paragraphs.

There are two trigger/breakpoint circuits which can operate independently to provide two separate trigger/breakpoint conditions, or they may work together to multiply their power for diagnosing complex problems. Operating modes are Independent, Limit, Arm, and Freeze.

Independent Mode. This mode illustrates the basic functions of the two circuits. Each operates independently and identically. The accompanying flow chart illustrates the Independent Mode. Each flow chart block, representing a circuit feature, provides significant debugging power.

- ***Event Specification.*** The first step in using breakpoints is to set up an event comparison register. Each of the two breakpoint circuits has one of these registers. When setting it up, you will make the following selections.

  ADDRESS. Set an address value, and select comparison on EQUAL, LESS THAN/EQUAL, or GREATER THAN/EQUAL. Address range is 16 bits.

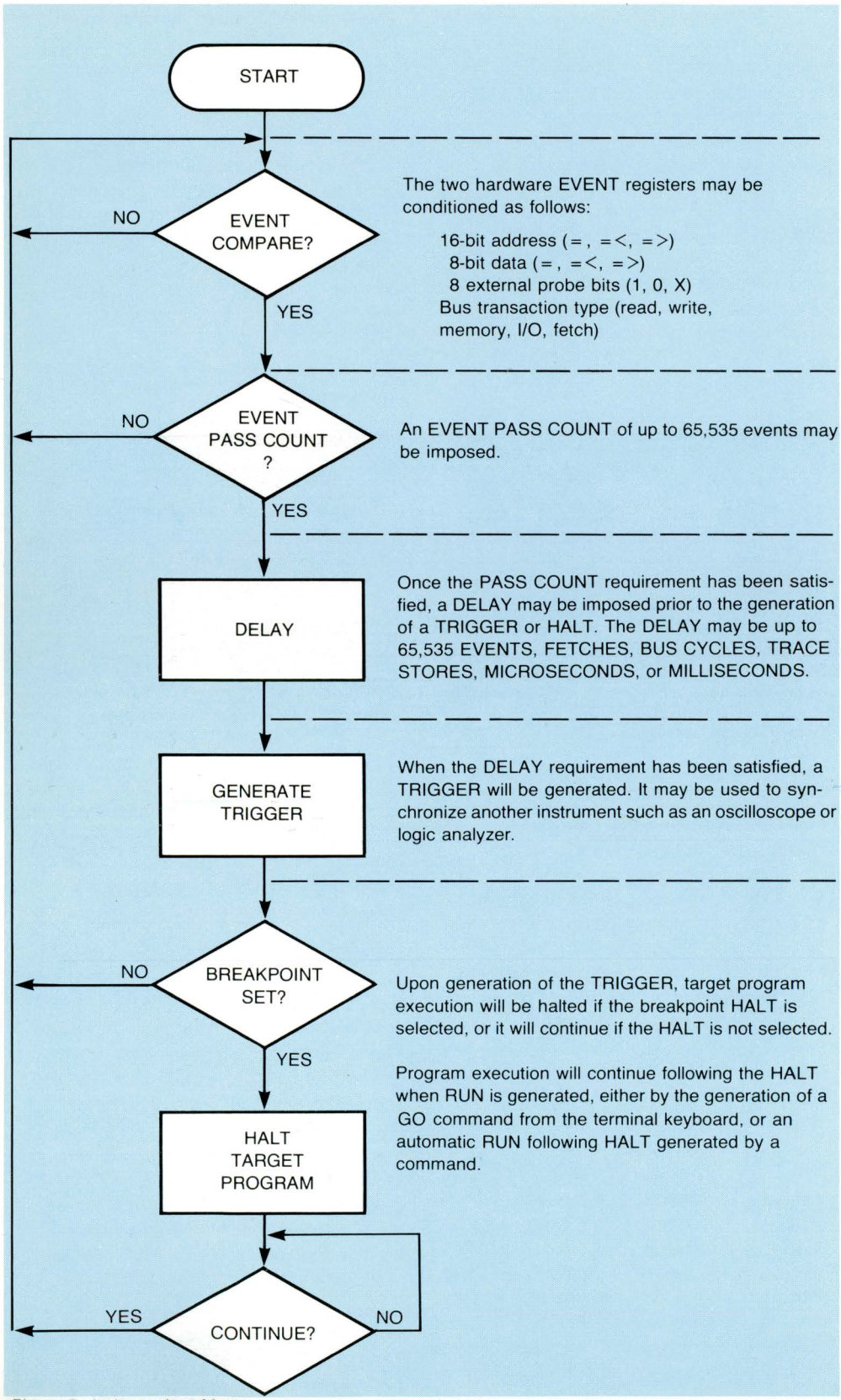


Figure 6. Independent Mode.

DATA. This 8-bit data selection allows such combinations as breaking on a specific value being written to a selected memory location or I/O port.

EXTERNAL DATA. The external data probes that are used to capture data for real-time trace may also be included in the breakpoint equation. Each bit may be selected as "1", "0", or "X" for don't care. You could, for example, include a single status bit in your breakpoint equation and ingore all other conditions. You could break on the generation of an interrupt, the acknowledge signal from a peripheral, or the lack of an acknowledge signal from a peripheral.

BUS TRANSACTION. The final event qualifier is the target microprocessor bus transaction type. You may select Memory Read, Memory Write, All Memory, I/O Read, I/O Write, All I/O, Instruction Fetch cycles, all Reads, or all Writes.

- ***Pass Count.*** An event pass count of up to 65,535 events may be selected. This means that the Event comparison logic must detect the satisfaction of the event conditions the number of times called for in the pass count selection. This is especially useful when you are debugging program loops, and you want to see what happens when you exit from the loop.
- ***Delay.*** A delay of up to 65,535 counts may be selected in addition to the Pass Count described above. When you select the Delay feature, you can specify the delay "clock". It can be Event comparisons, Instruction fetches, Trace buffer stores, Bus cycles, Milliseconds, or Microseconds.

  One of the most common uses for the Delay feature is to trace events *following* the satisfaction of the Event conditions. For example, a delay of 64 Trace buffer stores would center the condition that satisfied the Event comparison in the Trace buffer. The Trace buffer would then contain the 64 transactions preceding the event and the 64 following it.

- ***Trigger.*** A Trigger Pulse will automatically be generated whenever the Event, Pass Count, and Delay conditions are satisfied. You can use it to trigger another instrument such as a logic analyzer or oscilloscope, or you can use it to trigger an event in your prototype system. You could use it, for example, to generate an interrupt during a particular part of your program, and then examine the trace buffer to see if the interrupt was processed properly.

- **Breakpoint Halt.** If you want to examine machine status upon satisfying all Event, Pass Count, and Delay conditions, you can specify a Breakpoint Halt. Once the target microprocessor is halted, you can examine the Trace Buffer, microprocessor regsiters and status, and memory contents. You have the options of modifying the progam, modifying target microprocessor registers, loading a new program, or any other debugging tasks. You can continue executing your program from the point of the Halt or from any other point you may choose.

Limit Mode. The Limit Mode allows you to set up breakpoints or triggers within a range of addresses or target microprocessor conditions. As you can see from the flow chart, EVENT 1 *and* EVENT 2 conditions must be satisfied *simultaneously* in order for a trigger or breakpoint halt to occur on the EVENT 1 path. The EVENT 2 path still functions the same as in the Independent Mode, but a breakpoint halt would not normally be selected on this side in the Limit Mode.

