**Naval Research Laboratory**
Washington, DC 20375-5000

**NRL Report 9088**

# A Framework for Evaluating Computer Architectures to Support Systems with Security Requirements, with Applications

CARL E. LANDWEHR AND BRIAN TRETICK

*Computer Science and Systems Branch*
*Information Technology Division*

JOHN M. CARROLL

*Department of Computer Science*
*University of Western Ontario*
*London, Ontario*
*Canada*

PAUL ANDERSON

*Systems Engineering Division*
*Space and Naval Warfare Systems Command*
*Washington, DC*

November 5, 1987

SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited. |
| 2b DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>NRL Report 9088 | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Naval Research Laboratory | 6b. OFFICE SYMBOL *(If applicable)*<br>Code 5590 |
| 7a. NAME OF MONITORING ORGANIZATION | |
| 6c. ADDRESS *(City, State, and ZIP Code)*<br>Washington, DC 20375-5000 | 7b. ADDRESS *(City, State, and ZIP Code)* |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION | 8b. OFFICE SYMBOL *(If applicable)* |
| 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER | |
| 8c. ADDRESS *(City, State, and ZIP Code)* | 10. SOURCE OF FUNDING NUMBERS |

| PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO. |
|---|---|---|---|
| 64574N | X0911 | | |

11. TITLE *(Include Security Classification)* A Framework for Evaluating Computer Architectures to Support Systems with Security Requirements, with Applications

12. PERSONAL AUTHOR(S)
Landwehr, C. E., Tretick, B., Carroll, J. M., and Anderson, P.

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT *(Year, Month, Day)* | 15. PAGE COUNT |
|---|---|---|---|
| Final | FROM 1984 TO 1987 | 1987 November 5 | 58 |

16. SUPPLEMENTARY NOTATION

| 17. COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB-GROUP |
| | | |
| | | |

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

| | | |
|---|---|---|
| Domain | Computer security | Security requirements |
| Protection | Instruction set architecture | |
| Computer architecture | Hardware evaluation | |

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

This report develops a set of criteria for evaluating computer architectures that are to support systems with security requirements. Central to these criteria is the concept of a domain, here interpreted as a set of information and authorizations for the manipulation of that information in a computer system. Architectural requirements are grouped in three categories: logical structure, the processing of logical structures, and physical structure. Appendixes describe the Navy's AN/UYK-43 and AN/UYK-44, DEC VAX 11/780, IBM 370-XA, Intel 80286, and Honeywell SCOMP architectures within this framework. These descriptions are intended to inform readers about the characteristics of these particular architectures and to provide readers with examples so that they may evaluate other systems relevant to their particular needs.

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>UNCLASSIFIED |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>C. Landwehr | 22b. TELEPHONE *(Include Area Code)*<br>(202) 767-3381 |
| 22c. OFFICE SYMBOL<br>Code 5593 | |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

## CONTENTS

# A FRAMEWORK FOR EVALUATING COMPUTER ARCHITECTURES TO SUPPORT SYSTEMS WITH SECURITY REQUIREMENTS, WITH APPLICATIONS

## 1. INTRODUCTION

Many Navy embedded computer systems must satisfy stringent security requirements. To meet these requirements while imposing minimal constraints on the operational environment, system software and hardware must enforce appropriate controls on the system. Experience has shown that building systems to meet these requirements is difficult and that the instruction set architecture (*i.e.*, the abstract machine visible to system programmers) of the chosen computer can be a significant factor in determining whether security requirements can be met [1]. In many cases, systems unable to meet requirements have had to operate under waivers or substitute expensive and inconvenient physical controls and clearance procedures for controls lacking in the system.

The criteria presented here were developed for assessing a computer's instruction set architecture rather than operating systems or application programs developed for it. Programs that operate on a particular machine can to some degree compensate for deficiencies in the instruction set architecture (*e.g.*, by careful design of interfaces), and even the best instruction set architecture can be perverted through inappropriate or sloppy programming. Thus no analysis using this framework can by itself guarantee the security of a system built on a particular computer, nor, conversely, can it prove that a system built on a particular computer could never meet security requirements. Rather, an analysis of this sort should help those who build systems on a particular computer recognize those areas where they must focus their attention in order to assure system security requirements are met.

Building on earlier work [2,3], this report describes the general problem of implementing secure systems and discusses the likely effect of program verification technology on hardware architecture considerations. Next, requirements are proposed for hardware to be used in implementing systems that must meet security requirements. The structures of the AN/UYK-43, the AN/UYK-44, the DEC VAX-11/780, the IBM 370-XA, the Intel 80286, and the Honeywell secure communications processor (SCOMP) are described within this framework in the appendixes. No fundamental architectural problems are identified that would prohibit any of the machines from supporting systems that have security requirements. However, the architectures of the AN/UYK-43 and AN/UYK-44 encourage software designs in which large parts of the system operate in a privileged mode (*i.e.*, they have access to the full machine instruction set and potentially can violate security policies they are intended to enforce). Designers of secure systems based on this hardware should minimize these parts and adhere to software engineering principles in organizing them. The architectures of the commercial machines provide somewhat better capabilities for system designers to restrict execution in privileged mode to those programs that specifically require those privileges.

The examples treated in the appendixes were chosen because they represent machines that are in widespread use or are particularly relevant to military applications. They are included here in order to provide examples that others can follow in applying the criteria to other architectures, as well as to provide useful information about those specific architectures.

Manuscript approved June 26, 1987.

## 2. PROBLEM

Security requirements can be expressed as a set of assertions that a system must enforce. An example informal assertion is, ''No user can display information that has a classification higher than his clearance.'' There are several ways to enforce such an assertion about a computer system:

- all classified information can be removed from the system,
- physical controls can deny access to computer terminals to any user not cleared for all of the information in the system,
- all users can be cleared to the highest level, or
- the controls can be implemented in the computer so that it can compare the classification of specific information with the clearance of a particular user and determine whether or not to display the requested information.

The first three solutions can be imposed on any computer, regardless of its hardware architecture, but they impose substantial constraints on the outside environment. The last solution constrains the computer system, but leaves the environment relatively unrestricted. Solutions of this kind are of primary interest here.

Other security assertions are possible, and lists have appeared elsewhere [4,5]. The question addressed here is, ''What hardware architectures simplify both the enforcement of assertions like these and the demonstration to observers that these assertions are enforced?''

## 3. ROLE OF VERIFICATION TECHNOLOGY

One approach to answering this question is based on techniques for top-down design and program verification. For systems developed using this method, the security assertions would be stated formally as part of the top-level system specification, and as successively more detailed refinements of the design were created, each one would either be proven to enforce the assertions directly or to correspond to the previous refinement, which enforces the assertions. The final, most detailed refinement could be the programs. Alternatively, this approach could be extended to the underlying hardware; parts of the formal specifications for the Honeywell SCOMP correspond to functions implemented by hardware [6].

If all the required security assertions could be proven about the system software, hardware structure might be of less concern. The remaining issues would be whether the compiler generates code correctly and whether the hardware correctly implements the semantics assumed by the compiler, since the programs would have been proven not to violate the security assertions. In theory, even if the underlying hardware provided no more structure than a Turing machine, security violations would be impossible. Some Burroughs systems have relied on a related kind of protection; programs written in assembly language could cause security violations, but software controls allow users to program only in higher order languages. That is, users can only load and execute code that has been compiled by a certified compiler. The security of such a system depends on preventing users from creating their own translators (or coercing certified compliers) to generate insecure assembly language programs. (Wilkinson [7] provides a description, with an example of how the controls failed). In practice, it is difficult to prevent users from generating, via a certified compiler, programs that violate security, because compilers can often be subtly coerced into generating and initiating execution of arbitrary bit strings.

The approach to verification previously described (proving correspondence between specifications and implementation, down to the structures provided by hardware) is not yet feasible for systems of substantial size. Even if one could verify a system's properties down to the chip level, the system would still be open to attacks, such as those that exploit hardware malfunctions, based on gaining access through paths other than the "proven" top level set of functions. Specifications and proofs are also subject to errors, and in the end, increased confidence in the security of a system usually comes from the presence of several independent controls that would all have to fail in order to cause a security violation. Consequently, even with the benefits that program verification can bring, characteristics of a machine's architecture are still important in determining its suitability as a base for building systems that must enforce security assertions.

## 4. REQUIREMENTS FOR HARDWARE ARCHITECTURES TO SUPPORT SECURE APPLICATIONS

Past studies have developed general criteria for secure systems [8,9], and a few have focused specifically on computer hardware considerations [10,11]. The latter work, however, has been based on requirements developed from a specific security model and the notion of a security kernel. The National Computer Security Evaluation Center (NCSC) has developed a set of criteria for trusted computing bases [12] that mentions hardware features but does not address hardware requirements directly. Thus there is no generally agreed upon set of requirements for hardware to support secure systems.

We seek to develop such requirements based on the structure of the problem rather than on a specific security model and implementation approach. *Secret* is derived from a verb that means "to distinguish" and whose components are *se*—"apart" and *cernere*—"to sift" [13]. Thus, secrets represent information that is sifted apart from other information. For a computer to be able to keep secrets, it must be able to separate sensitive from nonsensitive information and cleared users from uncleared. This requirement leads to the concept of *domains* within the computer system. A domain is a set of information and authorizations for the manipulation of that information within a computer system.

There are many different ways domains can be realized within a computer. For example, an instance of an abstract data type can define a domain, since it is a set of objects and a set of operations that are authorized to manipulate those objects [14]. Alternatively, in some systems the concept of process is identified with that of domain: every resource (memory, file, device, processor, etc.) in the system belongs to some process that is authorized to use it without restriction [15]. Although hardware for systems based on the former view of domain may differ from hardware designed to support the latter, it is not necessary to choose among competing interpretations of the domain concept at the level of abstraction addressed here.

Lampson uses the term domain to refer to an entity that has access rights to objects, which could include files, processes, and other domains. These abstractions are rarely implemented directly by computer hardware; the concern here is how to implement domains using the structures hardware commonly provides. For this purpose, we view a computer as providing a *name space,* a *value space,* and a *mapping function* from the name space to the value space, and a set of *instructions* that can affect the mapping function and the name and value spaces. The name space is the set of addresses that a program can generate, the value space is the set of primary storage locations corresponding to these addresses, and the mapping function is defined by the tables or registers (if any) used to translate program-generated addresses (names) to physical addresses (locations). The value space refers only to primary storage; access to secondary storage is determined by the domain's instruction subset, mapping function, and authorizations. For purposes of this analysis, we define a domain to be a subset of the instructions, a subset of the name space, a corresponding subset of the value space defined by a particular mapping function, and a set of authorizations (sometimes called *access modes*) that determine the instructions that the domain may apply to each element of the name space. The mapping function is sometimes referred to as a *context*, since it defines the environment that gives specific meaning to a name.

The following identifies requirements for hardware structures that are to support secure processing. These requirements are grouped according to whether they concern:

- the logical structure of the hardware and its ability to support software structures of secure systems;
- the ability of the hardware to process these structures rapidly enough to be of practical use; or
- the ability of the hardware to resist external physical attacks (*e.g.* eavesdropping, physical removal of media).

The requirements and their categorization are based on the sources already cited and on the history of efforts to develop secure computer systems [1]. While we expect this set of requirements may be augmented or revised in the future, particularly if additional constraints are imposed according to the system to be implemented, we believe that the framework will remain a coherent way to assess the ability of a particular hardware architecture to support security requirements.

## 4.1. Requirements on Logical Structure

The abstract machine visible to system programmers defines the logical structure of the computer. (The instruction set architecture of the machine defines this abstract machine.) There are many ways to realize a given logical structure physically. In this section, we present five kinds of requirements on logical machine structure and then give examples of how each can be met in actual systems.

*Requirements*

### Define and Separate Domains

Any computer system that allows concurrent operations must include provisions for defining separate domains and protecting actions in one domain from improperly affecting those in another domain. Rushby and Randell [16] identify four ways to separate domains: physically, temporally, cryptographically, and logically. Particular computer hardware may simplify one or another of these approaches, but the requirement that the logical structure of the machine provides a way for programmers to define and separate domains is fundamental.

