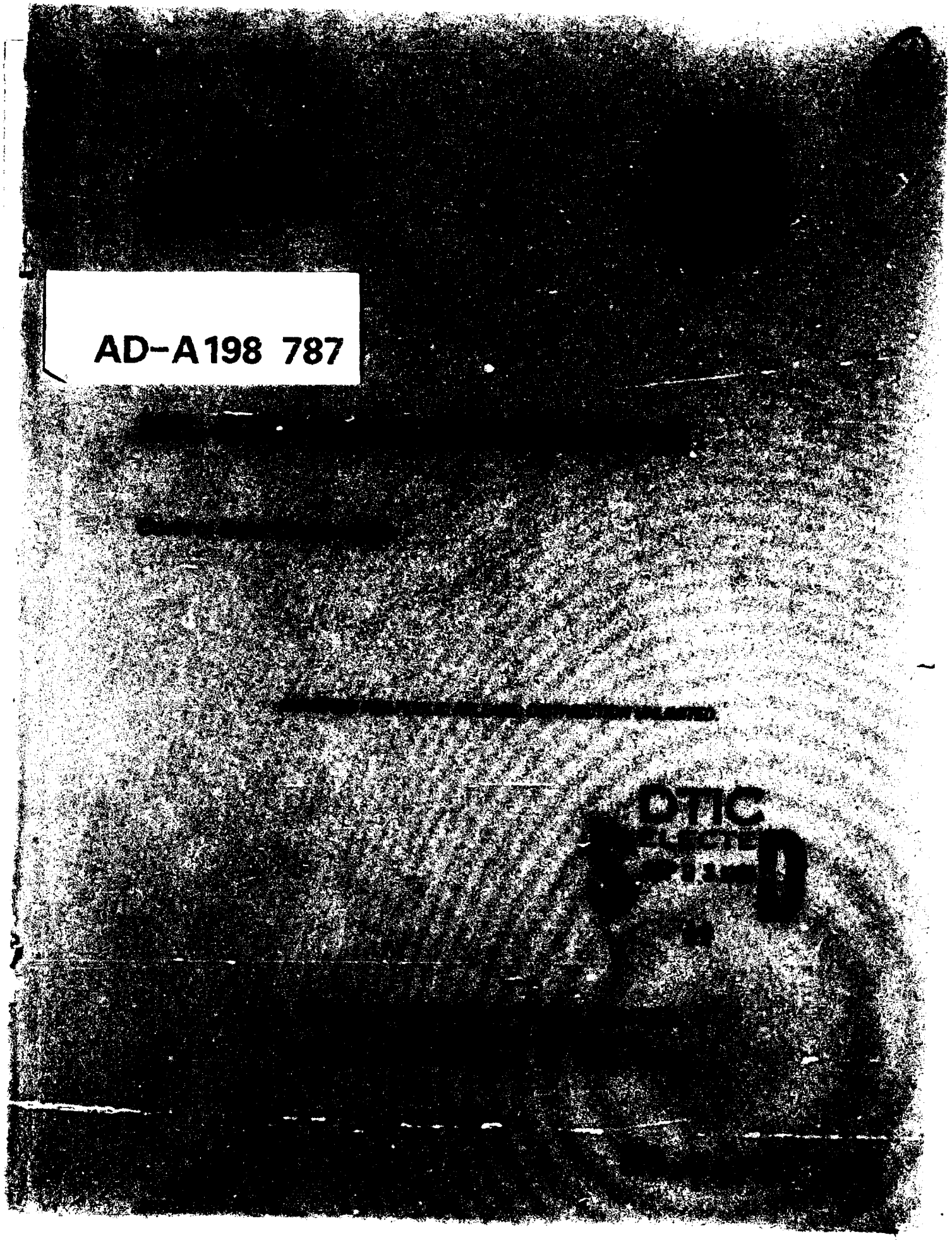


REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-91
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COTC)
6c. ADDRESS (City, State, and ZIP Code) Government Systems Division 3101 E. 80th Street, PO Box 609 Minneapolis MN 55440	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COTC	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-82-C-0175
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 63728F; PROJECT NO: 2529; TASK NO: 03; WORK UNIT ACCESSION NO: 05
11. TITLE (Include Security Classification) MIL-STD-1862B BRASSBOARD	
12 PERSONAL AUTHOR(S) N/A	
13a. TYPE OF REPORT Final 13b TIME COVERED FROM Sep 82 TO Jun 86	14. DATE OF REPORT (Year, Month, Day) June 1988 15 PAGE COUNT 44
16. SUPPLEMENTARY NOTATION N/A	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	MIL-STD-1862B Instruction Set Architecture Nebula
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The objectives of this effort has been to design, implement and install a Brassboard System which incorporates an implementation of the Nebula Instruction Set Architecture (ISA) defined by MIL-STD-1862B; to develop and install programs which support the development and execution of software targeted for this ISA; and to provide the programs required to test and maintain the hardware and microcode comprising the system. The Brassboard System and all software delivered and installed as part of this effort are the major elements of an experimental computer evaluation facility (testbeds) to be used for evaluating the 32-bit Nebula ISA in the context of Air Force applications.

This final report summarizes the work performed under Contract F30602-82-C-0175. It identifies and describes design criteria, implementation approach, accomplishments, conclusions and results derived from the MIL-STD-1862B Brassboard program. Major elements of the program are discussed in greater detail in the references cited in Appendices A and B.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☑ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL David E. Trad	22b. TELEPHONE (Include Area Code) (315) 330-2925 22c. OFFICE SYMBOL RADC (COTC)

DD Form 1473, JUN 86 Previous editions are obsolete.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1. INTRODUCTION

1.1. General. This final report identifies and describes the design criteria and accomplishments, and conclusions and recommendations of the MIL-STD-1862B (Nebula) Brassboard system development project.

1.2. Background. The objectives of this effort have been: to design, implement and install a Brassboard System which incorporates an implementation of the Nebula Instruction Set Architecture (ISA) defined by MIL-STD-1862B; to develop and install programs which support the development and execution of software targeted for this ISA; and to provide the programs required to test and maintain the hardware and microcode comprising the system. The Brassboard System and all software delivered and installed as part of this effort are the major elements of an experimental computer evaluation facility (testbeds) to be used for evaluating the 32-bit Nebula ISA in the context of Air Force applications.

The Nebula ISA is an outgrowth of the Military Computer Family (MCF) Project under the direction of the U.S. Army, which provides instruction-set-compatible computers of varying performance capabilities. This ISA has been developed in response to a need for a family of standard computers that decreases the life-cycle costs of automated battlefield systems by reducing the proliferation of ISAs. This is accomplished by increasing the portability and reliability of software through the use of a standard ISA. Formerly the acquisition of various ISAs embedded in military computer systems had been based on a strategy that involves purchases of processors with specified performance and physical characteristics from the lowest-cost sources. This approach results in a proliferation of instruction sets. Therefore these unique computers host different software-support capabilities, executives, data base systems, etc. Different parts of a major system may utilize different ISAs and support software, greatly increasing life-cycle costs. This proliferation has resulted in major problems which limit growth and flexibility and has produced severe interoperability problems, especially during system upgrades. The MCF project has focused on problems similar in nature to those experienced by the Air Force and the Navy. In addressing the problem the Navy has developed its own series of ISAs (AN/AYK-14, AN/UYK-43 and AN/UYK-44).

The Air Force has developed MIL-STD-1750A, the specification of a 16-bit ISA designed for use in avionics systems, and MIL-STD-1553B, a time-multiplexed data bus used extensively in avionics systems and having general applicability in real-time military systems. Based upon the success of these standards the Air Force is aware of the advantages to be realized by standardization at the ISA level. The use of MIL-STD-1862B in major Air Force systems has the potential of continuing this favorable trend. Within the next 5 years many of the existing computers in the Air Force inventory will have exceeded their useful life. Logistic support

problems are currently being encountered and it is difficult to properly support projected and changing mission requirements. The desirability of achieving software compatibility limits the ability to upgrade and modernize ISAs by exploiting modern technology. Although a specific ISA implementation such as the Nebula Brassboard might not utilize the latest hardware technology, it will be possible to employ the latest technology in future implementations. This approach achieves higher performance and greater system reliability while diminishing the amount of costly reprogramming.

1.3. Objectives. The primary objectives involved in the development and delivery of the Brassboard System are as follows:

1) Design, implement, and install a brassboard system which incorporates an implementation of the Nebula ISA defined by MIL-STD-1862B.