This mode is used when you want to set a breakpoint within a range of values. For example, if your program is attempting to write into a protected area of memory, you can set the EVENT 1 address at the upper end of the range with a comparison on LESS THAN/EQUAL, and a bus transaction qualifier of MEMORY WRITE. The EVENT 2 address would be set at the lower end of the range with GREATER THAN/EQUAL comparison selected. When both events occur simultaneously, your program is attempting to write into the protected area. If you select a breakpoint Halt, you can then examine the Trace Buffer and see how your program got there.

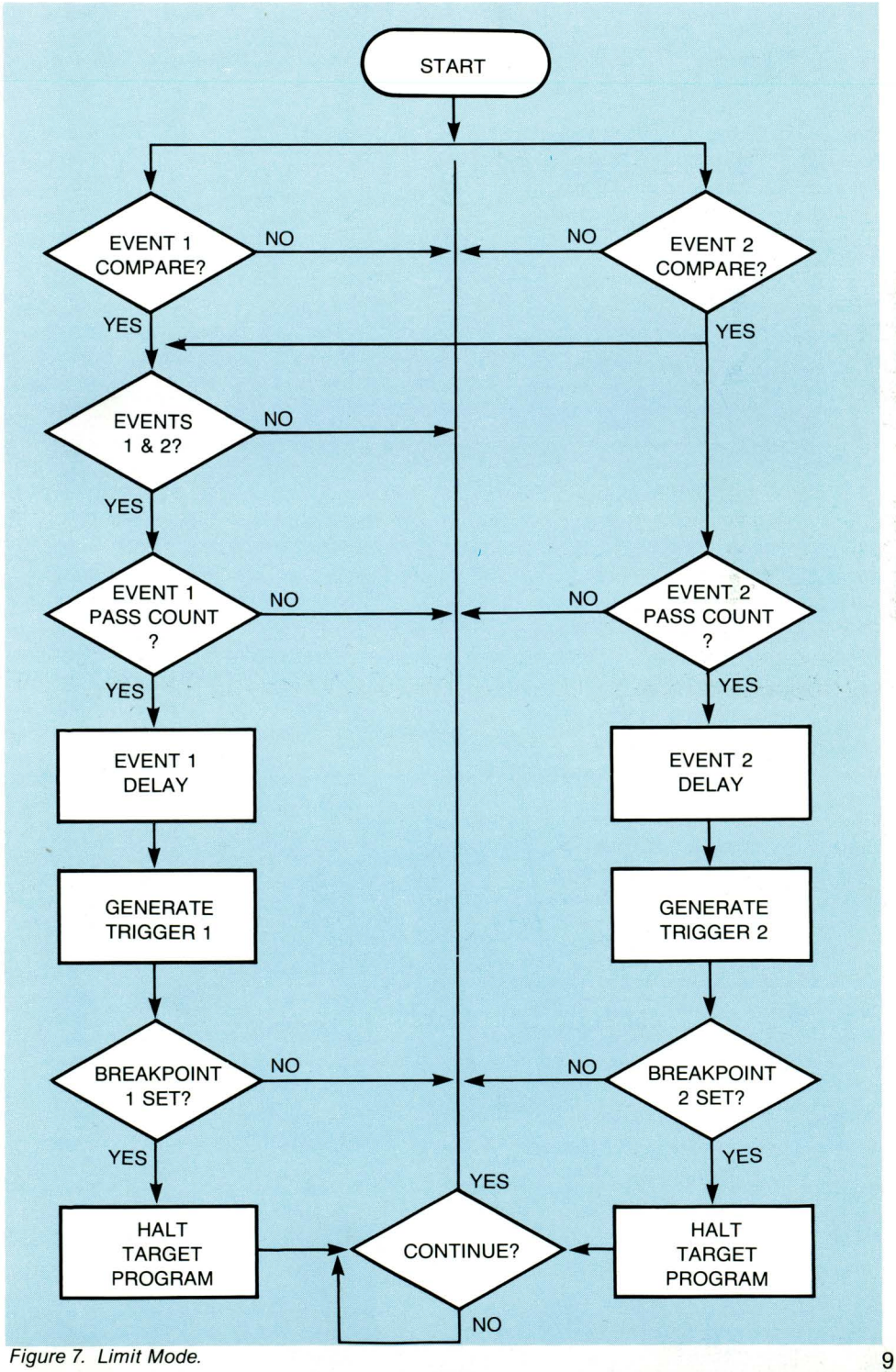


Figure 7. Limit Mode.

Arm Mode. The Arm Mode is used when one event is to be recognized *after* another event has occurred. (In the Limit Mode, the events must occur simultaneously; in the Arm Mode they must occur sequentially.) This is a powerful tool which can help you diagnose some very complex problems.

One example is the interaction of subroutines with the main program. A problem could occur in a subroutine when it is called from one point in your main program, but not when it is called from others. Set EVENT 1 to an appropriate point in the calling routine. You can even specify a Pass Count (the flow chart shows the sequence).

When the EVENT 1 comparisons and Pass Count are satisfied, TRIGGER 1 will be generated and will enable the EVENT 2 path and disable the EVENT 1 path. The EVENT 2 path will then function as if it were functioning independently.

If a Breakpoint Halt is selected in the EVENT 2 path, your program will halt with the Trace Buffer containing a record of events in the vicinity of EVENT 2. You can include a DELAY in the EVENT 2 path in order to see bus transactions following EVENT 2.

Freeze Mode. The Freeze Mode is identical to the Arm Mode except that the contents of the Trace Buffer are frozen following the satisfaction of TRIGGER 1.

The Freeze Mode is used when some operation must be allowed to run to its conclusion, but you want to examine something that happened in real-time prior to that conclusion. Set EVENT 1 at the end of the segment which you want to examine and EVENT 2 at the conclusion of the operation. Select a Breakpoint Halt. When the Breakpoint Halt is encountered, the Trace Buffer will contain a history of bus transactions immediately preceding EVENT 1 rather than EVENT 2.

The Freeze Mode can be particularly useful when debugging microprocessor-based controllers where external mechanical operations are involved. EVENT 1 could be the routine which initiates the operation, and EVENT 2 could be the signal that the external operation is done. You can examine a history of the initiating routine without stopping the external mechanical operations.

Breakpoint Command File

One significant feature of the 9508S is its ability to execute a user defined set of commands whenever a Breakpoint is encountered. The Breakpoint Command File may occupy up to 128 bytes of 9508S master RAM (NOT emulation RAM).

This feature may be used, for example, to automatically display the contents of the Trace Buffer or an area of memory, and then restart the program. You could have your program running in a tight loop with a constant update of your Trace Buffer display at the end of each pass through the loop. Any command which may be executed when the 9508S is in Command Mode may be included in the Breakpoint Command File.

The command sequence for this example is simple. Assume that your program segment starts at location 1000, and your Breakpoint Halt is set at some point after that. In order to automatically display the Trace Buffer and restart program execution at location 1000 again, your command sequence would look like this:

ONBRK	Defines the command sequence.
DRT	Display Trace Buffer.
GO 1000	Go to 1000 and resume execution.
(CARRIAGE RETURN)	End of "ON BREAK" sequence.

With this command sequence the contents of the Trace Buffer will be displayed every time the Breakpoint Halt is encountered, and execution will automatically be resumed at location 1000. You will have a dynamic history of program execution displayed on your video terminal.

Command Files

Command files provide a more general purpose macro capability. They are designed to make identical setups easier to repeat, commonly used sequences easier to use, and especially when used in conjunction with ONBRK and SVC to provide a level of automatic test execution. Command files may occupy up to 128 bytes of 9508S master RAM (NOT emulator RAM). They can be entered manually from the console or can be downloaded from any host machine. A command file can contain any 9508S command except for the CFILE command itself.