### Establish Initial Domain

The ability to define and separate domains in a machine will be of little use unless there is a reliable way of getting started. This implies that the logical structure of the system must allow the programmer to distinguish the occurrence of a system initialization event and to establish a consistent state for the system—a state from which additional domains can be initiated and separated. If the initial domain provides access to all of the information in the system, great care is necessary in constructing programs that operate in it. Arbitrary programs should not be able to invoke system initialization.

### Link Users with Domains

Security requires that actions be attributable to individuals. This principle leads to a requirement for a link between a user and a domain. So that operations invoked in a domain can be traced to a specific user, the logical structure of the hardware must support the creation and destruction of links between users and domains. This link need not be interpreted as a physical entity; it merely denotes an element of a mapping from users to domains.

### Control Communication Between Domains

In most applications, absolute isolation of domains in impractical—some information must be shared. The logical structure of the machine must support sharing with mechanisms that can limit communication in accordance with application requirements. For example, one domain may need to provide parameters to a function performed in a different domain. The hardware should support passing of these parameters without making additional information available to the recipient.

### Detect and Handle Faults

Finally, the logical structure of the machine must reflect the possibility of machine faults. These can never be eliminated entirely, and they can be a source of security breaches. If the occurrence of hardware errors is concealed from programs, it will be impossible to protect them from the effects of those errors. Consequently, it is required that the logical machine structure include fault detection and fault handling mechanisms.

*Examples*

### Definition of Domains

Techniques for segregating name/value spaces on the same computer include:

*Segregating Spaces in Time*—Programs in early batch processing systems used the same name space (*i.e.*, the entire address space of the machine) but were isolated in time. Programs did not interfere with each other so long as they were not executed concurrently. The operating systems that enforced this condition required mechanisms to keep themselves from being modified by user programs. Swapping systems are also based on this kind of isolation. Execution of different programs can be interleaved, but each program's value space must be saved when its execution is suspended and restored when it is reinitiated.

*Segregating Spaces Through Mapping Functions*—Different programs generate the same names, but all names are interpreted by mapping them to values. If the ranges of the mapping functions are disjoint, programs cannot interfere even if they are executed simultaneously. If the ranges of the mapping functions for two programs overlap, interference (or communication) is possible. Mechanisms for implementing mapping functions include relocation registers, base/bounds registers, and virtual memory mapping registers and tables. The mechanisms that effect the mapping should not themselves be in the name space of user programs (otherwise users can directly alter the mapping). The real address space of the machine need not be the same size as the name space available to a particular domain.

*Segregating Spaces Through Access Permissions*—Different programs can generate names that map to the same value, but may have different access permissions for the value. For example, a program may be allowed to read a certain part of physical storage but not to write it. Whether a given machine associates access permissions with the name space, the mapping, or the value space will affect the flexibility of its protection system. If permissions are associated only with value space and some part of that space is shared, all parties sharing it will necessarily have the same access rights to it. Example of controls that usually associate permissions with the value space include hardware read-only memory and storage keys; controls that can associate permissions with the name space include tags on descriptors or capabilities.

Limiting a domain to a subset of the machine instruction set is accomplished through hardware-defined *modes* of execution. The current mode of execution (*e.g.* "kernel mode," executive mode," "user mode") is defined by the contents of a hardware register and determines whether a particular machine instruction is allowed. On some machines, the mode also imposes constraints on the name and value spaces for the domain. At least two modes are required to control access to the map of names to values—otherwise, the instructions for altering mapping tables would be available to all programs and there would be no way to prevent any program from altering its own mapping table (or others) arbitrarily. Several other functions are normally controlled in this way, including the ability to initiate input/output (I/O), handle interrupts and traps, and halt the machine. The instruction subsets defined by different modes are usually hierarchical, so that each more privileged mode includes all the instructions of its predecessors.

Table 1 exhibits some of the differences in the way domains are defined by several current commercial systems and two Navy standard embedded computers.

### Establishing and Initial Domain

On some computers, the initial domain is established by loading a state word from a predefined location in response to a specific interrupt. This stateword places the machine in its most privileged mode and defines the location from which the next instruction (normally the start of a bootstrap loader) is to be fetched. Current medium scale and larger systems frequently include a separate processor to handle the system console and other utility functions, including system initialization. Initialization can include loading the microcode for the main processor itself as well as establishing its initial program load.

### Linking Users with Domains

A user initially establishes a link with a domain through an exchange of passwords or some other form of authentication data. The user must authenticate the machine as well as vice versa. The user may be provided with the serial number of the processor, or he may arrange for an exchange of questions and answers with a program operating in an initial domain to confirm that he is accessing the intended machine.

Once the link is established, if a domain initiates a new domain in response to a user request, the new domain is accountable to the same user. An exception occurs when the initial domain (accountable to the system administrator) is executing the log-in procedure, and once a new user is authenticated, a new domain is established for which the new user is accountable. Links can be destroyed when the domain ceases to exist, or if accountability for the domain passes to another user. The establishment and maintenance of these links are primarily under software control, but their correctness depends on the hardware path between user and machine. Although this path can be made resistant to physical tampering, the strongest nonphysical protection is provided by encryption.

Some hardware devices periodically reauthenticate the user to assure that the user has not changed (*e.g.*, if one user leaves his terminal running and is replaced by another user), and software

Table 1 — Examples of Domains in Specific Machines

| Machine | Instruction Subset Size | Name Space Size[a] | Mapping Mechanism | Mapping Granularity | Hardware-Enforced Access Modes on Granules | Value Space Size[b] |
|---|---|---|---|---|---|---|
| VAX 11/780 | Unprivileged: 298<br>Privileged: 5 | All Privilege Modes: $2^{32}$ | Page tables | 512-byte pages | 15 modes: no access, read-only, read/write for each of 4 privilege levels, with nesting[c] | All Privilege Modes: $2^{23}$ |
| AN/UYK-43 | Task Mode: 159<br>Executive Mode: 56 | Task Mode: $2^{19}$<br>Executive Mode: $2^{32}$ | Segment registers | 1-word to 64K-word segments | 16 modes: read, write, execute, allow indirect addressing, independently settable | Executive: $5 \times 2^{19}$<br>Task: $2^{19}$ |
| AN/UYK-44 | Task Mode: 257<br>Executive Mode: 37 | Task Mode: $2^{16}$<br>Executive Mode: $2^{22}$ | Page address registers | 1024-word pages | 8 modes: read, write, execute; independently settable | Executive: $2^{20}$<br>Task: $2^{16}$ |
| IBM 370-XA | Unprivileged: 158<br>Semiprivileged: 11<br>Privileged: 39 | 370-XA Mode: $2^{31}$<br>System/370 Mode: $2^{24}$ | Segment and page tables | 4K-byte pages | Key controlled (read/write), page store, and low address protection | 370-XA Mode: $2^{31}$<br>System/370 Mode: $2^{24}$ |
| INTEL 80286 | Real Address Mode: 168<br>Protected Virtual Address Mode:<br>Unprivileged: 155<br>Trusted: 11<br>Privileged: 8 | Real Address Mode: $2^{20}$<br>Protected Virtual Address Mode: $2^{30}$ | Local and global descriptor tables | 64K-byte segments in Real Address Mode; 1-byte to 64K-byte segments in Protected Virtual Address Mode | Privilege level protection: read-only, read and write, read and execute, and execute-only | Real Address Mode: $2^{20}$<br>Protected Virtual Address Mode: $2^{24}$ |
| SCOMP | Unprivileged: 156<br>Privileged: 6 | All Privilege Modes: $2^{20}$ | Descriptor tables | 2048-word segments or 128-word pages | Discretionary (read, write, execute) and privilege levels (ring brackets) | All Privilege Modes: $2^{20}$ |

[a] Number of distinct addresses a program operating in the specified mode can generate without intervention by programs operating in a more privileged mode. These figures are for machines configured with maximum primary and secondary storage.

[b] Maximum number of distinct primary storage addresses to which names can be mapped without intervention by programs operating in a more privileged mode. Bounded by size of name space and quantity of primary storage that can be attached to a processor, these figures are for machines configured with maximum primary storage. Units for the VAX 11/780 are 8-bit bytes; for the AN/UYK-43, 32-bit words; for the AN/UYK-44, 16-bit words; for the IBM 370, 8-bit bytes; for the INTEL 80286, 8-bit bytes; and for the SCOMP, 16-bit words.

[c] Granting an access mode to one privilege level implies that mode is granted to all higher levels as well. For example, if the supervisor mode is given read permission, then both the executive and kernel modes are implicitly given read permission (they may also have write permission).

may be designed similarly. A hardware mechanism for implementing watchdog timers can help enforce periodic reauthentications by software.

### Controlling Communication Between Domains

Communication among domains can occur in several ways:

- one domain may request service from a less-restricted one,
- two domains may share part of the same value space directly, or
- domains may pass messages to each other.

Unless links are set up when the domains are created, the first kind of communication is a prerequisite for the other two: for two isolated domains to initiate address-space sharing or exchange messages, one of them must request that service from a domain that can alter mapping functions or pass messages.

Service requests can be decomposed into three parts:

- invoking the more privileged domain and passing arguments,
- servicing is performed by the privileged domain, and
- returning arguments and control to the requesting domain.

Protected entry points (''gates'') and ''supervisor call'' interrupts allow a privileged domain to control its invocation. Where hardware provides a hierarchy of three or more domains, it can limit the number of levels an individual request can traverse. (Multics [17] introduced the concept of *rings* as a generalization of privileged mode that defined a linear ordering of up to 16 domains. In this scheme, a ring number is associated with each process and defines its privileges: each ring inherits the privileges of all higher numbered rings. A *ring bracket* is an ordered pair of ring numbers that can be associated with a region of memory (in Multics, a segment) and is used to determine whether external requests to read, write, or execute portions of the segment will be permitted. A requested access is permitted only if the ring number of the requesting process is within the interval defined by the ring bracket of the target segment.) An addressing mode that limits the privileged domain to the access rights of the requesting domain can help prevent the privileged domain from performing unauthorized operations on behalf of the requester. Such a mode is unnecessary in capability-based systems, since possession of a capability implies the right to use it. (A capability may be thought of as a ticket that includes both the address of an object and a set of permissions that the presenter of a capability is authorized to exercise. In a capability-based system, one process may pass a capability to another in order to have some desired operation performed on the object the capability addresses [18,19].) Mechanisms to protect a requesting domain from being entered at an improper location on completion of a request are rarely implemented, since the more privileged domain is normally considered benevolent and reliable. This lack leaves the requesting domain vulnerable to having control returned to an arbitrary location within itself, yielding unpredictable results.

When two domains share part of address space, changes made by one domain are directly visible to the other. Control of address-space sharing is aided by access-mode tags on parts of the mapping function. For example, one-way communication between domains is possible if the shared space is limited to read-only in the receiver's map but is read/write or write-only in the sender's. Ring brackets can also be used to limit the ability of different domains to write in a shared space.

Communication via messages is less direct, so the need for authentication is greater. Mechanisms that can label a message with an unforgeable tag identifying the sending domain allow the receiving domain to authenticate its source. Encryption techniques can be applied to this problem, though they rarely have been within a single machine, and the use of encryption will likely require solving a complementary key distribution problem.

### Fault Detection and Handling

Hardware is subject to faults, and unless these are detected and compensated for, security can be compromised. Some hardware mechanisms for fault tolerance, such as error detection and correction codes on units of storage, can function without software assistance and may not be visible in the logical structure of the machine. Nevertheless, conditions will remain in which machine-detectable errors cannot be recovered without software assistance.