2) Develop and install software which supports the development and execution of software targeted for the ISA.

3) Provide the software and hardware required to test and maintain the hardware and to develop and test the microcode.

4) Provide documentation and familiarization of delivered hardware, firmware, and software.

5) Provide, as required, maintenance and spares following installation and acceptance.

Figure 1-1 contains a high-level block diagram of the Brassboard System hardware and software.

1.4. General Description. The Brassboard System includes the following items.

o Nebula Computer - A Nebula ISA implementation to be used in a laboratory environment.

Major Components of the Nebula Computer are:

- o the power supply
- o 4 megabytes of semiconductor random access memory (RAM) (expandable to 8 megabytes) which receives program modules downloaded from disk storage in the Computer Control Panel (CCP); organized in 4-byte words (32 bits) addressable at the byte level. Each word has additional check and control bits for a total of 45 bits per words.

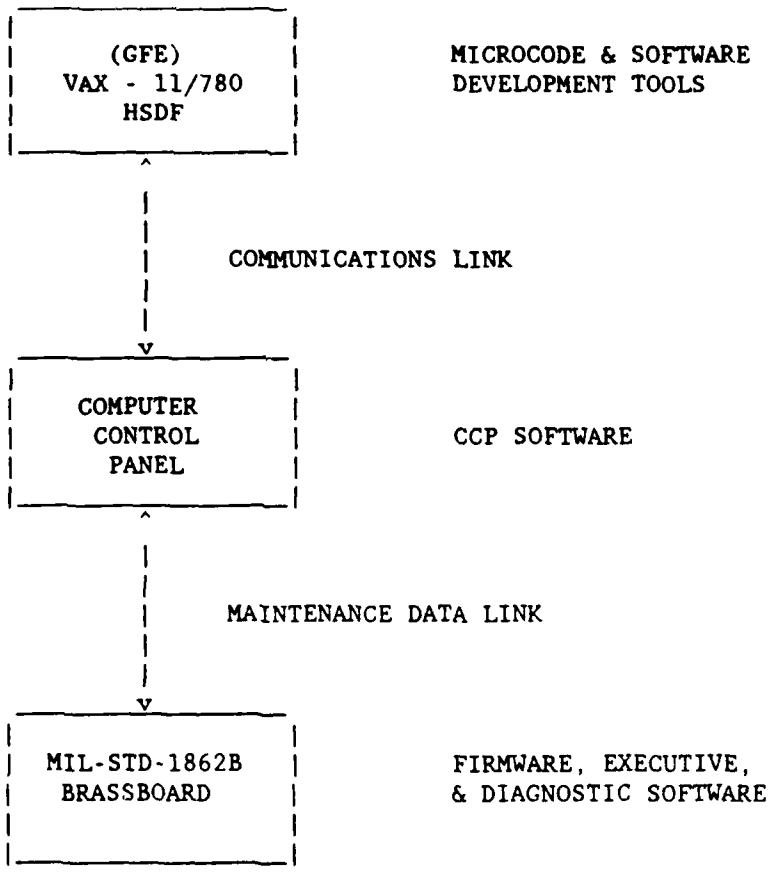


Figure 1-1. Brassboard System Hardware/Software Block Diagram

- an associative relocation cache of 4K x 82-bit words of random-access memory; organized in two partitions of 4K x 41-bit words each, which in turn are divided into 1K blocks of user/supervisor instruction and operand addresses.

- a memory map cache of 4K x 32-bit words

- a microcode (firmware) memory of 4K x 64-bit words distributed in various modules of the Input/Output Controller (IOC) (also downloaded from the CCP).

- a microcode (firmware) memory of 8K x 96-bit words distributed in various modules of the Central Processing Unit (CPU) (also downloaded from the CCP).

- additional microcode memory of 4K x 32 bits located in the Instruction Reformat (IR) portion of the Instruction Fetch/Reformat Module of the CPU.

- a cache of 4K x 32-bit words for the context stacks and working files.

- various read-only control memories and associated gate arrays which together are state sequencers.

- instruction fetch/reformat/distribution, memory management, input/output control, and arithmetic modules and interrupt/condition/event handlers.

- interface ports of various types and quantities, with the capability of having the quantity and types of ports changed.

- Computer Control Panel (CCP) - A group of devices designed to interface with the Nebula Computer to provide control, test and software development support functions. See the CCP-related manuals referenced in Appendix A. This provides the operator and maintenance interfaces and consists of items obtained from Heurikon Corp. and other vendors:

  - MC68000-based processor including 256K on-board memory and an additional optional 512K memory board

  - 20 megabyte hard disk

  - 1.2 megabyte floppy disk

  - Visual 210 keyboard and display terminal

  - Anadex DP9501A dot matrix printer

  - a UNIX System V operating system

- Microexecutive - A Nebula executive program installed in the Nebula Computer to provide an environment supporting the execution of Nebula programs. See the Executive User's Manual.

- Nebula Cross-Assembler and Linker - The GFE Nebula cross-assembler and linker are used to generate object and load modules for the Nebula Computer. These items reside in the host software development facility (HSDF). The CCP provides the storage and on-line interface for downloading the target modules from the HSDF to the Nebula Computer. See the Nebula Assembler and Linker User's Manuals.

- Nebula Microcode Development and Maintenance Tools - The tools required to maintain and modify the microcode (firmware) resident in the Nebula Computer have been developed and documented. These tools reside in and execute on the HSDF. The microcode routines developed using these tools are downloaded from the HSDF to the Nebula Computer using the CCP storage and maintenance interface. See the Universal Microcode Assembler and Linker User's and Maintenance Manuals.

- Nebula Emulator Microcode - The firmware has been developed to provide flexibility in the computer's normal operation by changing a firmware module. See the Nebula Brassboard Firmware User's and Program Maintenance Manuals. The firmware simulator allows the firmware to be tested prior to changing the brassboard. The Nebula Microcode Simulator User's Manual describes the firmware simulator.

- Validation Test Software - A loader/driver program has been written to control the running of validation routines and hardware/firmware integrity tests. A performance test measures the Nebula Brassboard execution rates. See the Acceptance Test Plan and the Nebula Test Software User's and Maintenance Manuals.

SECTION 2. DESIGN CRITERIA/ACCOMPLISHMENTS

2.1. Design Procedures/Criteria. This section briefly describes the design techniques and procedures used to develop all elements of the Nebula brassboard computer related to the Nebula ISA.

The hardware was designed on an in-house CYBER computer. Schematics were drawn on an in-house CALMA system. Hardware simulation was performed on the CYBER electrical computer-aided design (ECAD) system. The simulation output, a netlist, was sent to Multiwire® (a® division of Kollmorgen Corporation) for production of the Multiwire® boards. The boards were assembled and checked out in-house. The CCP was purchased off-the-shelf and best commercial practices were used in its prior development.

The logic module used in the brassboard is populated with small-scale integration (SSI) and medium-scale integration (MSI) logic devices. The Fairchild Advanced Schottky TTL (FAST) is used for the brassboard for the following reasons.

1) Low propagation-delays and power requirements

2) Improved dc thresholds

3) Reduced input loading; Schottky output drive

4) State-of-the-art logic functions

The module has the capacity to hold 375 devices. 343 of these may be in packages with 20 or fewer pins; 8 of these may be in packages with 28 or fewer pins; 4 of these may be in packages with 40 or fewer pins. Ceramic capacitors are distributed around the module to filter high-frequency noise on the power plane which may be caused by normal switching transients. Two bulk filter capacitors are located adjacent to the module connector. The module uses a 300-pin, low insertion force connector. It also has 125 test points arranged in five blocks. The module is 14.7 inches by 15.5 inches.

Specific design practices are documented in the Revised Design Plan for MIL-STD-1862B Brassboard System (13202326) and the Final Implementation Dependencies Addendum to Design Plan for MIL-STD-1862B Brassboard System (13202326). The latter document provides detailed descriptions of 48 design areas (39 of which were predefined to be implementation dependent; the remainder are specific to the implementation of the Nebula ISA design) requiring attention during implementation. Table 2-1 contains a list of these implementation dependencies.

All contractor-developed documentation followed Air Force standards.