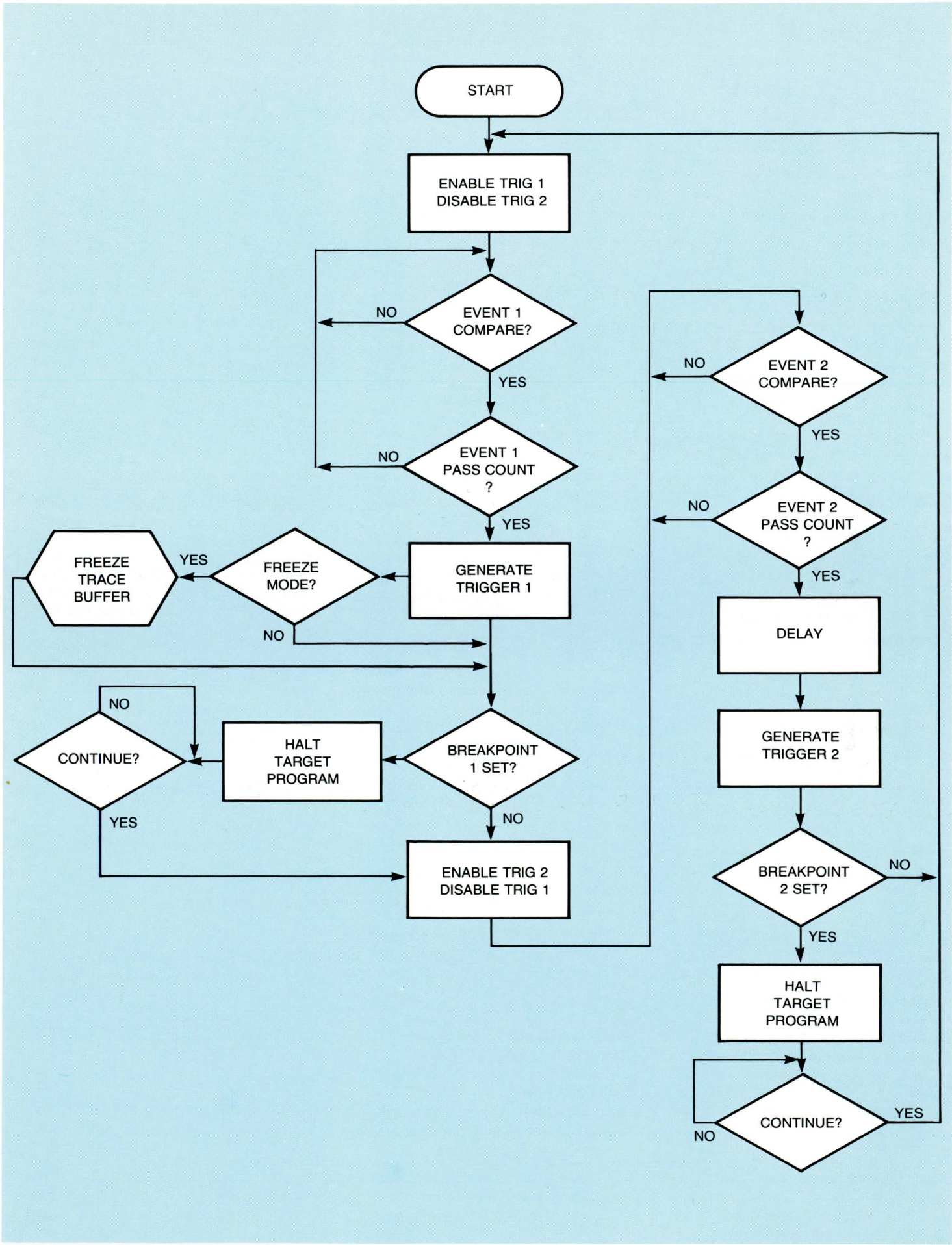


Figure 8. Arm and Freeze Modes.

A Powerful Command Set Provides the Control You Need for Effective Debug of Your Target System

Memory Display and Modification Control Commands

The 9508S relative memory addressing is a significant enhancement in debugging linked modules. The 9508S allows relative memory addressing with all commands.

ASM

Used to modify program memory using assembly language mnemonics rather than machine language. Command syntax is:

ASM (address)

where (address) may be relative or absolute and is the starting address for memory modification. The 9508S then prompts for input data. Input data is in the form:

OPERATION(OPERAND 1
OPERAND 2...)

where OPERATION is an instruction mnemonic or a pseudo operation. Pseudo operations that are supported include:

ASCII:	for entry of character strings.
BYTE:	for entry of 8-bit data.
BLOCK:	to reserve space without initialization.
WORD:	for entry of 16-bit data.

DISM

Used to display program memory is assembly language mnemonics rather than hexadecimal numbers. Command syntax is:

DISM (lower address)
and
(upper address).

Memory address may be relative or absolute. Display format is:

ADDRESS DATA OPERATION
OPERANDS*.

DUMP

Displays memory in hexadecimal and ASCII format. The hexadecimal display is the absolute value of data. The ASCII display is the upper and lower case alphabetic, decimal, and special character equivalents of the hexadecimal data. If there is no printable character equivalent, a period (.) is displayed. Command syntax is:

DUMP (lower address)
(upper address).

Memory addresses may be relative or absolute. Display format is:

ADDRESS SIXTEEN DATA
BYTES ASCII EQUIVALENT*.

EXAM

Used to display memory, one byte at a time, and optionally modify it. Command syntax is:

EXAM (address).

The 9508S will respond with the address, contents of that location, and will pause for possible modification. For example,

Input:

EXAM 1500

Response:

1500=A7−.

The 9508 will pause following the "−", waiting for user action. The user may:

- modify the contents, increment the address, and examine the next location.
- leave the contents unchanged, increment the address, and examine the next location.
- modify the contents of the current location and terminate the command.
- leave the contents of the current location unchanged and terminate the command.

FILL

Used to fill a range of memory with a constant data value. Any string up to 64 bytes long may be specified. When executing this command, the 9508S will perform a write followed by a read to verify each location within the specified range. Command syntax is:

FILL (lower address) (upper address) (ASCII or HEX string).

Example:

FILL 100 200 ABC82F

This example fills locations 100 through 200 with the values "AB", "C8", and "2F".

MOVE

Moves a block of memory from one location to another within the target microprocessor's on board memory, or 9508S emulation memory. Command syntax is:

MOVE (memory space) (lower FROM address) (upper FROM address) (lower TO address).

"Memory Space" defines the physical memory affected, which may be either 9508S emulation memory, target system on-board memory, or both. Specification codes are S for 9508S System Memory, and U for User Memory. Possible combinations are:

UU	Transfer is within user memory space where the memory is physically located within the target system.
SS	Transfer is within 9508S emulation memory where segments of the target system's address space has been mapped into 9508S emulation memory.
SU	From 9508S emulation memory to target system memory. This combination would be used to move a segment of memory from mapped emulation memory to target system memory. In this case the FROM and TO addresses could be identical.
US	From target system memory to 9508S emulation memory. This combination could be used to copy a program from ROM to 9508S RAM for debugging.

Example:

MOVE US 1500 2000 1500

This example will copy the contents of target system memory from location 1500 to 2000 into 9508S emulation memory starting at location 1500.

*Will appear on one line on CRT.

PATCH

Used to modify memory locations. Up to 64 continuous memory locations can be modified with a single command. When executing this command the 9508S will perform a write followed by a read to verify each location. Command syntax is:

PATCH (starting address)
(ASCII or HEX string).