The logical machine structure should accommodate techniques for detecting, signaling, and recovering from machine errors. Detection methods are based on redundancy; a stored check-sum is compared with a newly computed version, a critical value is checked against stored boundary conditions, or the same result is computed in two different ways. Programs that perform these functions may require access to particular domains (*e.g.* the ability to read certain programs or storage locations), but do not impose specific requirements on logical machine structure. Once an error is detected, a safe method to enter a domain where the error can be handled (perhaps by stopping the machine) is needed. Standard interrupt and trap mechanisms can be used to handle this problem. The error-handling domain may have access to special machine instructions for diagnosing problems or reconfiguring the system. Because of these special privileges, special care is usually required in designing and implementing error-handling domains.

## 4.2. Requirements for Efficient Processing of Logical Structures

In addition to supporting appropriate logical structures, the underlying hardware must be able to process those structures expeditiously. Several efforts to develop secure systems have attributed their performance problems to the inability of underlying hardware to switch rapidly between domains. In operational systems, if security checks cannot be implemented efficiently, they are likely to be omitted. This section describes requirements based on the logical structures discussed above and gives examples of mechanisms intended to satisfy them.

*Requirements and Examples*

### Creating and Destroying Domains

Systems with stringent security requirements are likely to require a large number of small domains. Consequently, the time and storage required to create or destroy a domain are important. Relevant aspects of machine structure include the addressing structure, which determines the smallest amount of storage that can be allocated to a domain, the size of the system state, which determines how many different registers and storage locations must be initialized, and the availability of specialized instructions to clear residues and manipulate system state information. If creating and destroying domains is time-consuming, sometimes software can be organized so that the set of domains is static.

## Switching Domains

The speed with which the processor can suspend execution of a program in one domain and initiate execution of a program in a different domain is important for the same reasons listed in the previous paragraph, but the effects of high overhead in this operation are harder to remedy. Domains can be combined to reduce the need for switching, but this defeats the principles of least privilege and isolation of domains. Mechanisms that facilitate domain-switching include multiple register sets, which reduce requirements for saving and restoring registers, instructions that can save or restore the machine state in a single operation, interrupt and trap mechanisms that automatically initiate a domain-change when invoked, and a machine structure that minimizes or eliminates residual information that must be cleared when a domain switch occurs.

## Moving Information Between Domains

Hardware must be able to pass information efficiently between domains while endorsing security constraints. Hardware that could automatically check the security levels associated with domains and data to be passed between domains has been proposed but not implemented. Nevertheless, a variety of machines have included hardware that can assist in this task. For example, argument validation can be assisted by automatically passing the mode (*e.g.*, supervisor/user) of the sending domain to the receiving domain, rings of protection can be used to control the domains from which information may be passed, and gates can be used to control the locations in a domain to which control can be passed.

## Security Checking on Operations Within a Domain

Security checks that are to be made on each reference to an object within a domain must either be performed on each access with virtually no overhead or else a single check (*e.g*, when a file is opened or a segment first referenced) must suffice for a large number of accesses. Hardware that supports virtual memory (*i.e.*, automates the mapping from name to value space) can be used to check the legality of each reference according to the current mapping function; if the access authorizations (read, write, execute) are associated with virtual memory addresses, these too can be checked on each reference. It is conceivable that hardware could support security level checking in the same way as access authorization checking. Systems have been built that use software checks at the first reference to a segment or file as well. If the check succeeds, the information can then be mapped into the user's domain, and the addressing mechanisms prevent references outside the domain. Base/bounds checks, virtual memory mapping mechanisms, associative stores, and data caches can help satisfy this requirement.

## Moving Information Between Levels of the Storage Hierarchy

I/O may involve moving information within a single domain or between two different domains. It has frequently caused problems in secure system developments. Often the difficulties can be traced to the logical organization of the I/O subsystem, which may not mesh with the organization of addressing within primary storage. Although these are difficulties in the logical structure, they can cause performance and security problems—because of the logical structure, the I/O must be handled by a highly privileged domain, which increases the demand for domain switches and interdomain communication. An I/O structure that uses the same kind of interface to primary storage as the central processing unit (CPU) (*i.e.*, the same mapping mechanism) can limit these problems.

## 4.3 Requirements on Physical Structure

Even if a computer provides the appropriate logical structures for building a secure system and it processes those structures efficiently, it may be subject to physical threats. In this section we enumerate some needs for physical protection and methods for satisfying them.

*Requirements and Examples*

**Prevent Unauthorized Physical Access to the Computer**

Placing equipment in locked rooms, providing guards, and installing authentication systems do not directly affect the system hardware. The hardware itself can contribute to meeting this requirement by providing console, terminal, and cabinet locks, and having covers for control panels and keyboards. Protected usage meters can detect misuse of resources that occurs when physical controls are defeated.

**Prevent Unauthorized Modification of Removable Media**

Some removable media need to be protected against alteration (*e.g.* master copies of programs and data bases). Write-inhibit switches on disk drives and the physical rings on magnetic tapes address this requirement. A tape drive used for collecting audit data can be inhibited against rewinding to assure that the information is not modified after it is written.

**Assure Secure Communication with Remote System Components**

Physical means can be used to secure connections between system components from wiretapping, and encryption can help assure that information that a wiretapper obtains is still protected.

**Prevent Unauthorized Viewing of System Output**

Video display devices can be designed so that they are only visible head-on, to reduce visual eavesdropping. TEMPEST controls also address this requirement, since electromagnetic emanations are a form (albeit unintended) of system output.

**Assure Continuous Service**

Reliability is associated with security for several reasons: (1) hardware failures may defeat security checks, (2) a system is often more vulnerable to penetration while recovering from a failure, since recovery may involve the use of highly privileged domains, (3) failures may be used to obscure penetrations, since logs may not be written properly and removable media may be irretrievably altered by a system crash, (4) when a system fails frequently, operators and managers become insensitive to small changes in behavior that otherwise would give warning of attempted penetration, and (5) managers of unreliable systems, trying to improve system performance, may take shortcuts that diminish system security.

Many hardware features that contribute to security are designed primarily to increase system reliability. Some of them are:

- Redundant devices and modules, especially power supplies and memory modules. Some systems can be reconfigured in case of partial failure. Security measures must not be inadvertently bypassed when the system is automatically reconfigured.

- Protection of storage against power failure
- Coolant leak detectors
- Manual input devices designed to reduce human error (*e.g.* raised separators between control-panel keys)
- Annunciator lamps to warn that protective features have been intentionally overridden (''battle short'').

## 5. SUMMARY AND CONCLUSIONS

Security requirements for a system depend on its functions, the kind of data it processes, the other systems with which it communicates, and the environment in which it operates. There are few specific properties that hardware *must* have if it is to support systems that have security requirements, but we have tried to develop a framework that illuminates those properties of hardware that determine how well it will be able to support such requirements. In constructing the framework, we have tried to avoid assumptions about the specific security properties desired or the specific methodology used to design the system that implements them.

The requirements identified concern the logical structure of the computer, its ability to process those logical structures, and its physical structure. The logical structure should provide mechanisms for

- defining and separating domains,
- establishing an initial domain,
- linking domains with users,
- controlling communication between domains, and
- detecting and handling faults.

Expeditious processing of logical structures requires mechanisms for

- creating and destroying domains,
- switching domains,
- moving information between domains,
- checking the security of operations within a domain, and
- moving information between levels of the storage hierarchy

Requirements on physical structure concern preventing unauthorized physical access to the computer, unauthorized modification of removable media, and unauthorized viewing of system output while assuring secure communication with remote components and continuous service.

In building systems that can meet these kinds of security requirements, it is helpful to have hardware that supports partitioning of software into different domains with restricted capabilities and responsibilities. This property allows software that enforces critical requirements, such as security requirements, to be isolated, protected from accidental or malicious modification, and thoroughly and independently tested for correct operation.

## 6. ACKNOWLEDGMENTS

Thanks to Richard Hale, John Ihnat, and Stan Wilson, all of NRL, for their reviews of this report. Earlier drafts of the main body of this report benefited from technical reviews by Virgil Gligor of the University of Maryland and by David Weiss (then) of NRL. J. Mallonee of the Naval Sea Systems Command arranged for helpful technical reviews of the AN/UYK-43 and AN/UYK-44 sections by Ken Wander, Bud Lowe, Tom Armstrong, and LCDR Kate Paige. H. O. Lubbes of the Space and Naval Warfare Systems provided continuing encouragement and support. We are grateful to them and to the representatives of several of the manufacturers who provided details concerning their equipment and improved our understanding of its behavior.

## 7. REFERENCES

1. C.E. Landwehr, "Best Available Technologies for Computer Security," IEEE *Computer* **16**(7), 86-95 (1983).

2. C.E. Landwehr and J.M. Carroll, "A Framework for Evaluating Computer Architectures to Support Systems with Security Requirements, with Application to the AN/UYK-43 and AN/UYK-44," NRL Technical Memorandum 7590-008B, Nov. 1984.

3. C.E. Landwehr and J.M. Carroll, "Hardware Requirements for Secure Computer Systems: A Framework," Proc. IEEE Symposium on Security and Privacy, 1984, pp. 34-40.

4. C.E. Landwehr, C.L. Heitmeyer, and J. McLean, "A Security Model for Military Message Systems," *ACM Trans. on Computer Systems,* **2**(3), 198-222 (1984).

5. J.K. Millen, "Kernel Isolation in the PDP-11/70," Proc. IEEE Symposium on Security and Privacy, 1982, pp. 57-65.

6. L.J. Fraim, "SCOMP: A Solution to the MLS Problem," IEEE *Computer* **16**(7), 26-46 (1983).

7. A.L. Wilkinson *et al.*, "A Penetration Analysis of a Burroughs Large System," *ACM SIGOPS Operating Systems Rev.* **15**(1), (1981), 14-25.

8. R.B. Blue, Sr. and G.E. Short, "Computer System Security Technology and Operational Experience," TRW Systems, Redondo Beach, CA, Mar. 1974.

9. "Data Security and Data Processing, Vol. 5," Joint Study by IBM Corp., MIT, TRW Systems, and State of Illinois, IBM Report Nos. G320-1370—G320-1376, 1974.

10. L. Smith, "Architectures for Secure Computing Systems," ESD-TR-75-51, Apr. 1975.

11. J. Tangney, "Minicomputer Architectures for Effective Security Kernel Implementations," ESD-TR-78-170, Oct. 1978.

12. "Department of Defense Trusted Computer System Evaluation Criteria," CSC-STD-001-83, DoD Computer Security Center, Ft. Meade, MD, 15 Aug. 1983.

13. *Webster's Third New International Dictionary*, Unabridged (G.&C. Merriam and Co., Springfield, MA, 1971).

14. D.E. Denning, *Cryptography and Data Security* (Addison-Wesley, Reading, MA, 1983), p. 219.

15. B.W. Lampson, ''Protection,'' Proc. Fifth Princeton Symp. on Inf. Sci. and Systems, Princeton U., pp. 437-443, Mar. 1971, reprinted in SIGOPS *Operating Systems Rev.* **8**(1), 18-24 (1974).

16. J.M. Rushby and B. Randell, ''A Distributed Secure System,'' IEEE *Computer*, **16**(7), 55-68 (1983).

17. E. Organick, *The Multics System: An Examination of Its Structure* (MIT Press, Cambridge, MA, 1972).

18. D.E. Denning, *Cryptography and Data Security* (Addison-Wesley, Reading, MA, 1983), pp. 216-219.

19. R.Y. Kain and C.E. Landwehr, ''On Access Checking in Capability-Based Systems,'' IEEE *Trans. Software Eng.* **SE-13**(2), 202-207 (1987).

# Appendix A

# NAVY SHIPBOARD COMPUTERS

## OVERVIEW OF AN/UYK-43 AND AN/UYK-44 COMPUTERS

The AN/UYK-43 and AN/UYK-44 computers are scaled for different kinds of shipboard applications. The former, as an AN/UYK-7 replacement, is suited for relatively demanding applications, potentially including control of databases, large sensor networks, and command and control systems. The AN/UYK-44 is suited for tasks that require less storage and processing speed, such as controlling a single sensor or communication link. Its smaller physical size makes it easier to embed directly in a system it controls.