TABLE 2-1. IMPLEMENTATION DEPENDENCIES

Description	Description
Use of PSW Bits 2:3	Device Vector Assignments & Use of Low Memory Space
Use of ASR Bits 1:7	Result of Illegal Access to I/O Space Registers
Representation of Context Stacks	Result of Writing Context & Map Pointers in I/O Space
Exact Location Referenced by Context Pointer	Trap or Exception Chosen Within Instruction
Setting of PSW Bits 2:3 for a New PSW	Order of Emulation Between "Nexts" in Instructions
Size and Format of Parameter Descriptors	LTASK Method of Forcing Consistency
Exception Handler State Encoding	SETSEG Action on Illegal Segment Specifier Address
Order of Acceptance of I/O Interrupts of Equal Priority	SETSEG Information Transmitted to IOC for Virtual Address
Ability to Detect Hard or Soft Memory Errors	Exact Time of Floating Underflow Check
Halt Required by RESET Function	Use of OP codes FA:FE
Definition of BIT Traps	Order Timing of Accesses to Memory Map Entries
Implementation Virtual Address Space	Implementation of Memory Interlock
Number of Hardware Supported Map Segments	Size of Parameter Three for Memory Management Traps
Effect of Self-Modifying Code	PPP IPL Parameter Code
Memory Map Caching Mechanisms	Floating Point Add/Subtract with Internal Result of Zero
Effect of Aliasing of Physical Addresses	Computer Generated Not a Number (NAN)
Subsetting of Memory Management	Conversion of Not a Number (NAN) Between 32 and 64 Bit Formats
Channel Configuration Register Definition	NAN Could Get Converted to Infinity
Recognition of Access to Program Counter	Floating Underflow
Recognition of Access to Message Pointer	
Channel Status Register Bits 2:14	
Optional RT Mode Commands	
Base Address of IOC Register Blocks	
Implementation Reserved IOC Registers	

2.2. Hardware Design Accomplishments. This section describes the studies, design, and development work to be accomplished for the brassboard hardware, the Computer Control Panel (CCP), and interfaces.

The brassboard hardware development includes the design or modification, fabrication, checkout, and documentation of the following module types:

1) Microprogrammable Processor (MP) Element
   - o Microsequencer
   - o Maintenance Interface

2) CPU
   - o Context Cache/Arithmetic
   - o Instruction Fetch/Reformat
   - o Memory Management
   - o Extended Arithmetic

3) Memory
   - o Memory Control
   - o Memory Storage

4) Input/Output (I/O)
   - o Instruction Execution
   - o Serial/Parallel I/O
   - o Serial/Digital (1553B) I/O

5) Motherboard

6) Chassis Assembly

Specific analyses and trade-off studies performed included implementation of memory segmentation and an analysis of micromemory width and depth. In conjunction with the CCP, an analysis of the interface between the CCP and brassboard was made.

The CCP development activity was divided into hardware and software. The hardware effort involved the selection and procurement of a commercial computer system to satisfy the CCP requirements. The software development is discussed in section 2.4.

CCP Hardware
- o CCP Selection/Procurement
- o CCP Communication/Data Interfaces

Specific analyses and trade-off studies performed included the CCP/Brassboard communications link and microprocessor system selection.

2.2.1. Brassboard Hardware Overview. The Nebula brassboard processor provides a vehicle for evaluation and optimization of the MIL-STD-1862B Instruction Set Architecture (ISA). To be a useful tool, the brassboard provides a flexible hardware architecture that will allow ISA experimentation. In addition, the processor executes 400 KIPS with I/O and 438 KIPS without I/O, contains 4 megabytes of memory (expandable up to 8 megabytes), and provides four I/O channels--two serial point-to-point, one parallel point-to-point, and one 1553 bus interface.

To achieve the flexibility required to allow experimentation with the ISA, and to handle the complexity inherent in the ISA, the brassboard makes extensive use of microprogrammable processors. The majority of the functions needed to implement the ISA are provided by hardware controlled by microcode. The microprograms that control the hardware are stored in a read/write control store to allow adaptation and experimentation.

There are four subsystems in the Nebula brassboard. Three of these contain microprogrammable processors. The CCP subsystem is based on an off-the-shelf microprocessor. The subsystem provides software and firmware loading capabilities and a man-machine interface. The CPU subsystem is based on a microprogrammable processor designed by Control Data. This subsystem performs the basic ISA processing. The third microprogrammable processor is located in the I/O subsystem. This subsystem provides the input/output channels for the brassboard. The final subsystem in the brassboard is the memory subsystem which provides the storage and control for up to 8 megabytes of memory.

The block diagram of the brassboard system is shown in Figure 2-1. The CCP subsystem is a commercially available microcomputer system. Each of the other subsystems is composed of logic modules, each of which can hold up to 375 dual in-line packages (DIPs). The modules utilize a multiwire board and is populated mostly with TTL circuitry. A single oscillator provides a clock signal to each of the modules. This signal allows each module to generate its own timing and keeps the system synchronized.

The CPU subsystem has six logic modules. Two modules implement a Microprogrammable Processor called an MP element. A description of an MP element is given in section 2.2.1.1. The other four modules in the CPU subsystem are the Memory Management module, the Context Cache/Arithmetic module, the Extended Arithmetic module, and the Instruction Fetch/Reformat module. The CPU subsystem performs basic ISA instruction processing (i.e., instruction fetching and interpreting), interrupt processing, and implements the Memory Management system defined by the ISA.

The I/O subsystem is composed of seven modules. Two modules implement an MP element; a third module is the Instruction Execution module. Together, these three modules implement an I/O Controller (IOC). The four other modules in the subsystem each provide one I/O channel. The I/O subsystem performs basic IOC instruction execution, transfers data to and from the

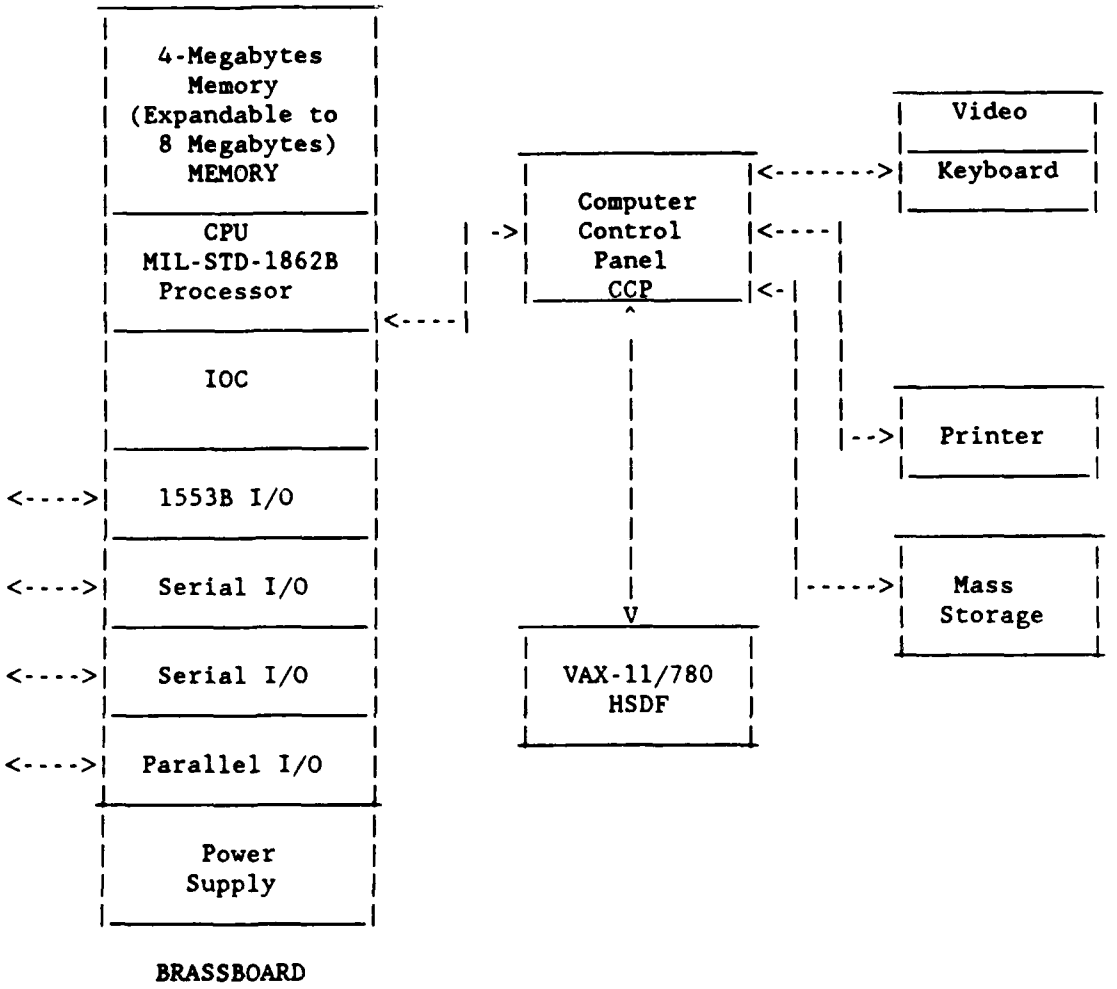


Figure 2-1. Brassboard System Hardware Block Diagram

memory subsystem, and from and to the I/O channels, and provides the I/O channel interfaces.