Example:

PATCH 200
00123456789A

This example will patch locations 200 through 205 with 00, 12, 34, 56, 78, 9A respectively.

Memory Management Commands

BIAS

Used to establish values for the base address symbols W,X,Y, and Z. These symbols are used as offset values for computing effective addresses in the ASM, DISM, DUMP, EXAM, and FILL commands. Command syntax is:

BIAS (base W)(baseX)
(base Y)(base Z).

Example:

BIAS W=41A5 Z=1000

The address specified for each base register is a hexadecimal value. Base symbols are used like this with the EXAM command:

EXAM Z4A.

The effective address is the sum of the specified symbolic value and the stated address value. In this case, it is:

1000 + 4A = 104A.

MAP

Used to place segments of 9508S emulation memory within the address space of the target microprocessor, replacing any target microprocessor memory that may reside in that address space. This enables you to execute programs out of 9508S RAM and debug them in RAM prior to burning them into PROMs. You can also copy PROMs from your prototype hardware into RAM for execution and debug.

The 9508S provides the RAM you need. You can get either 16K bytes or 64K bytes of high-speed CMOS static RAM. This RAM is mappable in 256 byte blocks and provides write protection for the entire target system address space. So you can write protect both emulation memory mapped in the target system address space *and* your prototype RAM. Memory writes to emulation memory will be blocked and emulation halted. Memory writes to prototype RAM cannot be blocked, but emulation will halt immediately after the write occurs. Command syntax is:

MAP (user or system)
(WP or nothing) (starting address or address range,...).

Examples: MAP U WP 2000

This will cause the address range of 2000 to 20FF in prototype RAM to be write protected.

MAP S 0, 1000-1FFF

This command entry will cause emulation memory to respond to addresses from 0 to FF and from 1000 to 1FFF. Each time you enter a map command the 9508S responds with the current map; after these two examples the current map would look like this:

RAM MAP	STATUS	FROM	TO
S		0000	00FF
U		0100	0FFF
S		1000	1FFF
U	WP	2000	20FF
U		2100	FFFF

Communications Control Commands

LINKDEF

Used to define the communications link when the 9508S is operating with a host system other than the 9520 Software Development System. Default conditions for all of the parameters defined by this command are those required to communicate with the 9520.

Parameters defined by this command are:

Baud Rate (110 to 19200).
Parity (odd, even, none).
Format for Data Transfer (Tek hex, Intel hex, Motorola hex, or binary).
ACK Character Sequence.
NAK Character Sequence.
Error Timeout Limit.
Number of Retries for Error Correction Sequence.
Action on Error.
START Synchronization Character.

HOSTDEF

Used to define the characteristics of the Host system and its upload and download programs. This command is used in conjunction with the LINKDEF command to define the total communications environment. The default conditions are those required for operation with the 9520 Software Development System.

- Half or Full Duplex.
- Busy Protocol (None, Dsr, XON/XOFF, ENQ/ACK).
- XON Character.
- XOFF Character.
- ENQ Character.

- ACK Character for ENQ.
- Line turnaround time.
- Delay between characters.
- Host abort sequence.
- End-of-line sequence for data blocks.
- Block prompt sequence.
- End-of-file sequence.
- Wait for Host to echo data characters.

RHEX

Used to download object programs from the host development system to the 9508S or target system. This command is preceded by a MAP command if any portion of the program is to be loaded into 9508S emulation memory. This allows some portions to be loaded into target system memory and other portions to be loaded into 9508S emulation memory.

There are two modes of operation for the RHEX command. The first mode is the Single Command Mode where only one command is used to establish the link between the host system and the 9508S and to transfer data. When the 9520 is the host system, the command syntax is:

RHEX '(file name).

If you must execute several commands within the host system in order to initiate the transfer of data, there is a Terminal Mode which temporarily puts the 9508S into a "pass through" mode to allow the transmission of a sequence of commands from the 9508S terminal. Command syntax is:

RHEX TERM.

There are numerous options with this command to enable the 9508S to operate with a number of host systems. The default condition in every case is the configuration necessary to communicate with the 9520 Software Development System.

WHEX

This command is used to transfer programs or data from the target microprocessor's address space to the host system. Its modes of operation and syntax are similar to the RHEX except that the transfer is from the 9508S to the host system. There are Single Command and Terminal modes with all default conditions being those for 9520 communications.

TERMINAL

Puts the 9508S into a Terminal Pass through Mode where the video terminal that is connected to the 9508S effectively becomes a terminal on the 9520 Software Development System or other host system. This mode allows the 9508S operator to access the host development system to perform edits, assemblies, or any other functions that can be performed from a terminal on the host system. Command syntax is:

TERMINAL.

Register and I/O Port Display and Modification Commands

PORT

Used to display I/O ports or to write them. An optional REPEAT parameter allows constant read from or write to a port until the ESC key is pressed. To read from a port, the command syntax is:

PORT (address).

To read with repeated updates, the command syntax is:

PORT (address) REP.

To write to a port, the command syntax is:

PORT (address) (data)
or
PORT (address)(data) REP.

REG

Used to read and optionally modify register contents of the target microprocessor. The specific operation is target-microprocessor dependent, but the general function is to display the registers and pause for optional modification. Command syntax is:

REG (register name)
or
blank.

STATUS

Used to display target microprocessor registers and flags, or 9508S emulation environment, or both. Display of emulation environment is discussed in another section.

For displaying target microprocessor registers and flags, the command syntax is:

STATUS REG.

The display resulting from this command is target-microprocessor dependent.

Real-Time Trace, Trigger, Breakpoint Control Commands

QUAL

Used to specify the types of information to be recorded in the Trace Buffer. Command syntax is:

QUAL (parameter).

The parameters which may be used are:

ALL	Record everything.
FETCH	Instruction fetches only.
I	All I/O cycles.
IR	I/O read cycles.
IW	I/O write cycles.
M	All memory cycles.
MR	Memory read cycles.
MW	Memory write cycles.
R	All read operations (memory,I/O).
W	All write operations.

DRT

Displays contents of the Trace Buffer. The operator may specify all entries since the last GO command, or the last n entries, where n is 128 or less. Command syntax is:

DRT (number of trace events)
or blank.

The display format is:

ADDRESS	Memory or I/O port address.
DATA	8-bit data; hexadecimal.
BUS	Type of cycle recorded.
EXTERNAL	Binary representation of external probe data.
INSTRUCTION	Instruction mnemonic of the cycle is an instruction fetch.
OPERANDS	Instruction operands, in hexadecimal.

Screen format is a header line, one blank line, and twenty lines of trace data.

EVENT

Used to define the contents of the Event 1 and Event 2 comparison registers for the Trigger/Breakpoint equations. Command syntax is:

EVENT (1 or 2) (keyword).

The 1 or 2 selects the comparison register to be defined. The (keyword) is the definition of the values to be set, using this code:

A Address value, 16-bit. Operators are =, =>, and =<. To "don't care" address, use the clear option.

D Data value, 8-bit. Operators are same as Address. To "don't care" data, use the clear option.

B Control bus activity, or type of activity to compare on. Codes and values are the same as for the QUAL command.

E External probe data. Each bit is set and compared individually. Operations are 1, 0, or X, where X is "don't care".

C Clear the event. This sets A = OFF, D = OFF, B = ALL and E = XXXXXXXX.

*Will appear on one line on CRT.

Example:

EVENT 1 A=1234 D=A4
B=F E= XX110X0X.

TRIG

Used to define the Pass Count and Delay Count for the Trigger/Breakpoint equations. Command syntax is:

TRIG (1 or 2)(pass count)(delay count).