Requirements for ''multilevel security,'' where the system must be able to protect information at different classification levels from users with different clearances, are likely to occur in large-scale applications using the AN/UYK-43. The AN/UYK-44 might be required to verify or append appropriate classification labels on data arriving over a communication link or to enforce restrictions on the classification level of data transmitted. It might also provide an interface to a sensor or weapons system in which unauthorized disclosure of information is a lesser threat than denial of service.

As the following sections show, both the AN/UYK-43 and AN/UYK-44 support two kinds of domains—task-mode domains, which can be restricted from exercising the full machine instruction set and address space, and an executive-mode domain, which cannot be similarly restricted. Each machine can support many separate task-mode domains, if the programs that run in its (single) executive-mode domain correctly enforce this separation. In systems that must meet security requirements, any program that is executed in the executive-mode domain has the ability to sabotage those requirements; consequently, it will be necessary to assure that only those programs intended to be run in the executive-mode domain do so, and, moreover, that those programs are trustworthy. ''Trustworthy'' means that even though the hardware cannot prevent it from circumventing security controls, a program will not in fact do so.

Both machines also include features intended to speed switching between a task-mode domain and the executive-mode domain: each has duplicate register sets for the different modes and some instructions that should simplify loading and storing multiple registers. Performance of these machines as components of secure systems is difficult to assess in the absence of a more detailed study of a specific application, however.

A tamperproof program that mediates all accesses by subjects (users or programs) to objects (files, devices, users, or programs) is called a reference monitor [A1,A2]. On either of these machines, a reference monitor would have to operate in the executive mode, and, to fulfill the requirements that it be tamperproof and mediate all references, it must include all of the programs that execute in executive mode. Further, since I/O programs initiated by the executive-mode domain at the request of programs operating in task-mode could affect security, all I/O programs must either be checked or generated by the executive-mode domain. These requirements are likely to yield implementations with relatively large quantities of code that must be trusted and hence must be closely scrutinized and thoroughly tested for potential security problems.

## ABILITY OF AN/UYK-43 TO MEET REQUIREMENTS

The AN/UYK-43 is a 32-bit main-frame computer [A3]. In its B-enclosure configuration, it can contain up to two central processing units (CPU), two power supplies, ten memory modules, two input/output controllers (IOC) each containing two processors with a total of 64 channels, a computer interconnection system (CIS), two display control units, and a remote operator's control unit. Up to sixteen B-enclosures can be interconnected. The AN/UYK-43 has a bus architecture, and its processors are microprogrammed.

The hardware provides two basic kinds of operation—one, AN/UYK-7 mode, is designed to execute software written for the earlier AN/UYK-7 computer virtually without changes, and the other, AN/UYK-43 mode, is designed for programs written expressly for the AN/UYK-43 to exploit its capabilities fully. The mode in which the machine is operating at any instant is determined by two bits in the machine's state word (called the Active Status Register (ASR)). These bits can only be set by instructions that are "privileged" in either of the modes. They affect the way addresses are calculated and the legality and interpretation of several machine instructions. The discussion below is limited to operation in the AN/UYK-43 mode. This simplifies the analysis but does not imply that the presence of the two modes on the same machine has no effects on security.

Within AN/UYK-43 mode, there are two submodes: task mode and executive mode (described in the following subsection). In addition to the ASR, the AN/UYK-43 CPU includes a 16-bit P-register (program counter), eight 32-bit arithmetic (A) registers (accumulators), seven 16-bit (B) index (B) registers, eight 16-bit stack pointer registers, and eight 32-bit segment (S) registers, that are of primary concern to those who write programs to be executed in task mode. An additional set of eight A-registers, seven B-registers, eight stack pointer registers, and eight S-registers are provided for each of the machine's four interrupt levels. Operating system programs that operate in executive mode will need to manipulate all of these register sets and also the eight storage protection registers (SPR) and segment identification registers (SIR) that are used to control the access modes user programs have to storage segments. The IOC subsystem, also primarily of interest to operating system implementors, is quite complex.

The analysis of the AN/UYK-43 hardware below follows the order in which the requirements were developed.

### Requirements on Logical Structure

#### *Define and Separate Domains*

There are two instruction subsets defined for the AN/UYK-43: those available in task mode (called nonprivileged) and those available in executive mode, which include both the nonprivileged and privileged instructions. Privileged instructions comprise those that load the segment base and protection registers, manipulate the ASR, communicate with IOCs, read and activate the CPU monitor clock, and maintain interprocessor timing compatibility. In the executive mode, the machine executes at one of four prioritized interrupt levels; some instructions access different register sets depending on the current interrupt level, but the same set of instructions is available to all executive-mode domains.

The name space available to a task-mode domain consists of the register sets previously described and primary storage addresses that can be generated by manipulating those registers. Since instructions for loading the segment registers are privileged, the name space of a task-mode domain is limited by the values in those registers. There are eight segment registers, each of which can be indexed by a 16-bit register, so the name space (in addition to the registers) available to a task-mode domain is $8 \times 64K = 512K$ names.

The value space to which a task domain's name space is mapped consists of 32-bit words. The mapping from name to value space is determined by the values of the segment registers, and operates as follows for operands: a 13-bit instruction operand (displacement) is zero-extended to 16 bits and added to the low-order 16 bits of the index register selected by the 3 b-field bits of the instruction to produce a 16-bit relative address. The relative address is zero-extended to 18 bits and added to the 32 bits of the segment S register selected by the three s-field bits of the instruction to form the final address. Instruction address generation is accomplished by adding the low-order 16 bits of the 20-bit P register (program counter) to the S register selected by bits 19-17 of the P register. For both instructions and operands, the final address is checked against the constraints specified by the SPR register paired with the S-register used in the address generation. The SPR defines a limit address and the modes of access (read operand, store operand, execute instruction, permit indirect addressing, or use executive state base and index registers) permitted for references within this limit. A reference that violates these constraints or that exceeds the limit address causes an interrupt. Since an SPR can restrict access to as little as a single word of the segment, the smallest conceivable task-mode domain has a value space consisting only of the contents of the registers and a single word of storage.

The name space for an executive-mode domain is equivalent to the name space for the entire machine, since the executive mode instruction set includes instructions for manipulating the segment registers. Thus there are $2^{32}$ different names that can be generated in executive mode.

The size of the corresponding value space is limited by the amount of physical memory that can be attached to the machine, currently 2,621,440 32-bit words. (This number may quadruple with the use of higher density memory chips.) A value space equal in size to the name space can be obtained by accessing storage in other enclosures via the Computer Interconnection System (CIS). Address generation for instructions and operands is identical in task and executive modes, except that physically separate sets of registers are provided for each of the four interrupt levels within executive mode, and the SPR constraints do not apply in executive mode. The control memory locations that contain the values of internal registers are also accessible in executive-mode, as is the breakpoint register set (or sets)* and the optional P-history file.

The appropriate way to segregate task-mode domains in the AN/UYK-43 is through the mapping functions. Programs operating in an executive-mode domain must manage storage and assure that segment base and protection registers are loaded properly before initiating programs that operate in task-mode domains and when responding to memory management requests from such programs. As long as these registers are managed properly, task-mode domains can be isolated from each other.

Executive-mode domains cannot be isolated from each other, since every such domain can gain access to every storage location. This interrupt level cannot be altered directly by software, but the contents of the registers loaded on occurrence of the interrupt can be altered and the programs invoked in the occurrence of an interrupt cannot be protected against modification by programs operating in executive-mode. The one exception to this is the code for processing Class I interrupts, which is held in hardware nondestructive read-only memory. So, there can be only one executive-mode domain.

*The breakpoint facility allows detecting when arbitrary physical locations are read, written, or executed or when arbitrary bit patterns are fetched or stored. The instructions to load the breakpoint registers are privileged, and the occurrence of a breakpoint causes a Class II interrupt. Although this mechanism is intended for debugging, it could conceivably by used to enforce some kinds of security constraints.

### *Establish Initial Domain*

To initiate operation of the AN/UYK-43, an operator is expected to press the Stop, Master Clear, and Start buttons in sequence on the Display Control Unit (DCU). From available documentation [A4], it is difficult to determine exactly how the system responds to these button pushes, but what appears to be the intended behavior is listed below.

- The microprocessor that operates the DCU halts the CPU in response to the Stop button.
- In response to the Master Clear button, it propagates a Master Clear signal, which causes the following actions to take place when the processor is in the stopped mode: in the ASR, the compare designator, overflow, bootstrap, interrupt lockout, and compatibility mode bits are cleared, and load base enable, memory lockout inhibit, and Class I-III lockouts are set. The ASR is also set for Class IV register selection, interrupt state, and State IV. Active repeats, interrupt requests, breakpoint control bits, hardware fault, and program fault indicators are cleared, as are cache memory, the P-History file (which is also disabled), the macro I register, and any CPU error message currently displayed by the DCU. The CPU monitor clock is disabled, and the clock count is reset to 1024 Hz. The CPU is placed into macro mode (macro step mode, if it was already in that mode, or macro run mode if it was in any micro step mode). Presumably, the Master Clear will be propagated to the IOCs as well when the system is initialized; their response to Master Clear is analogous to that of the CPU.
- In response to the Start button, the DCU presumably initiates execution at a location in the nondestructive read only (NDRO) memory that contains a bootstrap loader.

Thus if the DCU programs and microprocessor operate as intended, the system can apparently establish an initial domain that has access to all the system resources and can create domains with more restricted access. The contents of NDRO memory will determine whether or not the initial domain in fact functions as intended. The complexity of the fault tolerance and system reconfiguration features of the AN/UYK-43 (and their description [A5] place precise determination of the initial domain of the machine when restarting after a failure of some sort beyond the scope of this analysis.

### *Link Users with Domains*

Operators will communicate with the AN/UYK-43 through its DCU, Remote Operator's Control Unit (ROCU) and the Power/Temperature Panel (P/TP). The P/TP is located in the door of the main enclosure; the DCU and ROCU may be mounted up to 150 cable-feet away. Both the DCU and ROCU can be connected to two independent CPUs and IOCs via serial maintenance interfaces and include built-in microprocessors. These devices are primarily intended to allow operators to initiate system operation, monitor its behavior, and perform diagnostic and maintenance procedures. There are no provisions in the hardware for authenticating the operators of these devices. A program executing in a domain that responds to requests from a DCU must rely on the electrical connectivity of the DCU and on the program running on the DCU's microprocessor to assure that the request it receives corresponds to the buttons the DCU operator has pressed.

Users of the system (as well as sensors and peripheral devices) will gain access to the CPU through an IOC. The link between a user and a domain potentially depends on programs executed by his terminal and by the IOC. Programs executed by the IOC are constructed by the CPU and reside in main memory; the IOC executes them in response to privileged instructions executed by the CPU. There are no significant hardware aids for authenticating users or devices beyond the existence of clocks in the CPU and IOC that might be used to implement watch-dog timers for reauthentication.

Central processors are identified by bits 22 to 20 of the ASR, but this identification merely allows up to eight CPUs to be distinguished; it is not a CPU serial number.

The CIS allows computers in different enclosures to communicate with each other and so it can be part of the path between a user and a domain. Although its operation is described as "transparent to the software," it has the capability to generate requests to external processors and memory and to permit access to local processors and memory, so it could affect the link between a user and a domain. The CIS has two modules: the requestor extension interface (REI), which forwards requests to other enclosures, and the direct memory interface (DMI), which responds to external requests. Each DMI has a factory-wired identifier for itself and for all the CPUs and IOCs in the system; these are used to recognize requests received from external REIs. These requests are limited by the contents of the four sets of DMI memory protection registers. Each set includes an access mode authorization (execute, read, and/or write) and 32-bit lower and upper bound memory addresses. The protection registers can be loaded only via privileged CPU instructions. When a system is started its CIS is brought up in a "locked out" state so that a foreign CPU cannot gain control of it during initialization.

*Control Communication Between Domains*

Task-mode domains communicate with executive-mode domains via interrupts generated either as requests for service or as traps. Communication between task-mode domains can occur through sharing of primary storage and through the intervention of executive-mode domains to pass messages.