The memory subsystem is composed of at least three modules and may have as many as five modules. One module implements a memory control function. The other modules provide 1 or 2 megabytes of storage each. The memory is organized as 32-bit words, but each byte in the memory is individually addressable and accessible. The Memory subsystem provides a parallel bus port and four nibble bus ports. The subsystem performs basic storage, memory access prioritization (between CPU and I/O requests), data alignment for read and write operations, single error correction/double error detection (SECDED), and memory error logging.

2.2.1.1. The Microprogrammable Processor (MP) Element. The microprogrammable processor element (MP) is a standardized microprogrammed controller. It is intended to be combined with other application dependent parts to perform a specific processing function. Possible applications include central processing units, I/O channel controllers, memory controllers, and other processor type applications.

The MP element was created as one board or chip set to perform all the standard control functions of a processor. This controller is then used over and over again in many applications. This feature reduces design times and increases reliability. With the advent of very large scale integrated circuit (VLSI) technology, these become very important qualities.

Features of the MP element include:

- o Micromemory control
- o Micromemory sequencing
- o Maintenance channel interface
- o Fault detection/isolation
- o Deadstart bootstrap
- o Condition/event monitoring

All application dependent operations are implemented on additional boards or chips.

The MP element is partitioned into three major subsections; the microsequencer, the maintenance interface, and micromemory. They communicate with each other and the rest of a system over five buses; the data bus, the condition/event bus, the microcommand bus, the micromemory address bus, and the serial maintenance bus. The three subsections are synchronous and run off of a single 13.33 MHz oscillator.

The micromemory in the MP element is 32 bits wide by 8K words deep and contains some or all of the microcommand bits for the specific application. The RAM can be loaded with special firmware or diagnostic procedures from the console via the maintenance channel.

The maintenance interface performs the console interface functions. These functions involve transmission, detection, and responses to control messages sent via the maintenance channel. Console capabilities via the maintenance interface include:

- o Firmware level start/stop
- o Micromemory address and command display
- o Software/firmware breakpoints
- o Software debug aids
- o Display of test point and error status data

The microsequencer module controls the address sequencing of micromemory and the micromemory read/write functions. A deadstart Read-Only Memory (ROM) is included in the microsequencer to perform the initial load of the RAM micromemory on power up. Event and condition structures monitor the overall status of the system and are used to affect the flow of the firmware.

In the Nebula brassboard version of the MP element, the microsequencer and the micromemory share one multiwired board. The maintenance interface populates another board. For the VLSI version, it is hoped that the microsequencer and the maintenance interface shall each be on its own VLSI chip. The micromemory in this case is standard Field-Programmable Logic Array (FPLA) and RAM chips.

2.2.1.1.1. Microcommand Structure. A basic 32-bit microcommand is incorporated into the MP element design. This format provides enough parallel fields to handle all MP element controls, plus controls for application dependent parts. For high performance CPU emulation tasks, more microcommand parallelism is needed than is provided for by the basic format. Therefore, expanded microcommand formats are made possible.

The expanded microcommand format consists of the basic 32-bit format plus some expansion bits. The expansion bits are used entirely to control additional modules or VLSI chips. Each module in an MP element application receives only part of this microcommand. This feature cuts down on the number of pins required in a VLSI implementation.

2.2.1.2. The CPU Subsystem. The Central Processing Unit (CPU) subsystem of the Nebula brassboard is a microprogrammable processor consisting

of six interconnected modules. Two modules comprise an MP element. The other four modules provide the hardware needed to efficiently emulate the MIL-STD-1862B ISA.

The CPU is the mind of the brassboard system. Under microprogram control it performs the basic ISA instruction execution. The CPU fetches the instruction byte stream from memory. It interprets an instruction by fetching the operands, performing the operation(s), and storing the operand(s). The CPU also implements the ISA defined Memory Management system and interrupt scheme.

The MP element, consisting of two modules, provides the control function and the maintenance interface for the CPU subsystem. The MP element contains 8096 words of 32-bit micromemory and a microsequencer to address the memory. Each micromemory word is a command that controls the rest of the hardware in the CPU subsystem. A more detailed description of an MP element is contained in section 2.2.1.1.

2.2.1.2.1. Instruction Fetch and Reformat Module. The Instruction Fetch and Reformat module contains hardware that operates in the background with little help from firmware. This hardware fetches the instruction byte stream from memory; it preprocesses the instruction byte stream to produce instruction packets that are easier to decode; it passes the instruction packets across the instruction distribution bus to a FIFO buffer; and it reformats, upon firmware command, the instruction packet into entry points to firmware routines that interpret the instruction or access the operand. The remaining functions are CPU support functions. These functions include the interrupt system, the MIL-STD-1862B timers, and a nibble channel interface for CPU to IOC subsystem communication.

2.2.1.2.2. Memory Management. The Memory Management function of translating virtual memory addresses to physical memory addresses (also known as memory mapping) is performed using a high-speed associative cache memory called the Relocation cache. Portions of virtual memory addresses are used as indexes into the cache memory, other portions of the virtual addresses are used as associative data search keys, and relocated physical addresses are stored as data in the memory. This scheme performs address translation in one machine cycle (when the data is in the cache), and the associative cache memory is structured to provide a very good hit rate. The cache and its control take up a very large portion of the hardware on this module.

2.2.1.2.3. Context Cache/Arithmetic Module. The desired data manipulation section of the CPU is contained on two modules: the Context Cache/Arithmetic module, and the Extended Arithmetic module. The partitioning of functions between the two modules is as follows:

Major functions of the Context Cache Arithmetic (CA) module:

1) 32-bit arithmetic and logic section

2) Data transform section for conversions between differing operand word sizes

3) Floating point exponent section

4) Processor status word manipulation section

5) Context/file section to aid ISA register access, context file management, I/O space register storage, and extended working register storage which contains 1024 cached task context stack words, 1024 cached kernel context stack words, 1536 working file words, and 512 I/O space words

6) Shift matrix/shift-count control section

7) Data-test section that allows 46 distinct data-test functions for firmware conditional branching

8) High speed working register section with 25 individual specialized working registers

9) Set of interfaces to the CPU data bus, the memory bus, the Extended Arithmetic (EA) module, the CPU condition bus, and the standard CPU control signals

Major functions of the Extended Arithmetic (EA) module:

1) 32-bit arithmetic and logic section that is used along with the CA ALU to perform 64-bit operations, or that is used separately to allow two distinct 32-bit operations in one firmware instruction

2) 64-bit shift matrix that is able to shift left or right up to 31 positions

3) Multiplier section that multiplies 8-bit factors with 32-bit factors, thus producing 40-bit partial products

4) Parameter descriptor/floating point reformat section for firmware vector branching

5) 32-bit microcommand (UCMD) literal generator

6) 8096 x 48-bit micromemory with parity

7) UCMD matcher that produces pauses

8) Working register section with 11 individual specialized high-speed registers

2.2.1.2.4. Extended Arithmetic Module. The Extended Arithmetic (EA) module, in conjunction with the Context Cache/Arithmetic (CA) module, comprises the data manipulation section of the Nebula CPU. The EA module contains the following functional sections:

1) a 32-bit (LSB) extension of the CA module Upper Arithmetic Logic Unit (ALU) called the Lower ALU. Independent control of this ALU and the working registers in the EA module allows one 64-bit fixed or floating point arithmetic operation or else two independent 32-bit operations per instruction

2) a 64-bit input, 32-bit output shift matrix that is able to shift left or right up to 31 bit positions per instruction (SHIFT MATRIX)

3) a multiplier section which performs a 32-bit by 8-bit multiply and 40-bit partial product add iteration per instruction (8x32-BIT MULTIPLIER)

4) a 32-bit zero test unit (ZERO TEST)

5) a 32-bit microcommand literal generator (UCMD LITERAL)

6) a parameter descriptor/floating point reformat vector generator (PAR/F.P. RFMT)

7) an 8192 x 48-bit micromemory with parity (RAM MM)

8) a microcommand matcher that generates pauses when commands that initiate slow functions are recognized (COMMAND MATCHER)

2.2.1.3. The Memory Subsystem. The memory subsystem provides the Nebula brassboard with bulk storage for programs and data. The subsystem is composed of at least two modules. One module provides a memory control function for one or more (up to four) storage modules. Each storage module provides 1 or 2 megabytes of storage. The memory is organized as 32-bit words, but each byte in the memory is individually addressable and accessible. The memory subsystem provides a parallel bus port and four nibble bus ports. The subsystem performs basic storage, memory access prioritization (between CPU and I/O requests), data alignment for read and write operations, SECDED, and memory error logging.