Example:

TRIG 1 P=12 D=10.

BREAK

Used to enable or disable breakpoints for the two trigger/breakpoint circuits. Command syntax is:

BREAK (T1/T2 BOTH)(DISABLE) (CONT)*.

The first parameter states which of the two circuits is affected: T1 is Trigger 1, T2 is Trigger 2, and BOTH means that T1 and T2 will be affected by the remainder of the command.

The DISABLE parameter states that no breakpoint will occur. The default condition is to enable a breakpoint, so if this parameter is omitted, breakpoints are enabled.

The CONT parameter may be specified if a breakpoint is enabled. It causes the 9508S to halt target microprocessor execution upon encountering a breakpoint, display a "break line" and automatically continue execution. The break line information is target-microprocessor dependent, but it will contain this type of information.

Registers	Register contents.
PC Last	Program counter address upon occurrence of the breakpoint.
PC Next	Where the target microprocessor program will go when execution is continued.
Cause of Break	Conditions that cause the breakpoint halt.
Status	Emulation status at break.

ONBRK

Used to specify a set of commands (a "command file") to be executed by the 9508S when a breakpoint is encountered. This is one of the most powerful commands in the 9508S in that it allows the operator to set up conditions which will automatically be accomplished at a breakpoint. Command syntax is:

ONBRK
(Command file).

The ONBRK command puts the 9508S into an interactive mode for entry of the command set. If commands are already in the file, new commands will be appended. To clear the file and start fresh, the command syntax is:

ONBRK CLEAR
(Command file).

Example:

ONBRK CLEAR
DRT 10
GO 1000.

This command set will automatically display the last ten lines of the Trace Buffer that were accumulated prior to the break and will resume execution at location 1000.

The ONBRK command may be used with all 9508S breakpoints. Maximum size of the Break Command File is 128 bytes.

TMODE

Used to define the relationships between the two trigger circuits and event registers. Command syntax is:

TMODE (mode).

Mode selections are:

IND	Independent mode.
E12	Limit mode, where Events 1 and 2 must occur simultaneously.
ARM	ARM mode.
FRZ	Freeze mode.

Descriptions of these modes are discussed and flow charted elsewhere in this Product Description on Pages 8, 9, and 10.

COUNT

Used to set up the general purpose counter. Sets the counter to zero and specifies units to be counted. When the program encounters a breakpoint, the counter value can be displayed. Command syntax is:

COUNT (units)

Units which may be specified are:

( )	Causes 9508S to display counter status.
MS	Milliseconds for execution timing.
US	Microseconds.
BUS	Count all bus transactions.
EMCLK	Count all emulator clock cycles.
FETCH	Count all instruction fetches.
RTT	Count all stores into the real-time trace buffer.
E1	Count all occurrences of Event 1 comparisons.
E2	Count all occurrences of Event 2 comparisons.
CLR	Sets counter to zero.

REGBRK

Used to monitor internal target microprocessor registers in a single-step mode. One instruction is executed, registers are examined to see if they match the break conditions, and the next instruction is executed if they do not. If they match the break conditions, execution is halted. Command syntax is:

REGBRK (CLEAR) (Break conditions)

CLEAR is used to erase previous conditions. If it is omitted, new conditions will be appended to old ones.

Specific registers which can be monitored are processor-dependent, but the relations which can be defined are these:

.EQ.	Equal
.NE.	Not equal
.LT.	Less than.
.ULT.	Unsigned, less than.
.GT.	Greater than.
.UGT.	Unsigned, greater than.
.LE.	Less than or equal.
.ULE.	Unsigned, less than or equal.
.GE.	Greater than or equal.
.UGE.	Unsigned, greater than or equal.

A range breakpoint can be set by specifying a register twice. For example,

```
REGBRK  RA .GE.  25
        RA .ULE. 7F
```

Eight break conditions may be established.

Emulation and Execution Control Commands

EMUL

Used to select one of two possible emulation modes:

System mode	Clock is provided by 9508S; program memory is 9508S emulator memory.
User mode	Clock is provided by user's prototype system; program memory may be in the user's system or 9508S emulation memory or both.

System mode will normally be used to debug software without prototype hardware being present and in early stages of hardware debug. User mode will be used for advanced hardware debug and system integration. Command syntax is:

EMUL (Sys or User)

CFILE

Used to create a command file of up to 128 characters in length. The file can be typed in from the console or downloaded from any host machine. Any 9508S command can be placed in a command file except the CFILE command itself. Command files are cleared with a clear command parameter or by resetting the 9508S. Command syntax is:

CFILE (clear) (TERM or TRANS)
or or
blank blank

The parameters TERM and TRANS allow host download using the same protocol as the RHEX command. If that parameter is omitted, the current contents of the command file buffer will be displayed and an interactive mode is entered to allow the user to add commands to the file.

EXCMD

Used to execute the contents of the command file. Command syntax is:

EXCMD

Normal output from all the commands in the file will be displayed at the console as each command is executed.

GO

Used to specify execution conditions for the target microprocessor program, and to start execution. Conditions which may be selected are:

- Continuous run.
- Single step or N single steps.
- Set a simple breakpoint at a memory address, a memory read, write, or both.
- Trace program execution, displaying each instruction executed, or jump instructions only. Trace may be specified within a range of addresses or for the entire program, or it can trace N lines and stop.

Command syntax is:

GO (start address) (UNTIL address or STEP nnn) (Trace selection).

Example:

GO 1000 UNTIL 2000

This example starts execution at 1000 and sets a simple breakpoint at 2000. Remember, address specification can be either absolute or relative.

GO 0 UNTIL 1000 TRACE ALL FR 500 TO 700

This example starts at 0, sets a simple breakpoint at 1000, and traces instruction execution for that portion of the program between 500 and 700. The base registers W, X, Y, Z could be used to establish offset addresses.

RESET

Applies a RESET signal to the reset pin of the target microprocessor. Specific actions within the target microprocessor are processor-dependent.

STATUS

Displays current emulation status and contents of target microprocessor registers and flags. The emulation status which is displayed consists of:

- Name of Target Microprocessor.
- BIAS (W,X,Y,Z Base Register) settings.
- COUNTER selection and value.
- EVENT definitions.
- MAPPING selections.
- REAL TIME TRACE qualifications.
- TRIGGER MODE.
- TRIGGER and BREAKPOINT definitions.
- REGISTER BREAK definitions.
- Emulator State:
  - Paused, Halted, Faulted, Address specified in GO encountered, Breakpoint x encountered, Single Cycle completed, Memory Write failure.

SVC

Used to enable user programs to request service from the 9508S operating system during their execution. The request for service is invoked by the user program writing a service request code into the memory location specified by the SVC command. This memory location is the first byte of a five-byte-long Service Request Block (SRB). The SRB specifies which service is to be performed and any information needed by the 9508S to perform that service. Command syntax is:

SVC (clear) (addr)
or
blank.

The clear parameter disables the service request and the address parameter specifies the first byte of the SRB.

The SRB contains the following information:

Byte 0. Service Request Code specifies which service is to be performed. (The writing of this byte by the user program causes the service request.)

Bytes 1 and 2. Two-byte absolute address that points to a data buffer.

Byte 3. Request dependent information.

Byte 4. Status of the service request returned by the 9508S. 00 = Good; FF = Invalid Request.

TERMDEF

Used to display or change Special Function Key values. Command syntax is:

TERMDEF (key)=(value)

Key values define communication protocol with the terminal.

Hardware Development Is Fast and Accurate When You Use the 9508S

Figure 9.