There are four prioritized classes of interrupts in the AN/UYK-43; task-mode domains request service from an executive-mode program by issuing the Enter Executive State instruction, which generates the Executive Return class IV (lowest priority) interrupt. This causes the hardware to enter executive mode, in which the entire machine instruction set is available. Parameters can thus be obtained from the calling domain and any necessary communication can be performed. Class III interrupts are those related to I/O processing, while Class II interrupts occur primarily as a consequence of program faults—attempts to execute privileged instructions, exceed memory bounds, and so on. Class I interrupts include loss of power, parity errors, failure of a module to answer, and others. Although different kinds of processing can be initiated in response to different classes of interrupts and the priority mechanism allows nested handling of interrupts, from the standpoint of security there can be only one executive-mode domain.

The extent of primary storage sharing is determined by how the executive-mode memory management software sets the segment base and protection registers. Different task-mode domains may share from a single word up to their entire memory spaces, depending on the contents of those registers. Segment protection registers can also provide one domain read-only access to a region of storage while another domain has read/write access and a third has execute access. Since segment registers are associated with a CPU and not with memory modules, different CPUs can define different access modes to the same physical storage. CPUs in different enclosures can share primary storage via the CIS.

Messages can be passed between task-mode domains directly by the executive-mode domain or indirectly by allowing two domains to share access to secondary storage. Access to secondary storage is available only through an IOC, and the instructions to initiate I/O are privileged.

CPUs, IOCs, and the CIS within an enclosure have independent access to the same storage, so coordination of I/O with memory management is crucial. IOC accesses are governed by two kinds of memory protection. One kind applies to its Command Address Registers (CARs), which memory protection. One kind applies to its Command Address Registers (CARs), which communicate with CPUs, and the other applies to its Command Address Pointers (CAPs), which communicate with peripheral devices. The IOC can protect a segment with any combination of execute, read, or write permission for each of its 6 CARs, and it can protect a segment with any combination of execute, read, or write permission for each of 128 CAPs. The commands that load the IOC protection registers are privileged CPU instructions; the IOC cannot modify its own protection registers. In this sense, IOC programs are constrained in a way similar to CPU programs operating in task mode. IOC operation is discussed further in the subsection on moving information between levels of the storage hierarchy.

Within a CIS, the DMI can protect four segments of memory with any combination of execute, read, or write permission for each of the four remote REIs with which it can communicate (as described in the preceding section). Like those in the IOC, the DMI memory protection registers can only be modified by privileged CPU instructions.

*Detect and Handle Faults*

The AN/UYK-43 includes a variety of mechanisms for detecting and recovering from hardware faults. Parity bits, timers, bounds checks, and other forms of redundancy are used to detect errors. Different mechanisms are applied in each of the functional modules (CPU, IOC, CIS, DCU, and memory). Built-in Test Equipment (BITE) runs continually while the machine is powered up. Detection of a hardware error that cannot immediately be corrected causes a Class I (highest priority) interrupt; the specific kind of interrupt will depend on the fault detected. Class I interrupts cause the Fault Tolerant Systems Reconfiguration Module, which is stored in read-only memory, to be invoked. This firmware attempts to diagnose the problem and to reconfigure the system to bypass failed components.

Parity is checked at all functional module interfaces. Semiconductor memory provides seven parity bits for each 32-bit word to allow correction of single-bit errors and detection of two-bit errors. Core memory provides parity detection of single-bit errors. Parity errors in cache memory cause the failed portion to be by-passed. CPUs have access via the memory bus to memory interface adapters (MIA) that reflect the status of each memory module.

Detection of a momentary power failure may cause an automatic re-boot. Detection of a sustained power failure causes a power tolerance interrupt that allows 250 $\mu s$ to store volatile data. Enough energy is stored in the power supply to retain volatile storage for 130 ms. A battle short switch allows operation to continue even if the machine is overheated.

## Requirements for Efficient Processing of Logical Structures

### *Creating and Destroying Domains*

Creation of a task-mode domain requires the allocation of storage addressable to that domain and the associated system control structures. As described above, the AN/UYK-43 addressing structure provides the user with access to eight segment base registers that can each address up to 64K words. Creation of a domain does not require all of these to be allocated, however, and by setting the segment protection register appropriately, the effective length of a segment can be limited to a single word. If domains are implemented as processes, the minimum storage for an inactive domain will have to accommodate the contents of the following registers: P, ASR, arithmetic (eight), index (seven), segment base (eight), storage protection registers (eight), and stack pointers (eight). In addition, the system will require pointers to keep track of this storage. To destroy a domain, this storage must be deallocated and other activities may be necessary if the domain has been allocated secondary storage or has input/output operations pending.

Creation and destruction of executive-mode domains would involve some additional state information but is likely to be infrequent, since executive-mode domains cannot be isolated from each other.

### *Switching Domains*

To switch from one task-mode to another, an executive-mode domain must gain control, save the complete register set just described, locate the saved registers of the target domain, restore them, set up the 8 segment identification registers, and transfer control to the target domain. If breakpoint registers are in use by one of the task-mode domains, they will have to be saved and restored as well. In the original specification [A6], there was no single instruction that would load or store all of these registers (or even a single set) at once, but subsequent documents [A5] indicate that two instructions are to be added that will permit a designated set of registers, including A, B, S, SP, Q, SPR, and SIR registers to be loaded or stored in a single privileged operation. A designator in the instruction determines whether the task register set, one of the four interrupt sets, or all five sets are to be loaded or stored. The LBMP instruction will load a segment base register and the associated segment protection and identification registers. Since there are separate sets of arithmetic, index, base, and stack pointer registers for each of four possible interrupt levels within executive mode, it should not be necessary to save and restore the executive-mode register state during a domain switch. These additional register sets should also facilitate rapid switching among interrupt levels.* Interrupts initiate an automatic domain change by saving the interrupt status code, saving the current ASR and P register, and loading new values into P and ASR.

### *Moving Information Between Domains*

Two task-mode domains may share information directly if the executive mode domain assigns them overlapping value (physical address) spaces. Different operations may be permitted to each of the sharing domains by maintaining different values in their respective storage protection registers for the shared region. The possible values include execute, read, write, and permit-indirect-addressing.

Moving information between task-mode domains that do not share primary storage requires the intervention of an executive-mode domain. A possible approach is for the sending domain to issue an Enter Executive Mode instruction after setting up the message to be transmitted and the identification of the recipient in some predefined format. A program running in the executive-mode domain would be able to gain access to both the sending and receiving domains and could move the information requested. This kind of communication would require more overhead than the shared storage approach but would allow the executive-mode programs to implement more flexible controls on communication than the storage protection registers permit.

Any program operating in an executive-mode domain can move information between any two domains simply by setting the base registers to the necessary values and copying the data of interest. There are no facilities to allow executive mode programs to restrict their access to arguments provided by user mode programs to that permitted to the calling program.

*There is apparently space in control storage for three spare SPR/SIR register sets. Microcode changes could allow these to provide a separate SPR/SIR set for each of the four interrupt levels as well, eliminating the need to save and restore them on changes to the interrupt priority level.

### *Security Checking on Operations Within a Domain*

The addressing mechanisms already described can be used to enforce security checks if software structures are defined appropriately. For example, security levels could be defined for users, for segments and for domains. When an executive-mode program initializes a base register so that a particular task-mode domain gains read access to data, it could check that the security level of the data covered by that base register does not exceed the security level of the domain, or the user linked to it. Checks could also be made as to whether other segments in the same domain are writable, and whether their security levels are lower than that of the new segment. If these checks are successful, then the segment register is set up for the task-mode domain. Accesses made to that segment will then be controlled by the address mapping hardware.

### *Moving Information Between Levels of the Storage Hierarchy*

I/O is under the control of the IOCs; each IOC contains one programmable and one hard-wired processor. IOCs and CPUs communicate through privileged instructions and interrupts, so programs running in task-mode domains can be prevented from moving information directly from primary storage to a secondary storage device. The IOCs have direct access to all of primary storage, but references are made relative to base and protection registers loaded under CPU control A CPU can communicate with IOCs in remote enclosures via the CIS.

An IOC provides one CAR per attached CPU; there are six CARs in an IOC. There are four CAPs for each of 32 I/O channels; one CAP is associated with each of the following functions: input data, output data, external commands, and external interrupts.

To perform an I/O operation, the CPU constructs a program in main memory to be executed by a particular CAP and signals the IOC to start the operation by executing an Initiate I/O instruction. This causes the CAR for the IOC to be loaded with the main memory address of the I/O program to be executed. On detecting a new address in the CAR, the IOC sets up the appropriate CAP (according to the channel and function requested) to perform the transfer. While the CAP is executing, the CAR can accept additional requests from the CPU.

If security labels are to be reliably associated with data transmitted from main storage to an output or auxiliary storage device, it is clear that the I/O programs generated for the IOCs by the CPU are security-critical. If, for example, a program operating in task-mode supplies an I/O program to an executive-mode domain for execution, that I/O program must be checked to be sure it represents a valid request (from the security standpoint) before the executive-mode domain initiates its execution. Because this checking will be difficult to specify algorithmically, it will probably be necessary for all device drivers to operate in the executive-mode domain. All programs operating in the executive-mode domain have unlimited access to the machine, so the requirement that large quantities of code execute in executive mode increases the problem of certifying that these programs in fact do not violate security.

## Requirements on Physical Structure

### *Prevent Unauthorized Physicd Access to the Computer*

The AN/UYK-43 does not have keyboard, cabinet or terminal locks. All active components have built-in nonresettable usage meters.

### *Prevent Unauthorized Modification of Removable Media*

Prevention of access to removable media is a function of peripheral equipment not addressed here. The system has no specific hardware support for it.

### *Assure Secure Communication with Remote System Components*

Provision of encryption or protected distribution systems is a facility concern. There is no hardware support for it.

### *Prevent Unauthorized Viewing of System Output*

Control panels, which incorporate video display, are recessed, but not for this purpose. The system is not yet TEMPEST approved.

### *Assure Continuous Service*

The AN/UYK-43 includes extensive measures for fault detection and handling, as described previously. Some of these depend on system software to use hardware features properly. Specific features include:

- The AN/UYK-43 hardware can be highly redundant. The type B enclosure can contain up to two CPUs, two power supplies, two IOCs, a CIS, two DCUs, and a ROCU. Sixteen such enclosures can be interconnected.

- The AN/UYK-43 can be provided with any desired combination of magnetic-core (nonvolatile) or semiconductor (volatile) memory. Core memory modules are smaller (32K vs 64, 256, or 512K) and their read-write cycle time is longer (900 vs 450 ns).

- There is a power-tolerance interrupt feature and a 250 μs power down cycle for the CPU to accommodate an emergency power down. Each semiconductor memory module has sufficient energy storage to maintain its contents for 130 ms during a power interruption.

- Although there are no coolant leak detectors on the water-cooled models, both the B enclosures and the smaller A enclosure, which is a direct replacement for an AN/UYK-7, can be air cooled.

- The control panels (DCUs and ROCU) have raised separators between keys to reduce the occurrence of operator error on manual input.

- The battle short, which bypasses the thermal overload protection, has a warning lamp that is lit if it has been activated.

- The AN/UYK-43 is designed to the sheltered controlled environment requirements of MIL-E-16400. It has an operating temperature range from 0° to 50° C and humidity tolerance up to 95%.

## ABILITY OF AN/UYK-44 TO MEET REQUIREMENTS

The AN/UYK-44 is a 16-bit, microprogrammed minicomputer with a bus architecture [A7]. Its instruction set extends and modifies that of the AN/UYK-20 computer; it can emulate an AN/UYK-20 and specified operations of the AN/AYK-14 computer as well. It is available in three physical configurations. As a militarized reconfigurable processor (MRP), it consists of a set of hardware modules called Standard Electronic Modules (SEMs). As a militarized reconfigurable computer (MRC), it consists of a set of SEMs in a cabinet along with a power supply, memory, control and maintenance panel, and cooling system. As a microcomputer development system (MDS), it consists of an MRP housed in a cabinet with memory and power supplies, a keyboard-display, and, optionally, a printer.