The memory storage module contains the integrated circuit memory array to provide storage for 256K of 45-bit words. Each half of the storage array consists of four ranks of 64K x 1 bit dynamic memory elements and associated data buffers and multiplexers. The 45-bit word contains 32 bits of data, 7 bits of SECDED code, 2 reserve bits, and 4 byte breakpoint bits.

2.2.1.4. The Input/Output Subsystem. An I/O subsystem in a general purpose computer provides one or more communication paths or channels into and/or out of the computer. The I/O subsystem for the Nebula brassboard consists of seven modules. Three modules constitute an I/O channel controller (IOC). The four other modules provide the hardware necessary to implement the four channel types specified in the SOW. The IOC is a microprogrammable processor that interprets channel programs composed of Nebula I/O controller instructions. These programs move information from a channel to the computer's memory and vice versa.

2.2.1.4.1. The I/O Channel Controller (IOC). The IOC is composed of three modules; two for the MP element, and one called the Instruction Execution module. The MP element shall provide a 32-bit microcommand word to control the other modules and is described in sections 3.1.2. The Instruction Execution Module shall contain a 32-bit x 4096 micromemory used for the control of the Instruction Execution Module.

The Instruction Execution module provides all the hardware needed to execute up to a maximum of eight I/O channel programs in a time-shared manner. There is a large (4096 words x 32 bits) register file to hold channel configuration registers, channel status registers, channel program status registers, channel program address registers, pointer registers, etc., for the channel programs. 4096 words of storage support up to eight sets of 32-bit I/O space registers for Nebula I/O channels. The channel registers are easily accessible. The Register File is accessed by both the IOC firmware and by the CPU via the XIO nibble bus.

The module also provides an ALU to be used by the firmware when interpreting instructions and/or performing memory mapping and management. The module provides an 8 x 16 bit file to be used as an accumulator by the different channels. The module also provides the firmware with valuable internal status and conditions via an event and a condition system. The IOC communicates with the CPU via the XIO nibble bus. The final major portion of the hardware on the Instruction Execution module is the REFORMAT function to convert ISA instruction OP codes into micromemory addresses.

2.2.1.4.2. I/O Channel Modules. The memory interface handles all I/O channel transfers to and from the computer memory. It is also responsible for all instruction fetches and message transfers needed by the individual channels. The modules have the hardware necessary to perform these transfers independent of the IOC. The modules inform the IOC that a

transfer has completed via the event system. At the completion of a transfer, the status of the transfer is relayed to the IOC via the condition system and allows the IOC to read various status registers.

One of the modules provides a serial point-to-point interface and a parallel point-to-point interface. The other module provides a serial point-to-point interface along with a 1553B interface. Each module has the necessary hardware to allow the software to select what interface shall be assigned to each module. Each module also has the hardware necessary to make the appropriate internal "modifications" needed to handle the different interface types.

Each module provides a 4-bit high speed nibble bus to be used to transfer data between the I/O subsystem and the memory subsystem. With its own high-speed nibble Direct Memory Access (DMA) bus to memory available, and hardware packing and unpacking 32-bit words, each channel is capable of over three times the 0.5 megabytes/second aggregate I/O rate required for the system.

2.2.1.5. Computer Hardware Physical Description. Descriptions of the brassboard cabinet, logic modules, I/O connectors, and cables are provided in the Revised Design Plan for MIL-STD-1862B Brassboard System.

Motherboard Design Approach. The Nebula brassboard contains two motherboard assemblies; one contains and interconnects the CPU and memory subsystem modules, and the other contains and interconnects the I/O subsystem modules. Each motherboard consists of a multiwire board with space for 15 module connectors and wiring for intermodule signals. Several maintenance module connectors are located on the back of the motherboard. The maintenance connectors allow a module to be removed from the front of the motherboard and reinserted in the appropriate maintenance connector. This feature allows access to module components for scoping and other maintenance and troubleshooting activities.

2.3. Firmware Design Accomplishments. This section describes the design and development work accomplished for the brassboard firmware.

The brassboard firmware development involved the design, coding, debug, and documentation of the following subsystems:

- o CPU Emulation
- o IOC Emulation

Microprogramming is a design implementation involving the execution of each software-level instruction by executing a series of microcommands stored in a control memory. The control memory (micromemory) exists within the logic structure of the machine. The series of microcommands forms a microprogram, referred to as the firmware.

Firmware source code contains the microprogram and the mnemonic definitions for microcommands. Mnemonic definition at assembly time provides the flexibility to reflect hardware changes to microcommand formats and is accomplished by simply changing the firmware source. The assembler output may be downloaded to the CCP and saved on the CCP disk for loading into the brassboard micromemory.

Firmware segments can be loaded from the CCP disk. Segment No. 1 is loaded by a deadstart program and contains the firmware loader, a firmware debugging package, and basic self-test firmware. Segments No. 2 through N-1 contain firmware tests for those functions required to execute software test modules. The last segment loaded contains the MIL-STD-1862B ISA emulator and the firmware used for the software CCP interface. Additional flexibility is gained by providing the capability to change the microprogram after it is loaded into the brassboard micromemory.

Firmware for the IOC is defined, assembled, downloaded, and loaded into the IOC micromemory using the same procedure and support software used for the CPU firmware.

The brassboard firmware consists of a series of independent, functionally cohesive modules that interface with each other through a control structure. Within the control structure, each of the modules provides a specific function. For example, there is a functionally cohesive module for each software instruction, each event (microprogram interrupt), each CCP function, etc. This firmware design approach allows the flexibility to change the function of a software instruction, event, or other functions by changing a single firmware module.

The power up sequence includes testing the hardware using a sequence of firmware tests called the Initial Program Load (IPL) Test Firmware (ITF).

2.3.1. CPU Emulation. The CPU firmware is initialized during the firmware startup sequence and enters the idle loop to wait for an active event to occur. The CPU firmware consists of Instruction Fetch (IF), Instruction Reformat (IR), and Instruction Processor (IP).

Each software instruction execution begins when the IF obtains an instruction from memory and decodes it into individual characters or tokens, such as OP code and operand specifiers. These tokens are then packed into the IR FIFO stack.

The IR takes the instruction tokens from the IR FIFO and translates them into a series of firmware vectors and literal values. These vectors are IP firmware addresses that point to the various IF firmware routines needed to fetch operands, perform the operation, and store the result. The literal values can be sign/zero extended literals from the instruction stream, displacements, or the instruction stream virtual address.

The IP fetches each operand by using the appropriate firmware routine. The operation is performed using a firmware routine unique for each instruction. After the results are stored and the instruction process is completed, the firmware cycles to begin processing the next instruction.

For a detailed example of an instruction execution, refer to the Nebula Brassboard Firmware Program Maintenance Manual.

2.3.2. IOC Emulation. The IOP firmware consists of various event routines used to accomplish the I/O functions. I/O instructions are initiated by an event from the appropriate I/O channel. The firmware then reads the I/O instruction and uses the hardware format to obtain a firmware address vector. The vector points to an instruction processing routine which executes the instruction and returns to wait for the next event.

The IOC also has event routines to process unique I/O channel functions and error conditions.

Software visible I/O information is transferred to the CCP via the maintenance interface. Firmware event routines are used to process the requests from the CCP.

2.4. Software Design Accomplishments. The support software required for the Nebula brassboard computer system emulating the Nebula instruction set architecture (ISA) consists of the following:

1) Nebula Support Software (GFE)
   - o Nebula Assembler
   - o Nebula Linker

2) Microcode Support Software
   - o Microcode Assembler
   - o Microcode Linker
   - o Microcode Simulator

3) An Executive to control execution of programs on the brassboard system.