The 9508S is the only development tool you need for developing and debugging microprocessor-based hardware. The setup is simple. You just connect the 9508S emulation cable to the microprocessor socket of your prototype hardware, and you are ready to start.

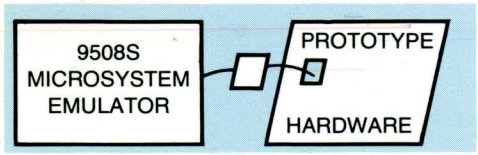


Figure 10.

Your system block diagram will probably look something like the one in the accompanying drawing. Here's how you could go about developing your hardware.

1. Replace the microprocessor in your system with the emulation probe from the 9508S. This gives you the ability to observe and control your hardware's operations using the powerful debug capabilities of the 9508S.

2. Build your system clock. Use 9508S RAM in the earlier stages of your design, and implement your system's RAM as you need it.

3. Build I/O ports into your system and check them out one at a time.

4. No matter how complex your I/O may be, from a simple solenoid control line to a complex communications channel, you can debug it knowing that it is the only unknown in the system, once your system clock is functional.

5. You will probably wait until the System Integration stage of your development before introducing ROM into your system. But you can build in the sockets and check the data paths. Just plug the ROMs with known data in them, and see if you can read them properly.

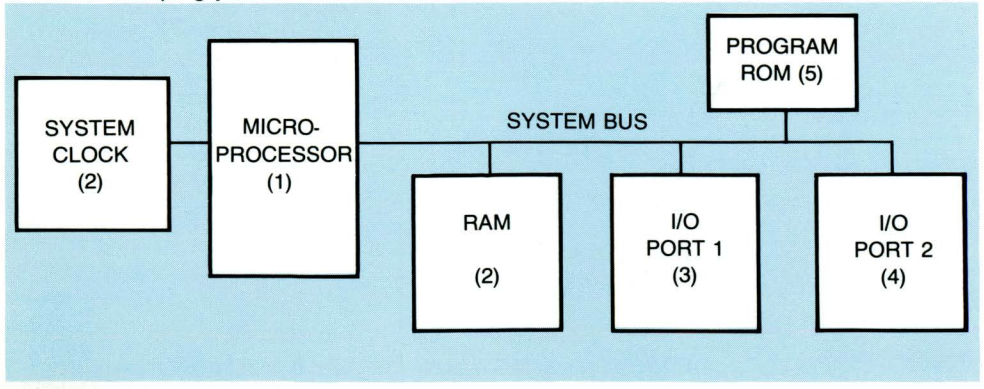


Figure 11.

The key to your hardware development is that you introduce resources into your prototype only when you need them. The 9508S substitutes its resources until yours are available in your prototype hardware. That way you work with only one or two unknowns at a time, and you can concentrate on the problem at hand. The 9508S will:

- Replace the microprocessor throughout hardware development.
- Provide a ***program memory*** to hold your development software routines.
- Provide a ***terminal interface*** so you can get your development software into memory and have an "intelligent control panel" into your system.
- Provide a ***command structure*** that enables you to observe and control operations in your prototype hardware.
- Do all of these things without demanding any time from your 9520 Software Development System or any other host computer.

The power of the 9508S emulation and command structure makes it possible for you to assign design tasks to several engineers and have them work independently. An engineer designing an I/O subsystem will not have to depend on the engineer who is designing the system RAM. Imagine what that can do for your schedules.

Software Development Demands 9508S Debugging Power

There are two distinct but closely related stages in the development of software for microprocessor-based systems, and two distinct development tools are needed to support them. The first stage involves writing your program and translating it from symbolic form into a form which the microprocessor can execute. This requires a software development system like Gould Millennium's 9520.

The second stage involves executing the program in a controlled environment so that errors can be detected and corrected. You need something that can execute the target microprocessor's program where the program is the only unknown. The 9508S does this very nicely.

The configuration for software development is simple. The 9520 Software Development System is used to enter, edit, assemble or compile, and store your target microprocessor's program. A temporary link is then established between the 9520 and the 9508S to download a program for execution and debug. When the 9520 is used, with its multi-tasking capabilities, the download is invisible to anyone who may be entering or editing code. The link may be broken once the download is completed.

Now you can begin to develop your software even before the prototype is available and be ready for integration as soon as the hardware is. Here's how you would go about developing your software.

1. Start in system mode, using emulation memory to completely debug all your non-hardware-dependent subroutines and main code and simulate I/O routines using the SVC facility.

2. Once the target system clock and I/O subsystem are ready, use user mode and emulation memory to integrate the hardware and software and completely debug all hardware-dependent subroutines.

3. When all the target hardware is complete, burn PROMs (if any) and move all memory resources to the target system to perform a complete system test.

4. Replace the 9508S emulator probe with the microprocessor and watch your prototype in action.

Software debug is accomplished in the 9508S without need for the software development system which can be generating other programs simultaneously. There are some significant benefits to this approach to software development.

- Your programs are executed in real-time using the actual type of microprocessor that will be used in your final system. You can completely debug major portions of your program. Only those portions which must interact with unique characteristics of your target system hardware must be left for system integration.
- No prototype hardware is needed for this debug stage. The only unknown is your software. Debugging is simplified significantly.
- Your development capabilities are effectively doubled because you don't tie up the software development system while debugging your software. One 9520 Software Development System can support several 9508S Microsystem Emulators.
- The commands that you use for software debug are the same ones that are used for hardware debug and for system integration. You don't have to learn the characteristics of several instruments so engineers and programmers can concentrate on debugging software and hardware rather than trying to remember how to perform a particular test sequence.
- The 9508S is transparent to your software just as it is to your hardware. With the exception of hardware-dependent code, if it runs in the 9508S, it will run in your hardware.