This discussion centers on the MRC. We assume the presence of one or more IOCs and nondestructive read-only NDRO memory. We also assume the ''user growth'' option, which provides additional microprogram read-only memory (ROM) for user-defined instructions, is *not* present. Addition of new instructions to the machine would affect the following analysis; any microprograms added would have to be carefully examined to assure they do not sabotage security.

The state word for the machine is held in two 16-bit status registers (SR1 and SR2). Like the AN/UYK-43, the AN/UYK-44 status registers define whether the machine is operating in one of two modes, task or executive (described in the following section). In addition to the state word, the AN/UYK-44 CPU includes a 16-bit P-register (program counter) and 16 16-bit general-purpose registers that are accessible to programs operating in task mode. A second set of 16 general-purpose registers is available to programs operating in executive mode. Programmers who implement operating system functions will need to manipulate the four sets of 64 16-bit page-address (base) registers that implement memory mapping. The IOC subsystem, also primarily of interest to operating system implementors, can use the three page-address register set numbered 0, 2, and 3; the central processor can use all four of the page-address register sets. For debugging, there is a single breakpoint register and an eight-word P-history file. The breakpoint register can only be loaded manually from the system console; programs cannot alter it.

The analysis of the AN/UYK-44 hardware below follows the order in which the requirements were developed.

### Requirements on Logical Structure

#### *Define and Separate Domains*

Like the AN/UYK-43, the AN/UYK-44 defines two instruction subsets: those available in task mode (called nonprivileged) and those available in executive mode, which includes both the nonprivileged and privileged instructions. Privileged instructions include those that load the page-address registers, perform direct addressing, manipulate the status registers, initiate I/O, read and activate the CPU real time and monitor clocks, stop and initialize the computer, transfer data between executive and task-mode general-purpose registers, execute diagnostics, and load the breakpoint register.*

The name space available to a task-mode domain consists of the register sets previously described and primary storage made available by the currently active page-address register set. The

*Breakpoints are less flexible in the AN/UYK-44 than in the AN/UYK-43. Occurrence of a read, write, or execute reference to the address that matches the contents of the breakpoint register causes the MRC to stop.

name space of a task-mode domain at any one time is thus limited to 64 1024-word pages, a total of 65,536 names.

The value space to which a task-mode domain's name space is mapped consists of 16-bit words. The mapping from name to value space is determined by the contents of the active set of 64 page-address registers and operates as follows for 16-bit operands: status register 1 bits 4 and 5 select one of four page-address register sets and bits 15 to 10 of the 16-bit relative address select one of 64 16-bit page-address registers. Bits 11 to 0 of the selected page-address register become the 12 high-order bits of a 22-bit absolute address. Bits 9 to 0 of the relative address become the 10 low-order bits of the absolute address. Bits 14 to 12 of each page-address register specify the access privileges to be locked out on the page: execute, read, or write. (Bit 15 is a ''modified'' bit.) A reference that violates these constraints causes an interrupt. The smallest conceivable value space for a task-mode domain comprises the contents of the registers and a page of storage (1024 16-bit words). The operations that can be applied to this page are restricted by the lockout bits just described; indirect addressing words are exempt from execute lockout.

As in the UYK-43, the name space for the executive-mode domain is equivalent to the name space for the entire machine, since the executive-mode instruction set allows the page address registers (including the lockout bits) to be altered. The maximum number of names (in this case, absolute addresses) that an executive-mode domain can generate is $2^{22}$ or 4,194,304. The size of the corresponding value space is limited by the amount of physical storage that can be attached to the machine, currently 512K 16-bit words of semiconductor memory or 256K 16-bit words of magnetic core memory per cabinet. Expansion cabinets with 512K or 256K of storage, an IOC, and a power supply can be added on the bus. Address generation for instructions and operands is similar in task and executive modes, except that the status registers may select physically separate sets of general-purpose and page-address registers.

The appropriate way to segregate task-mode domains in the AN/UYK-44 is to assign different domains non-overlapping sets of (absolute) page addresses. Programs operating in the executive-mode domain must manage memory and assure that page address registers are loaded with appropriate absolute addresses and lockout bits before initiating programs that operate in task-mode domains and when responding to memory management requests from such programs. As long as these registers are managed properly, task-mode domains can be isolated from each other.

### *Establish Initial Domain*

The initial actions of the AN/UYK-44 when power is applied depend on the state of the console switches and the contents of the NDRO memory. To start the system (''cold start'') an operator moves the Load/Stop switch on the Control and Maintenance Panel momentarily to the Load position. This causes the MRC to execute a Master Clear, select Run mode, and begin executing instructions from NDRO memory address 002. The Master Clear clears the P-register, the status registers, and bits 8-15 of each page register, and sets bits 0-7 of each page register set to its own register address. It also initializes all I/O channels, and generates a bus initialization signal. Clearing SR1 places the computer in executive mode, using general register set 0, causes it to select NDRO memory, page register set 0, and to lock out all interrupts. In this state, the program starting at NDRO address 002 gains control of the machine. This state allows that program to have unlimited access to the machine and its I/O devices, and, potentially, to create domains that are more restricted. The actual program in NDRO will determine whether these domains are in fact set up properly; the NDRO is expected to contain a bootstrap loader starting at address 002 which will load code from a specified device.

If the computer stops because of a power failure, a similar sequence is initiated when power is restored if the Auto Start/Start switch is in the Auto Start position. In this case, however, the Master Clear signal also causes Run mode to be selected, and execution begins at NDRO location 000. This location is expected to contain a jump to the instruction located at the address specified by the contents of memory location 177 octal. Presumably, system software will have placed at this location the address of a restart routine that will attempt to recover from the power failure. Again, from a security standpoint, this routine will have to establish that the domains of the restarted system are properly defined.

### *Link Users with Domains*

Operators will communicate with the AN/UYK-44 through the Control and Maintenance Panel, which is mounted on the door of the cabinet. An operator can access any register or storage location and single-step through any resident program or enter one himself a byte at a time. There is no provision in the hardware for authenticating an operator.

Users of the system (as well as sensors and peripheral devices) will gain access to the CPU through an IOC. The link between a user and a domain is established and authenticated using much the same mechanisms as used in the AN/UYK-43.

### *Control Communication Between Domains*

Task-mode domains communicate with executive-mode domains via interrupts generated either as requests for service or as traps. The privileged instructions Load/Store Inter-Register (LIR/SIR) permit an executive-mode domain to exchange the contents of a task-mode general-purpose register with an executive-mode register. Communication between task-mode domains can occur through sharing primary or secondary storage and through the intervention of executive-mode domains.

There are three prioritized classes of interrupts in the AN/UYK-44; task-mode domains can request service from an executive-mode program by issuing the Executive Return instruction, which generates the Executive Return Class II interrupt, or by simply attempting to execute an executive mode instruction, which causes an Executive Mode Fault Class II interrupt. In processing any interrupt, the hardware causes the executive mode bit to be set, so the entire machine instruction set is available and, consequently, all storage as well. Parameters can thus be obtained from the calling domain and any necessary communication can be performed. Class III interrupts are those related to I/O processing, while Class II interrupts occur also as a consequence of program faults. Class I interrupts are loss of power, parity errors, and failure of a module to answer.

The extent of primary storage sharing is determined by how the executive-mode domain manages the name-value space mappings, as in the AN/UYK-43. Access to secondary storage is available only through an IOC* and the instructions to initiate I/O are privileged, so sharing of secondary storage space among task-mode domains is also under the control of the executive-mode domain. IOCs address primary storage through page-address registers, and these references are subject to the lockout bits associated with those registers. The CP can protect main storage from wayward accesses from an IOC through this mechanism as well as by controlling the programs it requests the IOC to execute.

*A direct memory access (DMA) option is available for the MRC that allows an external controller to read and write main memory without the intervention of an IOC; using this option can defeat the controls the IOC provides.

*Detect and Handle Faults*

The AN/UYK-44 includes less extensive mechanisms for detecting and recovering from hardware faults than the AN/UYK-43. It uses the same basic memory modules as the AN/UYK-43 with the same kinds of parity and error detection/correction capabilities. Unlike the BITE in the AN/UYK-43, the Built-In Test (BIT) in the AN/UYK-44 does not run continuously. BIT is microprogrammed and is invoked by the privileged IM (Initiate Microdiagnostic) instruction or through switches on the Control and Maintenance Panel. An error log is automatically maintained. Unlike BITE, BIT only detects trouble; there is no automatic reconfiguration. However, in case of a parity, resume, or power fault, a Class I (highest priority) interrupt is generated; programs that service these interrupts are responsible for saving or restoring the machine state and treating the reported problem.

Detection of a momentary power failure could cause an automatic ''re-boot'' if appropriate code were stored in the 192-word NDRO memory. Detection of a sustained power failure causes a power tolerance interrupt that allows 250 $\mu$s to store volatile data. A battle short switch allows operation to continue even if the machine is overheated.

## Requirements on Processing of Logical Structures

*Creating and Destroying Domains*

The AN/UYK-44 addressing structure provides the user with access to 64 page-address registers that can each address up to 1024 words. Creation of a task-mode domain requires these registers to be initialized, though they need not all contain different values. If domains are implemented as processes, the minimum storage for inactive domain will have to accommodate the contents of the following registers: P, SR1, SR2, 16 general-purpose registers and the contents of the 64 16-bit page-address registers. In addition, the system will require some additional pointers to keep track of this storage. To destroy a domain, this storage must be deallocated, but other activities may also be necessary if the domain has been allocated other resources, such as secondary storage, or if it has I/O operations pending.

*Switching Domains*

To switch from one task-mode domain to another, an executive-mode domain must gain control, save the complete register set just described, locate the saved registers of the target domain, restore them, and transfer control to the target domain. The (LMR) instruction will load multiple registers from sequential storage locations; its complement is the (SMR) instruction. The (LARM) instruction will load the contents of sequential page-address registers from sequential storage locations, and its complement is the (SARM) instruction. Interrupt processing initiates a domain change by saving the contents of the P, SR1, SR2 and real-time clock (RTC). P, SR1, and SR2 are then loaded from control memory locations that depend upon the class of interrupt. Bit 15 of SR1 is set, putting the computer in executive mode. Because there are additional sets of general purpose and page address registers for use in executive mode, it should not be necessary to save and restore them when switching between domains of different modes.

*Moving Information Between Domains*

Two task-mode domains may share information directly if the executive-mode domain assigns values to their page-address registers that allow their value spaces (physical address space) to overlap. Different operations may be permitted to each of the sharing domains by maintaining different values in their respective lockout bits. The possible values include execute, read, and write.

Moving information between task-mode domains that do not share primary storage requires the intervention of an executive-mode domain, as in the AN/UYK-43. Approaches similar to that described for the AN/UYK-43 apply to the AN/UYK-44 both in this case and in moving information between a task-mode domain and an executive-mode domain.

*Security Checking on Operations Within a Domain*

The addressing mechanisms already described can be used to enforce security checks if software structures are defined in a way similar to that described for the AN/UYK-43, but pages, rather than segments, would be the unit of storage to which labels would be applied. When an executive-mode program initializes a page-address register so that a particular task-mode domain gains read access to data, it could check that the security level of the data covered by that page does not exceed the security level of the domain, or the user linked to it. Checks could also be made as to whether other pages in the same domain are writable, and whether their security levels are lower than that of the new page. If these checks indicate that the appropriate security model has been observed, then the page-address register is set up for the task-mode domain. Accesses made to that page will then be controlled by the address mapping hardware. The hardware assures that the task-mode domain cannot gain access to storage except via a page-address register; the executive-mode programs that load these registers must assure that they are consistent with security requirements.

*Moving Information Between Levels of the Storage Hierarchy*

The IOC controls input/output and, thus, transfers of information between levels of the storage hierarchy. The CPU communicates with the IOC by issuing the privileged (IOCR) (start I/O) instruction. This causes the IOC to begin executing at a specific location. The IOC program selects one of 32 IOC command chains resident in page-address register set 0. The CPU suspends operation until the IOC completes processing the I/O instruction. The Memory Address Expansion option provides a duplicate set of page address registers slaved to those in the CPU in order to reduce contention.