4) CCP Software

All of the above programs, except the executive, execute on the host software development facility (HSDF). The HSDF is a VAX 11/780 running the VMS operating system at RADC. All software developed or modified is in a higher order language (HOL), FORTRAN 77 or Instruction Set Processor Specification (ISPS), with the exception of the executive, which executes on the brassboard and must be written in the Nebula ISA.

2.4.1. Nebula Support Software. The Nebula support software consists of a GFE macro-assembler and a linker program written in FORTRAN. This system is used to assemble and link Nebula ISA programs.

2.4.2. Microcode Support Software. The Nebula brassboard computer is a microprogrammed machine, and, therefore, requires software running on an HSDF to support the microcode development. The support software consists of a microcode assembler to translate ASCII source to linkable object modules and a microcode linker to link these separately assembled object modules into an executable form for loading micromemory. A microcode simulator is required by contract to functionally simulate the linked microcommands and produce reports to assist in debugging microcode sequences.

2.4.2.1. Microcode Assembler. The microcode assembler, hereafter called UASSM, developed for the Nebula brassboard computer produces relocatable object modules to be linked later by the microcode linker, to produce a microcommand load module. UASSM is an enhancement of an existing assembler, written in FORTRAN, to accomplish the above.

The enhancements to the current assembler to develop UASSM include the following:

- Expand the size of the microcommand assembled from 48 bits to 96 bits.

- Previously, the assembler required the user to define the mnemonics used during coding, what their meaning is, and how they fit into the microcommand. The assembler built tables from this input each time it was invoked. The capability has been provided to dump these tables to a file once they have been created, and to use this file as input at assembly time until directed to recreate the tables by the user.

- Delete the automatic memory allocation from pass 1. This feature did not disappear, but was moved as a feature to the linker.

- Produce output consisting of relocatable object modules and a relocatable listing with errors flagged. The object modules produced by UASSM get quite lengthy compared to the source input. This feature is necessary to provide the information required by the linker for automatic memory allocation and the linking of separately assembled microcommand modules.

2.4.2.2. Microcommand Linker Editor. The microcommand link editor, hereafter called ULINK, accepts relocatable object modules produced by UASSM as input. These object modules accepted by ULINK are assigned space in micromemory and linked together to produce an absolute microcommand load module. ULINK was developed and implemented in FORTRAN.

The nature of the microcode dictates that the modules accepted by ULINK be a combination of relocatable, partially relocatable, and/or fixed sequences of microcommands. ULINK may be thought of as a generic linker with additional capabilities. The additional capabilities consist of:

1) Automatic memory allocation, which assigns memory to the fixed sequences first, then the partially relocatable sequences next, and, lastly, the run anywhere sequences based on longest sequence to smallest sequence of microcommands.

2) Producing an ORG TABLE. This table contains information such that when relinking and adding or deleting or changing object modules, the unchanged modules can be assigned to the same micromemory addresses they had from the previous linker run. This table is also used to deallocate an object module from memory assignment (when it is the module that changed). This file only exists after an initial linker run and is used and regenerated on subsequent runs.

3) Accepting the relocatable listings from the assembler as additional input, and, at link time, generate new listings with the microcommands absolute address replacing the relocatable address. Only those listings input are regenerated.

4) Accepting a command language to direct the linking process with respect to what the user wishes to accomplish. One command is accepted per line. Commands may be typed in directly from the user's terminal or can be retrieved from a command file.

As in most linkers, ULINK produces a number of reports, in addition to absolute listings, such as a micromemory allocation map and a cross-reference listing.

2.4.2.3. Microcommand Simulator. The microcommand simulator, hereafter called USIM, is a functional simulation of the real machine. USIM is coded in the ISPS hardware description language. The use of ISPS makes the implementation, modification, and maintenance of USIM a much simpler task. The ISPS language and run-time system support a wide range of applications, and its development was supported by the Defense Advanced Research Projects Laboratory. ISPS allows the user to describe the interfaces (external structure) and the behavior of hardware units (called entities). The interface describes the number and types of carriers used to store and transmit information between units. The behavioral aspects of the unit are described by procedures which specify the sequence of control and data operations in the target machine. The ISPS language favors the behavioral aspects of a machine over the structural aspects by abstracting information.

2.4.3. Executive. The executive (hereafter referred to as the EXEC) is a single tasking executive. The source language for EXEC is the Nebula assembly language. A User's Manual and Programmer's Maintenance Manual have been generated for EXEC.

The user structures an application program into one or more tasks. A task is an executable entity that performs a particular function required in the application program. The application program and hence the tasks run under the control of EXEC. The major functional sections of EXEC are listed below.

- o Task Management
- o Memory Management
- o I/O Management
- o Interrupt Management
- o Error Management
- o Initialization

2.4.4. CCP Software. The CCP software operates the CCP which, in turn, controls and monitors the Nebula computer. The CCP software also provides a data transfer mechanism between the brassboard and the HSDF, and provides tools for firmware and software development on the brassboard. The CCP function is implemented by a series of software programs resident on a commercial microcomputer. These programs are written in C and take full advantage of the Unix V operating system based on the microcomputer.

The CCP software consists of two main programs, NEBCCP and Kermit. The NEBCCP program monitors and controls the Nebula brassboard. It consists of routines which perform the various command and display functions of the program. The NEBCCP program is described in the Final Computer Control Panel (CCP) Software User's Manual for MIL-STD-1862B Brassboard, document number CRDBM001. Kermit is a standard computer interface package, used here as the interface between the CCP and the HSDF. A C language implementation of Kermit was used, and was modified to meet the particular constraints of this system. Kermit is described in the Kermit (VAXLNK) Reference Manual.

2.5. Testing. This subsection describes the test software. The following test software was developed:

- o Performance Test
- o System Function Test (SFT)

2.5.1. Performance Test. The performance (benchmark) test developed by the Military Computer Family (MCF) program was used to measure the instruction execution rate of the brassboard. This test implements the instruction mix specified in Appendix 1 of Annex 1 of the Statement of Work (SOW). The performance test included one exception to the instruction mix as specified in Appendix 1. Because it was determined to be feasible and practical, the test included a single execution of each of the instructions listed as unused. The WINDOW, EXCEPT, and RAISE instructions were implemented in the performance test. The interval timers available to the software were used to measure instruction execution rate. The source language was the Nebula assembly language. The test was loaded and controlled via the CCP.

The performance test was run three ways: without memory mapping and without I/O the system executed 441 KIPS; with memory mapping and without I/O the system executed 438 KIPS; with I/O the system executed approximately 400 KIPS.

2.5.2. System Function Test. A system function test (SFT) was developed to verify proper operation of the brassboard. The SFT included several individual programs. The major functional tests included were CPU, memory, I/O controller, 1553B I/O channel, parallel I/O channel, and serial I/O channel. All of these tests were individually linked and run under the control of an executive. Testing of the I/O channels was accomplished via wraparound.

The validation tests developed by Tartan Labs provided the bulk of the testing to be included in the SFT. The GFE validation software verified the following ISA characteristics:

- o Variable length instructions and operands
- o Addressing modes
- o Control structure
- o Data types
- o I/O
- o Exception handlers

Validation tests for the firmware implementation of MIL-STD-1862B were included as part of the CPU/IOC programs.

2.5.3. Acceptance Test. Successful completion of the acceptance test at the customer site constituted acceptance of the entire Nebula ISA brassboard system including hardware, firmware, and software. The same acceptance test was used for both the preliminary and final acceptance.

SECTION 3. CONCLUSIONS

3.1. Hardware. The Nebula hardware implementation, architecture concepts, and performance parameters closely match the Revised Design Plan. Due to the nature and philosophy of the Nebula ISA, the brassboard implementation is somewhat hardware intensive. The approach is to eventually take advantage of high gate counts available in VLSI integrated circuits to provide a processor with features to efficiently execute Ada software. The brassboard hardware design makes extensive use of cache memories, microprogrammed hardware architectures, and hardware "state machine" processors based on FPLAs (field programmable logic arrays), PROM memories, and RAM memories.

Cache memories are used in the memory management section to gain performance in the logical to physical address mapping process. Cache memories are also used in the main arithmetic section, to provide fast access to general registers and parameters associated with the current procedure context.

Microprogramming is used to control the execution of software instructions by the CPU hardware modules. This technique provides flexibility, and also utilizes high density memory devices to contain the complex control sequences involved in instruction execution. The IOC also utilizes microprogram control.