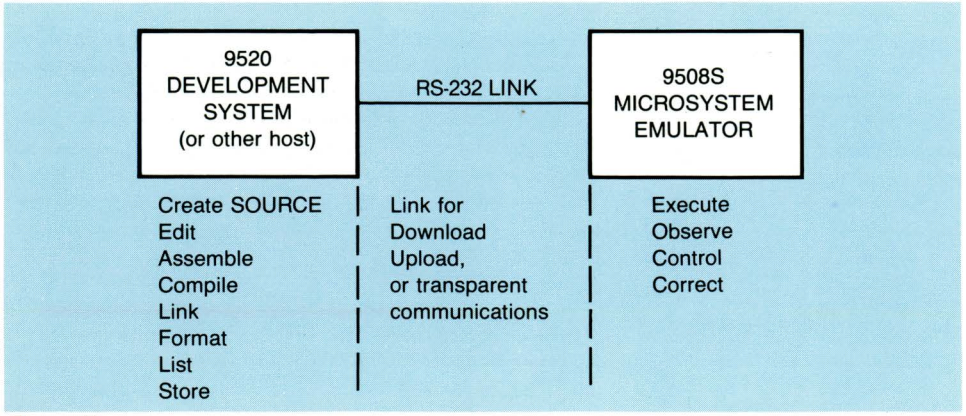
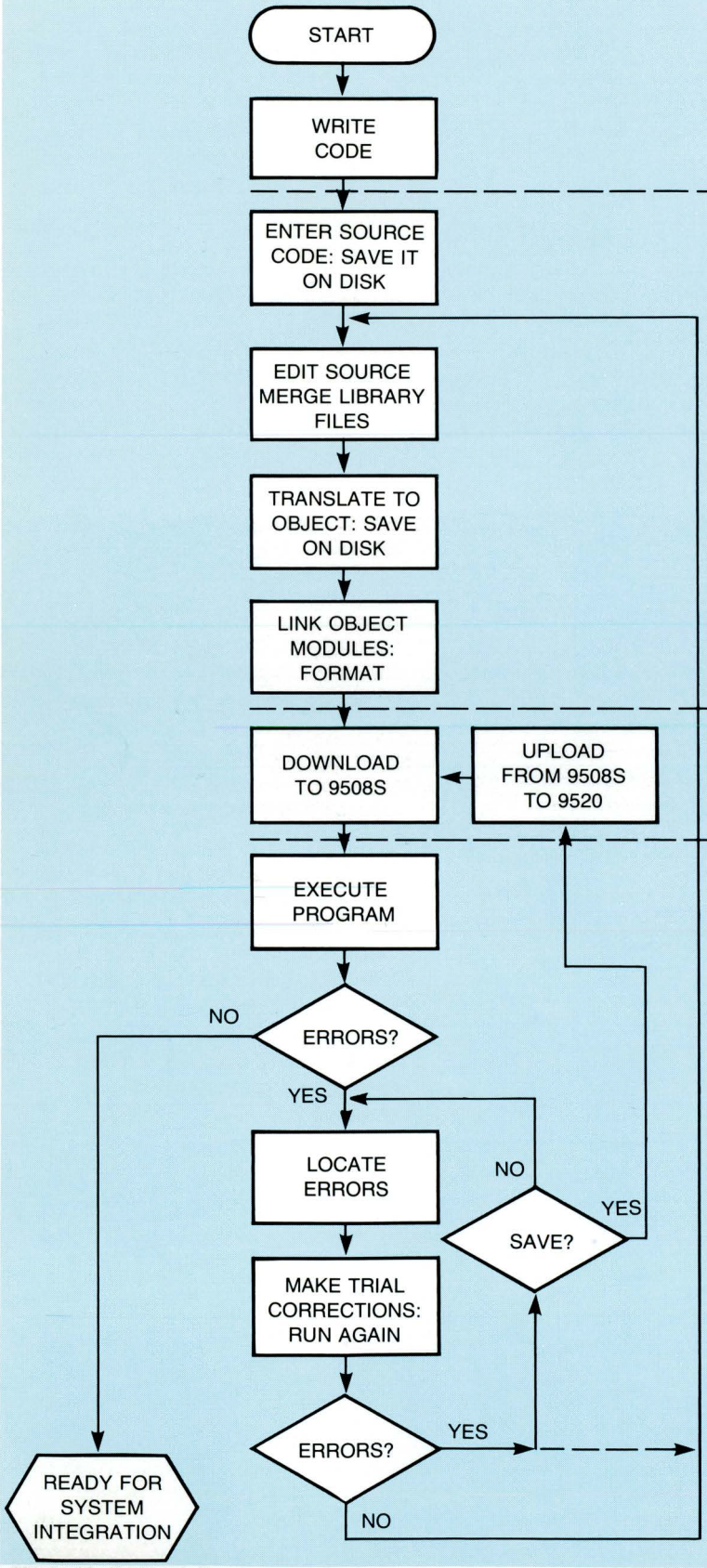


Figure 12.



Off-Line Functions

You must plan and write code off-line, but knowing that you will be developing it on the 9520 and 9508S lets you write to target microprocessor specs, not instrument specs.

9520 Software Development System Functions

Enter your code into the 9520 and create a Source File. The 9520's menu-driven Editor and Operating System will help you all the way. Save it on disk.

You can merge previously written library routines into your program so you don't have to reinvent the wheel with each program.

Translation into machine-executable code can be accomplished by a Cross Assembler for programs written in assembly language or a Compiler for programs written in PASCAL.

Object code may consist of relocatable modules. They must be linked for address resolution and formatted for transmission to a 9508S for execution and debug.

9520/9508S Link

The 9520 and 9508S are linked for downloading or uploading of the program. Once the load is done, BREAK THE LINK. The 9508S and 9520 can again operate independently.

9508S Microsystem Emulator Software Debug Functions

The 9508S is used to execute and debug the code that is produced by the 9520 since a software development system usually can't execute the programs it generates. The 9508S contains the actual target microprocessor and the controls necessary to fully execute it and debug it.

Errors will probably be found during the first debug pass and during subsequent passes. The 9508S will help you locate them with its powerful execution control, its powerful breakpoint capabilities, and its trace capability.

Trial corrections can be made using the 9508S memory display and modification commands. You can try the corrections immediately or you can save the partially debugged program by uploading it to the 9520. Download it and continue debugging when you are ready.

You will continue in this loop until your program runs properly. At certain points you will want to reassemble your code with corrections. You reestablish the 9520/9508S link, edit the program, assemble it, download again, and continue debugging.

When all bugs are exterminated, you are ready for final system test.

Figure 13. Software Development Sequence.

The Tough Job is System Integration

The 9508S Makes It Manageable

In hardware development you are working with a single unknown — your prototype hardware. Your new software is the only unknown in software debug. But in system integration, where you will make your new software run on your new hardware, you are working with two unknowns. No matter how much you test your hardware and software individually, there will be residual bugs that won't show until you try to make them work together. And these will be the deepest bugs—the most difficult to extract and exterminate.

This is where you will truly appreciate the transparency and debugging power of the 9508S. During the earlier development stages you will consider them to be useful and convenient. During system integration, you will realize that they are absolutely essential.

The setup for system integration brings the 9508S, the 9520, and your prototype hardware together. The 9520 has both source code and object code from your software development stage. You will connect the 9508S and 9520 together using the RS232 port, just as you did for downloading programs for software debug. The 9508S is connected to your prototype hardware via the emulation cable. Now you are ready for final debug of both software and hardware.

With this configuration you can methodically debug one program module at a time. Load your program module into 9508S emulation memory. Execute it with your prototype hardware using in-circuit emulation. Make changes in either hardware or software when bugs are found. When you have debugged your program as far as is possible in emulation memory, move it to your prototype hardware. This can be accomplished by burning a PROM using any standard PROM programmer (the 9520 will provide files in the appropriate format), by downloading your program directly into prototype RAM, or by moving from 9508S RAM to prototype RAM. The 9508S will handle all of the communications functions necessary to do it. You can then map the next program module into 9508S emulation RAM and repeat the process.

All of the debugging power of the 9508S will be brought to bear during system integration. Some of its unique features, which may not have obvious utility during hardware or software development, will be especially important here.

- Real-time trace will be useful during all stages of development, but the ability to record external data along with internal operations will become especially important. The "glitch catching" capability of the external data probes will be very useful. They extend your 9508S eyes beyond the target microprocessor and into your prototype hardware.

- The complex multiple breakpoint capability will be appreciated, especially the Arm, Limit, and Freeze modes. You will appreciate the ability to include external probe data in the breakpoint and trigger equations. They will let you break on events in your prototype hardware that are not immediately visible to the target microprocessor.

- You may choose to maintain the link between the 9508S and the 9520 Software Development System. If you do, you will be able to quickly call, edit, assemble, and download corrected program modules without having to reestablish a physical link. And you can talk to both the 9508S and the 9520 using the same video terminal with the 9508S in the Pass Through mode. Your debugging is simplified because you don't have to worry about your instrument configuration.