As in the AN/UYK-43, I/O programs generated for the IOC by the CPU are security critical, particularly if security labels are to be stored with data on auxiliary storage devices. Because of the difficulty of analyzing the security implications of user-generated I/O programs, these programs will have to be created by software that can be relied on not to sabotage security requirements. This requirement for intervention by programs operating in the executive-mode could lead to substantial bottlenecks or increased certification requirements in some applications.

## Requirements on Physical Structure

*Prevent Unauthorized Physical Access to the Computer*

The AN/UYK-44, like the AN/UYK-43, does not have keyboard, cabinet or terminal locks, but the CPU does have a built-in nonresettable usage meter.

*Prevent Unauthorized Modification of Removable Media*

Prevention of access to removable media is a function of peripheral equipment not addressed here. The system has no specific hardware support for it.

*Assure Secure Communication With Remote System Components*

Provision of encryption or protected distribution systems would be a facility concern. There is no hardware support for it.

*Prevent Unauthorized Viewing of System Output*

Control panels are recessed to inhibit someone from viewing displays from over the operator's shoulder. The system has been designed to meet standards for TEMPEST but testing has not been complete.

*Assure Continuous Service*

The AN/UYK-44 includes the measures for fault detection described previously. They are less extensive than for the AN/UYK-43. Some of these depend on the operator and the system software to use hardware features properly. Specific features include:

- The AN/UYK-44 hardware can be redundant. Memory expansion modules can be attached to the bus; each is a cabinet containing up to 256K (magnetic core) or up to 512K Semiconductor Memory (SCM) of storage. Memory modules are connected to the bus by Memory Interface Modules (MIMs). Like the MIAs in the AN/UYK-43, the MIMs permit mixing magnetic core and SCM, but they do not allow faulty modules to be bypassed. The power supplies can operate from any of six different kinds of AC sources.

- The AN/UYK-44 can be provided with any desired combination of magnetic-core (nonvolatile) or semiconductor (volatile) memory. Core memory modules are smaller (32K vs 64K) and their read-write cycle time is longer (900 vs 350 ns).

- There is a power-tolerance interrupt feature and a 250 $\mu$s power down (energy storage of SCM) to allow an emergency power down.

- Although there are no coolant leak detectors on the water-cooled models, water-cooling is an option; the standard equipment is air-cooled.

- The battle short, which bypasses the thermal overload protection, has a warning lamp that is lit if protection has been bypassed. If the overload sensor isn't bypassed, it sets off an audible alarm when actuated.

- The AN/UYK-44 is designed to the sheltered controlled environment requirements of MIL-E-16400. It has an operating temperature range from $-54°$ to $65°$ C and a humidity tolerance up to 95%.

## REFERENCES

A1. C.E. Landwehr, "Best Available Technologies for Computer Security," IEEE *Computer* **16**(7), 86-95 (1983).

A2. "Department of Defense Trusted Computer System Evaluation Criteria," CSC-STD-001-83, DoD Computer Security Center, Ft. Meade, MD, 15 Aug. 1983.

A3. "AN/UYK-43 Navy Embedded Computer System (NECS) Technical Description," Sperry Univac, 1982.

A4. "AN/UYK-43 Interim Instruction Set Architecture," Naval Sea Systems Command, PMS-408, Oct. 1983.

A5. E. Organick, *The Multics System: An Examination of Its Structure* (MIT Press, Cambridge, MA, 1972).

A6. J.M. Rushby and B. Randell, "A Distributed Secure System," IEEE *Computer* **16**(7), 55-68, (1983).

A7. "AN/UYK-44 Navy Embedded Computer System (NECS)," Technical Description. Sperry Univac, 1982.

# Appendix B

# DEC VAX-11/780

## OVERVIEW

The VAX-11/780 is a 32-bit computer with a virtual memory space of up to 4G-bytes [B1]. The physical memory can be as large as 64M-bytes, with the rest on mass storage devices. Its processor can execute instructions in either the native mode or compatibility mode. The native mode instruction set is an extension of the PDP-11 instruction set with variable length instructions, some that correspond directly to high-level language constructs. The compatibility mode executes a set of nonprivileged PDP-11 instructions. Both modes are governed by the VAX memory management hardware.

The VAX-11/780 provides four hierarchical privilege modes to control access to memory and execution of privileged instructions. These access modes are (from most to least privileged): kernel, executive, supervisor, and user. Unprivileged instructions can be executed by a process in any access mode, while privileged instructions can only be executed by a process in the kernel mode. Memory is protected at the page level granularity (512-bytes), where a page may be read only, read and write, or no access, for each of the access modes. Any access given to a less privileged mode is implicitly given to all more privileged modes.

There are 16 32-bit general registers. Most can be used as accumulators, index registers, and base registers, some are used implicitly for special purposes, and two are used for the stack pointer and program counter. There are 42 registers in the privileged register space. These registers are either accessible only by kernel mode processes or by the microcode, and are used primarily for address translation, interrupt servicing, and bus controlling. The execution state of the processor is encoded in the processor status longword (PSL). The low-order bits are exception flags; the high-order bits define the access mode information, the interrupt priority level, and the current instruction set.

## REQUIREMENTS ON THE LOGICAL STRUCTURE

### Define and Separate Domains

The name space consists of 4G-bytes of virtual memory, divided into a 2G-byte process space and a 2G-byte system space. The general registers can be accessed in any mode, but the privileged registers are protected from all nonkernel mode processes; the move to and from privileged register instructions are privileged. The physical memory unit initially can contain up to 4M-bytes; it can be increased to 64M-bytes by adding memory controllers and interleaving the memory. The memory management hardware maintains the virtual address space and memory protection.

The virtual address is composed of a 23-bit virtual page number and a 9-bit byte offset. The virtual page number is an index into the page table. Page table entries are composed of a 21-bit page frame number (page physical memory address), a valid bit (V), a modified bit (M), protection information, and five unused bits (reserved for DIGITAL). The page frame number is concatenated with the byte offset to determine the byte physical memory address. (See *Page Table Entry*.)

| | | | | |
|---|---|---|---|---|
| V | Protection | M | Unused (5-bits) | Page Frame Number |

*Page Table Entry*

The process space is divided into two regions: P0 (program region) and P1 (control region). Each user mode process has a page table for each of the P0 and P1 regions. These page tables are maintained in the system's virtual address space (they are not always resident in physical memory). There are also per-process base and length registers for both the P0 and P1 regions. The base registers point to the process's page tables for those regions, and the length registers determine the sizes (number of page table entries) of the page tables.

All processes share a system page table that is located in physical memory. Physical page addresses in the system space are calculated in the same way as those in the process space. The system base register points to the system page table; the system length register determines the size of the system page table.

A domain consists of memory described by a process's page tables, the register set (general and possibly privileged registers), and the instruction set (depending on the privilege mode). User mode domains can be isolated by the operating system if no two processes have page table entries corresponding to the same page. The page table base and length registers are privileged registers and therefore protected from nonkernel mode processes. The operating system must protect the page tables so that only authorized software has access to the entries. This can be achieved by isolating the page tables (giving page table entries that point to the page tables only to authorized processes), and also with the page protection mechanism (for example, setting the protection bits to allow writing only in the kernel mode).

## Establish an Initial Domain

After a power-down or machine halt, the VAX-11/780 is initialized by the bootstrap sequence. The bootstrap sequence can be initiated by either setting switches on the front panel to automatically reboot or entering the boot command from the console terminal. In either case, the bootstrap sequence performs the following series of actions: micro-verify code tests processor data paths, the general and most privileged registers, the cache memory, and the boot path (failure of a test halts the machine and gives a diagnostic error message); the PSL is initialized from ROM (the processor is placed in kernel mode and at a high interrupt level); the bootstrap code is loaded into physical memory and executed. The machine executes the bootstrap code with memory management off. In this state, there are no access checks on memory and no virtual addressing. The bootstrap code must load the map enable (MAPEN) register to turn on memory management. This sequence establishes an initial kernel mode domain that can then create other domains with more restricted access.

## Link Users with Domains

Operators communicate with the VAX-11/780 via the console terminal and switches on the front panel of the cabinet. The processor receives commands from the console through a special interface, preventing other devices from invoking the console's privileged commands. There is a console key switch that can be set to disable communication between the console and the processor.

Users, peripheral devices, and some storage devices communicate with the processor through UNIBUS adapters; mass storage devices communicate through MASSBUS adapters. The bus adapters interface to the synchronous backplane interconnect, which controls I/O and memory transactions. There is no significant hardware mechanism for authenticating users or devices.

### Control Communication Between Domains

Messages can be passed between domains through the intervention of a kernel mode process, either at domain creation or at run-time. Message passing can be accomplished by adding an entry for the message to the recipient domain's page table (and then removing the entry for the sender, if necessary), or by copying data between domains. Sharing can be accomplished if the sharing domains have entries to a common page. Since page protection information is recorded in the page table entries, access rights can be assigned per domain. Since processes running in the system space share the system page table, page protection is the only hardware mechanism that can control access to memory.

When a process calls a more privileged domain, the processor maintains system integrity by controlling the entrance to and the exit from the privileged domain. When a service routine is called, the access mode of the calling process is stored in the previous mode field of the service routine's PSL, allowing the privileged process to determine the access rights of the calling process. The success of any read or write instruction depends on the calling process's access mode, not that of the more privileged service routine. In this way, areas of memory originally protected from the calling process remain protected. At the completion of the service routine, the old processor state is restored and a check is made to insure that the access mode of the process regaining control is equal or less privileged than the access mode of the processor when the privileged routine was called. This means a program cannot increase its privilege by altering the processor state to be restored. Software interrupts and exceptions are handled in a similar manner.

### Detect and Handle Faults

The VAX-11/780 features several mechanisms for fault detection and recovery. In physical memory, error correcting code detects double-bit errors and corrects single-bit errors. Parity is checked on bus transactions (both processor internal and external) and in the cache memory. Timers are included to test time-dependent functions and hung machine conditions. There are several (privileged) maintenance registers that can be examined to help determine the causes of parity errors and bus faults.

On power-up, boot, or reset, microverify routines test internal processor paths (specifically the boot path), the general registers, most of the privileged registers, and the cache memory. Failure of any of the tests causes a machine halt and generates a diagnostic error message. In case of a power failure, a battery back-up unit can maintain the contents of physical memory for up to 10 minutes. If the power failure lasts longer, the memory is over-written with zeros on power-up.

## REQUIREMENTS FOR EFFICIENT PROCESSING OF LOGICAL STRUCTURES

### Creating and Destroying Domains

All information relevant to the hardware context of a process is stored in its process control block (PCB). The PCB contains images of the PSL, page table base and length registers, program counter, general registers, and stack pointers—a total of 24 32-bit words. The process control block base register points to the physical memory system space location of the current process's PCB. To create a domain, a PCB must be created in physical memory and loaded to privileged registers (loading a PCB is handled with a single instruction). To destroy a domain, a system process must take control of the processor and clear the memory image of the domain's PCB.

### Switching Domains

For a process to get control of the processor, the process's PCB must be loaded into the appropriate privileged and general registers. The two instructions that implement domain switching are privileged. The save process context instruction, *SVPCTX*, stores the current PCB from the registers to the system space location pointed to by the PCB base register. The load process context instruction, *LDPCTX*, loads the registers from the PCB pointed to by the new PCB base.

### Moving Information Between Domains

Information can be passed between user mode domains only through the intervention of a system process (either at domain creation time or at run time). That process must modify the page tables of the communicating domains so that some communication is possible (for example, shared pages), or copy the data from one domain to another. This requires a call and context switch to the system process, a modification to the page tables or a message passing function, and a return.

Pages can be given system-wide access (limited by page protection) if they are placed in the system space. Message passing can be accomplished by passing memory pointers to the recipient (assuming the recipient has read permission to the page).