Hardware state machines are essentially small-scale microprogrammed controllers. They are used in the memory control section to provide the DRAM (dynamic random access memory) control signal sequences, and to implement the complex memory access commands. These commands involve variable-length data accesses which may begin on any byte address. Thus, a data access may cross 32-bit main memory word boundaries. Two hardware state machines are used in the instruction fetch and reformat section. These machines control the instruction fetch from main memory, pre-processing of the instruction OP code and operand specifier fields, and conversion of the instructions to information packets executable by the firmware. Finally, a state machine is used in the memory management section to control the memory map table search when the required mapping information is not in the associated cache memory.

The resulting hardware design has the potential to be implemented using VLSI technology. This implementation would consist of several VLSI parts, along with memory parts for micromemory, cache memory, state machine control memory, and main memory.

The hardware utilizes a fully synchronous, single clock-per-cycle timing mechanism. A 75 nanosecond cycle time is achieved by use of Fairchild FAST TTL circuits, and by choosing a three-stage pipeline in the main

microprogram control structure. (The three stages include microcommand addressing, access, and execution.) The 13.33 MHz clock signal is distributed to each module in the system, including processor, memory, and IO sections, in a manner which minimizes clock skew.

The hardware performance, combined with the firmware emulator, achieves a machine performance of approximately 400 KIPS, compared to the required 250 KIPS. The synchronous timing approach would be compatible with a higher performance future VLSI implementation.

3.2. Firmware. The Nebula firmware implementation and structure closely match the Revised Design Plan. Due to the complexity of the Nebula ISA, the firmware needed to implement the ISA is also complex. The general organization of the firmware is a sequence of micromemory overlays. The overlays include a loader overlay, several test overlays, and the emulator overlay.

The use of the firmware overlays to accomplish firmware testing of the various hardware functions allows the hardware to be set up in test modes not possible with the ISA emulator in micromemory. These firmware test modes reduce the test generation times and improve test coverage.

The CPU ISA emulator is organized into the three levels of firmware: the instruction fetch (IF), the instruction reformat (IR), and the instruction execution (IE) processor. The use of three levels of firmware is an effective way to reduce the complexity of the firmware and increase performance by pipeline processing. Additional performance is obtained by the use of a wide microcommand (96 bits) in the IE processor. The use of structured firmware modules in a control structure reduces the complexity of individual firmware functions to a manageable level.

The IOC event (firmware interrupt) driven processor allows the IOC functions to be independently structured firmware modules. Each module's complexity is easily reduced to manageable levels.

Firmware mnemonic definition at assembly time and under control of the senior firmware design engineer allows rapid changing of microcommand mnemonic definitions. The rapid response to changing hardware also allows the microcommand mnemonic definitions to become the communication tool to define hardware functions. The resulting up-to-date hardware definitions significantly reduce firmware coding and hardware design time.

The mnemonic style used is free-field arithmetic expression. An example of this style is A=B+C+1 (add the content of register B to the content of register C plus one and place the result into register A). This mnemonic style is easy to read and understand by anyone picking up a firmware listing.

The firmware is assembled as small modules and links with the existing firmware or as a complete package if required. Multiple use of microcommand bits for control and jump fields is facilitated by micromemory address matching and allocation functions of the firmware linker. The linker also takes care of deallocation of removed or changed firmware modules. Complete linking of all modules is only required when matching memory addresses are no longer available.

3.3. Software. The microcommand support software developed for assembly of the firmware proved to be quite valuable. The assembler allows for definition of mnemonics at runtime; this allows the firmware programmers to change the mnemonics without changing the assembler. The linker optimizes micromemory usage by its memory allocation scheme that assigns memory to the fixed locations first, then the partially relocatable sequences, and lastly, the "run anywhere" sequences based on longest sequence to smallest sequence of microcommands. The absolute listings that the linker outputs are valuable in checkout. The flexibility of the assembler and linker allows the firmware programmer to generate patches in the lab; this allows for much better checkout times since there is no wait before new firmware could be tested.

The ISPS simulator matches the hardware, and was used to verify hardware design of individual modules. As a system simulator though, it did not meet the original design expectations. Due to the complexity of the hardware, the simulator became large and cumbersome often requiring over 8 hours to initialize the system. This makes simulation of the majority of the functional firmware impractical compared to running on the actual hardware.

After simulation using the government furnished software simulator, the Executive ran on the hardware with only a few minor modifications. The test software provided by the government is useful for debugging the firmware.

Midway through the development process it was discovered that Kermit, a standard file transfer protocol, fit the needs that were to be provided by a transfer utility we were going to develop. Rather than develop a new utility, it was decided to use Kermit; this provided a download capability from a greater number of systems.

SECTION 4. RECOMMENDATIONS

Recommendations dealing with detailed implementation dependencies are included in the Revised Design Plan addendum and Table 2-1. Specific items not addressed by the addendum are described in this section.

4.1. Hardware. For a future, higher performance VLSI implementation, two additional changes should be considered. First, a main memory cache structure would be useful for enhanced performance. This cache memory would be at the expense of additional parts, however, and may be complicated due to the byte addressing nature of the ISA. Second, the size (in terms of bits per word) of the existing micromemory and memory management cache memory should be reduced if the performance impact is not too great. This would reduce the number of VLSI chip pins required to interface to these memories, and may reduce the number of required VLSI parts.

4.2. Firmware. The firmware design, mnemonic definitions, extensive in-code documentation, and the use of the UASSM and its linker produced a modular and supportable firmware package. The firmware development tools have applicability to future developments.

The hardware simulator was not available until late in the project, therefore reducing checkout efficiency and allowing errors to remain undetected until the hardware design was difficult to change. The simulator would have been more beneficial if it would have been available in advance of the hardware.

Small performance improvements may be obtained by the optimization of some firmware modules.

Significant performance improvements can be accomplished by the redesign of the instruction reformat (IR). The approach would involve moving the operand fetch, operand store, and branch instruction processing to the IR.

4.3. Software. More software diagnostics should have been added to the set provided by Tartan Labs that would more fully test addressing modes of instructions.

The computer generated arithmetic tests provided by Tartan Labs would have been more helpful if the computer generated addresses and offsets had been in hexadecimal rather than in decimal. It became difficult to calculate the complex addressing modes using 10-digit decimal numbers.

The CCP software should have been completed much earlier in the project to facilitate development of the ISA.

APPENDIX A. REFERENCES

13202326 Revised Design Plan for MIL-STD-1862B Brassboard System, March 1985

13202326 Addendum Final Implementation Dependencies Addendum to Design Plan for MIL-STD-1862B Brassboard, June 1985

--- Final System Reference Manual for MIL-STD-1862B Brassboard System, June 1985

13203188 Final Nebula Computer Hardware Reference Manual for MIL-STD-1862 Brassboard System, June 1985

CRDAM001 Final Nebula Assembler User's Manual for MIL-STD-1862B Brassboard System, November 1984

CRDAM002 Final Universal Microcode Assembler and Linker User's Manual, March 1985

CRDAM003 Final Universal Microcode Assembler and Linker Program Maintenance Manual, November 1984

CRDBM001 Final Computer Control Panel (CCP) Software User's Manual for MIL-STD-1862B Brassboard, August 1985

CRDBM002 Final Computer Control Panel (CCP) Software Program Maintenance Manual for MIL-STD-1862B Brassboard, August 1985

CRDDM001 Final Test Software User's Manual for MIL-STD-1862B Brassboard System, January 1985

CRDDM002 Final Test Software Program Maintenance Manual for MIL-STD-1862B Brassboard System, January 1985

CRDFM001 Final Firmware User's Manual for MIL-STD-1862B Brassboard System, December 1984

CRDFM002 Final Firmware Program Maintenance Manual for MIL-STD-1862B Brassboard System, April 1985

CRDJM001 Final Nebula Executive User's Manual for MIL-STD-1862B Brassboard System, January 1985

CRDJM002 Final Nebula Executive Program Maintenance Manual for MIL-STD-1862B Brassboard System

CRDKM001 Final Nebula Firmware Simulator User's Manual for MIL-STD-1862B Brassboard System, March 1985

CRDKM002 Final Nebula Firmware Simulator Program Maintenance Manual for MIL-STD-1862B Brassboard System, November 1984

CRDLM001 Final Nebula Linker User's Manual for MIL-STD-1862B Brassboard System, November 1984

APPENDIX B. BIBLIOGRAPHY

The following documents are used in the development of the brassboard design and constitute additional information for the reader.