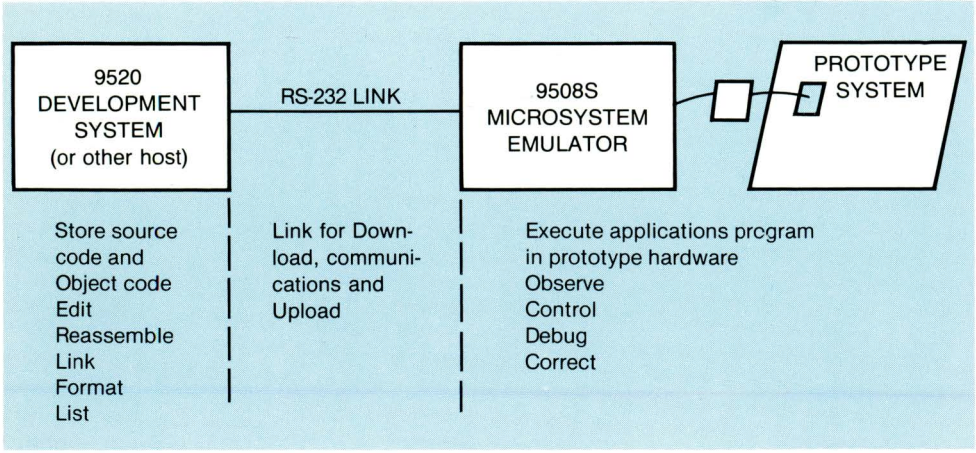


Figure 14.

Compatibility and Expandability

The 9508S Microsystem Emulator is the Gould Millennium best 8-bit stand-alone in-circuit emulator. Current users of the 9508 Microsystem Emulator can upgrade their system to a 9508S to take advantage of the expanded memory, higher speed and more versatile host communications, service request mechanism, and command file capability. The Gould Millennium 16-bit stand-alone ICE is the 9516 Microsystem Integration Station. Its features and capabilities are in keeping with the needs of 16-bit microprocessor-based system development.

All of the emulation modules which you purchase to work with the 9508S will also work with the 9516.

The 9508S, 9516, and 9520 can all be used with the multi-user 9540 Software Development System. High-speed RS-422 and IEEE488 ports are used to ensure upward compatibility.

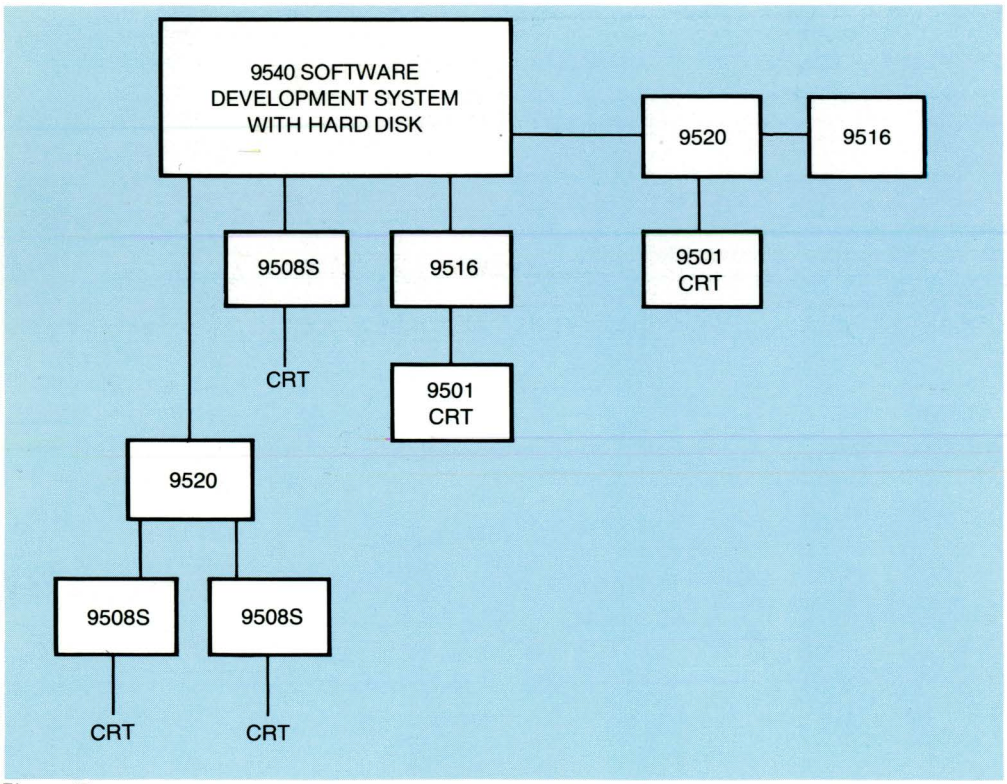


Figure 15.

Specifications

Microprocessor Emulator Support

8021	8048	6800A
8035	8748-4	6801
8035-4	8748-8	6802
8035-8	8049	6803
8039	8049-6	6808
8039-6	8050	6809
8040	8080A	68A09
8041A	8085A-2	68B09
8741A		6809E
8741-6	Z80A	68A09E
		68B09E

Software:
Break on target microprocessor registers.
Compare on equal, greater than, less than.
Exact operation is target microprocessor dependent.

Simple hardware:
Address compare for instruction fetch.

Complex hardware:
Two Event registers, comparing on
Address, 16-bit: =, ≤, ≥
Data, 8-bit: =, ≤, ≥
External data, 8-bit: 1, 0, "Don't care"
Bus transaction: Fetch, Memory read, Memory write, I/O read, I/O write, Read, Write, Memory, I/O, All.

Pass Count: 1 to 65,535 Events.

Delay Count: 3 to 65,535 counts.
Events.
Instruction fetches.
Trace stores.
Bus cycles.
Microseconds.
Milliseconds.

Satisfaction of Event compare, Pass Count, and Delay results in a Trigger being generated, and optionally a Breakpoint.

Modes: Independent, Limit, Arm, Freeze.

Real-Time Logic Analyzer

Trace Buffer:
35 bits and 128 states.

Content of each word:
Address, 16 bits.
Data, 8 bits.
External data, 8 bits.
Bus Operation, 3 bits.

Storage qualifications:
Operator selectable options:
All operations.
Instruction fetches only.
All I/O.
I/O Reads only.
I/O Writes only.
All Memory operations.
Memory Read only.
Memory Write only.
All Read operations.
All Write operations.

Base Address Registers

Four, labeled W, X, Y, and Z; 16-bits wide.

Contents added to called address to form effective address.

Command Files

128-byte buffer in master memory available for command file execution. Files can be downloaded from any host.

Service Request

The available service requests are:
Character in.
Character out.
Console status.
Line out.
Message out.

Data Communications Format

Gould Millennium Binary
Block Structured Hexadecimal ("Tek Hex", Intel Hex, or Motorola Hex)

External Data Probes

Ten probes on external cable:
8 data probes.
Clock probe.
Ground clip.

Stored in Trace Buffer synchronously.
Synchronous or Asynchronous clocking.
Used in Trigger/Breakpoint equations.

"Glitch catching" capability, conditioned by three switches on external probe pod.

Emulation Characteristics

Emulation speed:
Full speed of target microprocessor, its programs are stored in target system.

Processor-dependent if programs are stored in emulation memory (may be slower speed, or may be full speed).

Address usage:
All address space for memory and I/O is available to the target system. The 9508S uses a "shadow RAM" technique which does not intrude on the microprocessor address space.

Emulation RAM:
16K and 64K RAM modules optional.

Mapping:
16K to 64K memory mappable and write protectable in 256-byte blocks on any 256-byte block boundary.

Gould Inc., Instruments Division
4600 Old Ironsides Drive, Santa Clara, CA 95050-1279
(408) 988-6800 TWX: 910-338-0509
Gould Biomation and Gould Millennium Products

ICE49-101182 Printed in USA

GOULD

Electronics & Electrical Products