- Revised Statement of Work for MIL-STD-1862A Brassboard, dated 82-4-27.
- Amendment No. 1 to Revised Statement of Work, dated 82-6-18.
- Annexes to Revised Statement of Work.
- Annex 1 - Revised Development Specification for Nebula Computer
- Annex 2 - Revised Development Specification for Computer Control Panel
- Annex 3 - Revised Development Specification for Serial Point-to-Point Interface
- Annex 4 - Revised Development Specification for Parallel Digital Point-to-Point Interface
- Control Data MIL-STD-1862A Brassboard Technical Proposal, DG82G001082, dated June 1982.
- Contract for MIL-STD-1862A Brassboard, F30602-82-C-0175.
- Revised Contract Data Requirements List, dated 82-4-27 and associated CDRL backup sheets.
- Contract Modifications for Engineering Change, Spare Parts and Software License.
- UniPlus+™ System V Unix Operating System Manuals
  - User's Manual, sections 1-6
  - Administrator's Manual
  - Administrator's Guide
  - User's Guide
  - Document Processing Guide
  - Support Tools
  - Programming Guide
- Heurikon Introduction to Unix

APPENDIX C. GLOSSARY OF TERMS

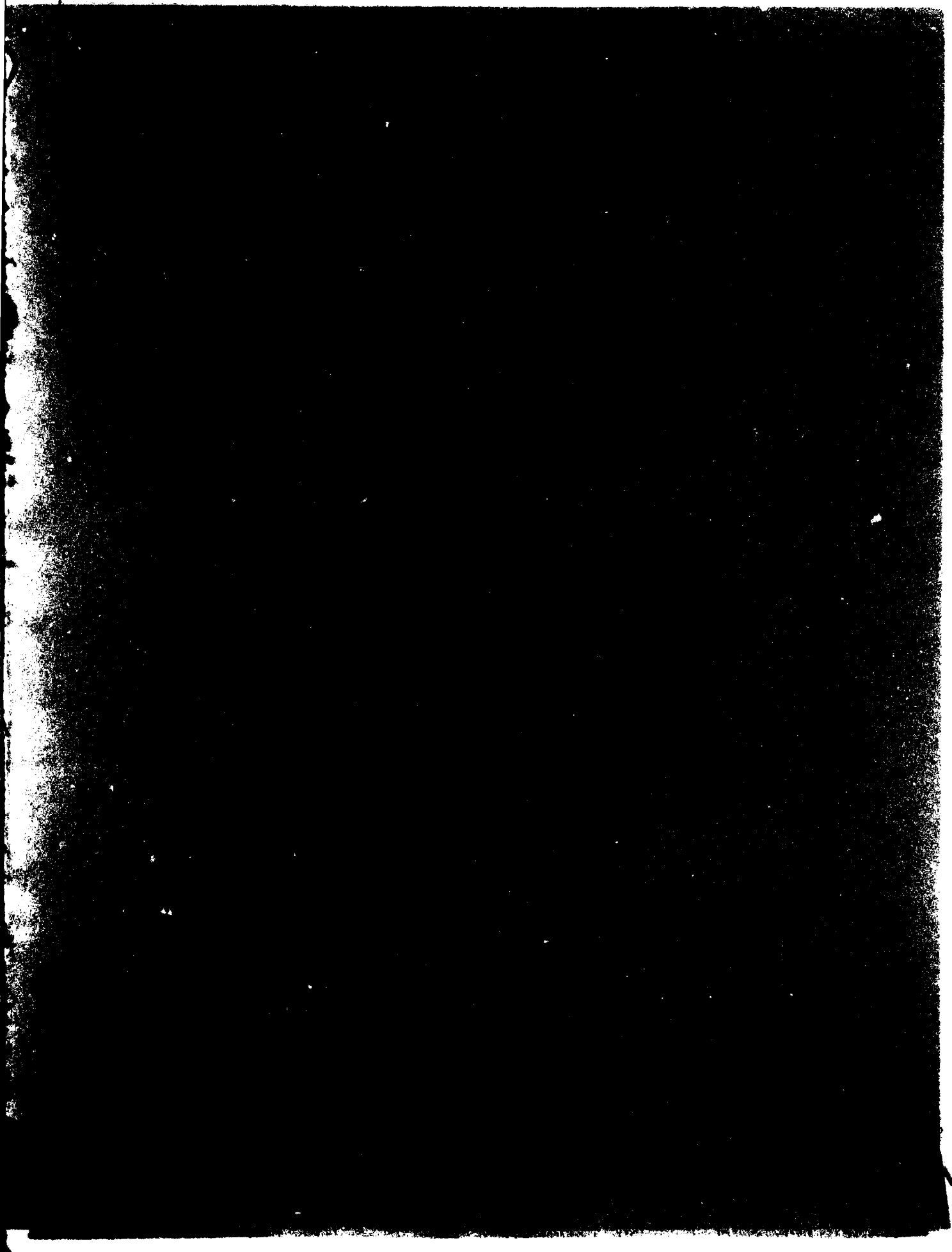
ALU	Arithmetic Logic Unit
BIT	Built-In Test (in MI module)
C	Programming Language used for the CCP software
CA	Context Cache Arithmetic Module
CCP	Computer Control Panel
CDRL	Contract Data Requirements List
CPU	Central Processing Unit
DIP	Dual In-line Packages
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
EA	Extended Arithmetic Module
ECAD	Electrical Computer Aided Design
EXEC	Executive
FAST	Fairchild Advanced Schottky TTL
FIFO	First-in, First-out
FPLA	Field-Programmable Logic Array
GFE	Government-Furnished Equipment
HOL	High Order Language
HSDF	Host Software Development Facility (VAX-11/780)
IE	Instruction Execution Module
IF	Instruction Fetch/Reformat Module
I/O	Input/Output
IOC	I/O Controller; I/O Peripheral Controller mode (for PPP)
IP	Instruction Processor
IPL	Initial Program Load
IR	Instruction Reformat
ISA	Instruction Set Architecture
ISPS	Instruction Set Processor Specification
ITF	IPL Test Firmware
KIPS	Thousands of Instructions Per Second
LSB	Least Significant Bits
MCF	Military Computer Family
MHz	MegaHertz
MI	Maintenance Interface Module (part of MP Element)
MP	Microprogrammable Processor element (has one each MI and US)
MSI	Medium-Scale Integration
RADC	Rome Air Development Center
RAM	Random-Access Memory
ROM	Read-Only Memory
SECDED	Single Error Correction/Double Error Detection
SFT	System Function Test
SOW	Statement of Work
SSI	Small-Scale Integration
TTL	Transistor-Transistor Logic
UASSM	Microcode Assembler
UCMD	32-bit Microcommand
ULINK	Microcommand Link Editor
UNIX	Operating System that runs on the CCP
US	Microsequencer Module (part of MP element)

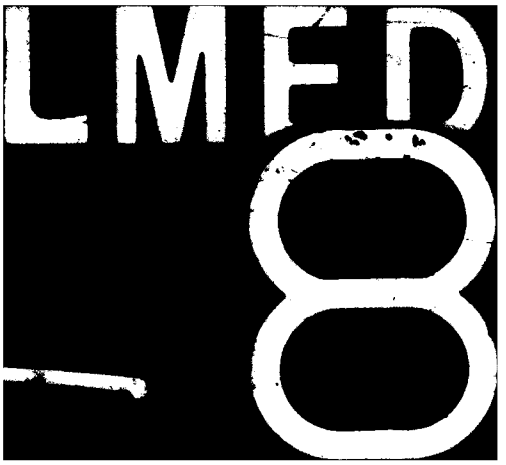
USIM	Microcommand Simulator
VLSI	Very Large Scale Integrated Circuit
1553B	Time-multiplexed data port

A brassboard is an item used for experimentation or tests to demonstrate the technical feasibility of a design and to determine its capability of achieving performance requirements.

An emulator is that part of the firmware for a computer which controls the hardware in a manner that allows execution of a specific ISA.

Firmware consists of two types of microcode: volatile and non-volatile. Volatile code is downloaded from disk storage into random access memory each time the system is initialized. Non-volatile code is burned into programmable read-only chips during fabrication and is not subject to dynamic alteration. The latter, together with burned gate arrays and other logic devices, may be thought of as hardware state sequencers; but the burned code must first be assembled, just as is the volatile code.




LMFD
8