### Security Checking on Operations Within a Domain

Virtual address translation involves a microcode protection check at the page level granularity. Repeated references to memory pages is sped up by the translation buffer, a fast memory that records the results of recent address translations. There are five bits in the page table entry that can be used by software (two are used by VAX/VMS as ownership bits, the other three are 'DIGITAL reserved'), possibly for extended security checks.

### Moving Information Between Levels of the Storage Hierarchy

The upper half of physical memory in the system space is the I/O space. The I/O space contains memory-mapped control, data, and status registers of the I/O drivers. I/O space is accessed and protected like the rest of main memory: non-I/O related processes can be prevented from modifying the contents of the I/O space by assigning appropriate page protection. I/O device protection is assigned per page of devices (512 register locations). Different pages of devices can be given different protection.

## REQUIREMENTS ON PHYSICAL STRUCTURE

- *Prevent Unauthorized Physical Access to the Computer*

  The VAX-11/780 has a cabinet lock. There also is a console key switch that can be set to disable communication between the console and the processor.

- *Prevent Unauthorized Modification of Removable Media*

  There is no hardware support for this.

- *Assure Secure Communications with Remote System Components*

  The console terminal is linked through a special interface to the processor. There is no hardware support for verifying any other remote device.

- *Prevent Unauthorized Viewing of System Output*

  Tempest enclosures are available for certain processors and peripherals.

- *Assure Continuous Service*

  The VAX-11/780 has fault detection and recovery mechanisms as described above. Other features that affect reliability are:

  - In case of partial failure, the operating system can modify the system configuration until maintenance is performed, such as disabling defective memory modules, the cache memory, and nonresponding peripheral devices.

  - An optional battery back-up can maintain 4M-bytes of physical memory for at least 10 minutes. The power controller has sensors that detect power loss and temperature out of the normal operating range.

  - The cabinet is air cooled.

## REFERENCE

B1. Digital Equipment Corporation, 1980. VAX Hardware Handbook.

# Appendix C

# IBM 370-XA

## OVERVIEW

The IBM 370-XA is a computer architecture [C1], implemented in computers such as the IBM 308X, 4381, and 3090. Its 31-bit addressing mode allows for up to 2G-bytes of physical memory in a 2G-byte virtual address space. The computer can be configured with one or more processors, a channel subsystem for I/O processing, I/O devices, and physical and secondary storage units. Two modes of operation are provided: System/370 mode and 370-XA mode. System/370 mode is a 24-bit addressing mode compatible with System/370 operation. 370-XA mode utilizes the extended-architecture features, such as 31-bit addressing, an enlarged instruction set, and a more flexible channel subsystem. There are two states of operation: the problem state and the supervisor state. Users' programs should run in the problems state, while the operating system should run in the supervisor state.

The instruction set is divided into three privilege-based categories: 158 unprivileged, 11 semiprivileged, and 39 privileged instructions. Unprivileged instructions include general computing instructions; semiprivileged include address space translation, subsystem-linkage, and authority control instructions, and, privileged include processor control, I/O, and diagnostic instructions. Privileged instructions can only be executed by processes operating in the supervisor state. Semiprivileged instructions can be executed in the problem state only if certain authority checks pass, and some semiprivileged instructions even require that supervisor state processes pass the authority checks. Semiprivileged instructions are divided into four groups, each with its own required authority check. Therefore, it is possible to allow a process the execution of some semiprivileged instructions and to prohibit the execution of others. Unprivileged instructions can be executed when the processor is in either the problem or the supervisor state.

The register set of each processor includes 16 32-bit general registers, four 64-bit floating point registers, 16 32-bit control registers, a 32-bit prefix register, the 64-bit program status word (PSW), time-of-day clock, clock comparator, and CPU timer. Because the instructions that load the PSW, prefix register, control registers, time-of-day clock, clock comparator, and CPU timer are privileged, processes operating in the problem state can directly access only the general and floating point registers. The PSW contains state and mode of operation, interrupt masking, access permissions to storage, address space, and instruction address information. The control registers contain authority, address translation, program-event recording, tracing, and interrupt processing control information.

Real memory is protected at the page (4K-byte) level. Key-controlled protection is implemented by associating a 4-bit storage key with each page. When a process attempts to write to real memory, the process access key in the PSW is compared to the 4-bit storage access key; access is allowed only if the keys match. The fetch protection bit determines whether the key check must also be made for virtual memory reads. Key-controlled protection also applies to accesses by the channel subsystem.

Access to virtual pages is controlled by page-protection. Each virtual page has an associated page-protection bit that, when set, allows only fetch operations on the page. Otherwise, when the bit is not set, both fetch and store operations are allowed.

Low address protection prevents system information from being modified by unauthorized processes. It applies to effective (virtual) addresses from 0 to 511 decimal. These addresses correlate to a processor's first real page, and contain interrupt service and initial program load information. Low address protection is enabled by a bit setting in a control register, and applies to processes operating in both the problem and supervisor states, but not to processor and channel subsystem accesses.

An access to memory is permitted only if all three of these controls permit the access.

## REQUIREMENTS ON THE LOGICAL STRUCTURE

### Define and Separate Domains

Memory is partitioned into 1M-byte discrete segments and further into 4K-byte discrete pages, both assigned on integral boundaries. The smallest addressable unit is the 8-bit byte. In System/370 mode, the 24-bit address allows for a name space and value space of 16M-bytes. The absolute addresses in this mode point to the first 16M-bytes of memory. 370-XA mode allows for a 2G-byte name space and value space. Virtual addresses are translated into real addresses by dynamic address translation (DAT), and real addresses are translated into absolute addresses by prefixing. Prefixing allows each processor in a multiprocessor configuration to have its own first page (real addresses 0 to 4095 decimal) for interrupt save area and other control information, minimizing interprocessor interference. Real addresses are filtered through the prefix register. When the real address page number is zero (corresponding to the processor's control area), the prefix is added to the address, thus mapping the processor's first page to a page other than the first absolute page.

When DAT is disabled, addresses generated by processes are considered to be real addresses, and the current prefix value determines the relation between real and absolute memory.

When DAT is enabled, addresses generated by processes are considered to be virtual addresses. Virtual addresses are translated into real addresses by hardware through a series of table look-ups. The address space number (stored in a control register), determined by two address space tables, in turn determines the process segment table, page table, and authority table designations. Once these designations are initially determined for the address space and the control registers are loaded (by hardware) with the necessary address control information, virtual address translation is performed by the hardware using a segment table and a page table look-up. Each process has a primary and a secondary address space, as defined by primary and secondary segment tables. The segment table's origins and lengths are loaded into control registers.

Virtual addresses contain an 11-bit segment table index, an 8-bit page table index, and a 12-bit byte offset. The segment table index selects the segment table entry. The entry contains a page table origin and length. The page table index selects the page table entry, which contains a 19-bit page-frame real address. The byte offset is concatenated with the page-frame to develop the 31-bit real address.

| | | | |
|---|---|---|---|
| Page-Table Origin (25 bits) | I | C | Page-Table Length (4 bits) |

*Segment Table Entry*

The invalid (I) bit determines whether the segment can be used, and the common (C) bit is set for shared segments for use by the translation look-aside buffer (a hardware mechanism used to speed successive address translations).

| | | |
|---|---|---|
| Page-Frame Real Address (19 bits) | I | P |

*Page Table Entry*

The page table entry also includes a page protection (P) bit. If set, store access to the page is prohibited.

A domain consists of memory described by the primary and secondary address spaces, and the instruction set determined by the operation state and any authority granted by settings in the control registers.

Address space control is determined by settings in the PSW, prefix, and control registers. These registers are directly accessible only by supervisor-state processes. A supervisor-state process can change the prefix and control registers to access any absolute address. These programs must be written so as not to corrupt the address translation process or memory values.

Problem-state domains can be overlapping or disjoint. An overlap can occur at different levels: domains can share segment tables, page tables, or individual pages. Any overlap occurs in integral units of 4K-byte pages. Memory protection is implemented at the page level (both virtual and real pages). Access to virtual pages is controlled by page-protection. When a store to real memory is attempted, the processor compares the process access key to the storage access key: store access is allowed only if the keys match. The PSW key mask (control register 3, low half) is used to control what keys a process can use with the move instructions that supply an access key as an operand. When the fetch protection bit on a page is set, fetch accesses are also key-controlled. The operations that set storage and process access keys are privileged.

## Establish an Initial Domain

The IBM 370-XA has four mutually exclusive processor states: the operating, load, stopped, and check-stopped states. When in the operating state, the processor execution is described by the PSW and control registers. The processor enters the load state to perform the initial program load. In the stopped state, the processor does not respond to any instructions or interrupts except a restart interrupt. The check-stopped state is entered as a result of certain machine faults.

The IBM 370-XA is initialized with an initial program load (IPL) sequence. The operator begins the sequence by specifying the IPL input device and then activating a load key on the control panel. The load operation resets the processor and channel subsystem, puts the processor in the load state, resets any other processors in the system, and starts the IPL channel program with the supplied channel-command word. When the IPL operation has ended successfully, the new PSW is loaded from absolute addresses 0 to 7 and execution continues at the location specified by the program counter in the PSW. If any machine checks are generated or the IPL is not successful, the processor remains in the load state.

The initial domain is determined by the outcome of the IPL and the new PSW. The initial program and the operating system start-up routine are responsible for leaving the processor in a secure state.

## Link Users with Domains

Operators communicate with the IBM 370-XA through the console terminal and the control panel. The control panel includes the following (model-dependent) controls and indicators: memory and register display and alteration, operation mode, machine state, load operation, power, and system reset. The console terminal may implement some or all of the control panel facilities. The domain access through the panel and console consists of the entire address space: no fetch or store permissions are checked. There are no significant hardware features to authenticated the operator.

Users and devices communicate over subchannels of the channel subsystem. There are no significant hardware features to authenticate users of peripheral devices.

## Control Communication Between Domains

Messages can be passed between problem-state domains only through the intervention of a supervisor-state process (either at domain creation or at run-time). The supervisor-state process can copy the message between domains, or set up overlapping domains by assigning common segment tables, page tables, or table entries. Domains can be disjoint or made to overlap to any degree down to a single page.

Supervisor state processes share a domain that consists of the entire address space and instruction set. Each supervisor-state process can be engineered to operate in a self-restricted subdomain of the system domain. Such a process would not alter address space information unless that operation were part of its task. By arranging a communication protocol, supervisor-state processes can directly communicate between each other's sub-domains.

Problem-state domains can communicate with supervisor-state domains using the supervisor-call instruction. This instruction generates a supervisor-call interrupt and the service routine specified in the instruction code gets control of the processor. The old PSW is saved in real address 32 decimal and the new PSW is loaded from real address 96 decimal.

Program-call and program-transfer are semiprivileged inter- or intra-domain branch instructions. When either of these instructions is executed, the value of the problem-state bit in the new PSW is set to one, indicating operation in the problem-state. This allows called processes to restrict their access to that of the calling problem-state process.

For secure operation, segment tables, page tables, and other address space information should be protected by storage keys available only to authorized supervisor-state processes, and located only in the domains of processes that manage the address spaces.

## Detect and Handle Faults

The IBM 370-XA has several hardware features for detecting and recovering from faults. Error correcting code in the memory system detects double-bit errors, and detects and corrects single-bit errors. Machine checks may be recovered from with a CPU retry, which restores a previously check-pointed state. The channel subsystem has an analysis and recovery procedure for restoring I/O processing after a subsystem error is detected. The IBM 370-XA may also be reconfigured to bypass certain malfunctioning modules and devices.

Interrupts are handled by saving the old PSW and the register set in low real memory (the processor's low address space), and then loading the new PSW for the appropriate service routine.

During a machine-check interrupt, an interrupt code is also available for use in the diagnosis and recovery.

The privileged diagnose instruction can be used to test for the locate faulty components. The trace facility can be used to audit branches, address space changes, and other program events.

## REQUIREMENTS FOR EFFICIENT PROCESSING OF LOGICAL STRUCTURES

### Creating and Destroying Domains

The creation of problem-state domains requires initializing segment and page tables that define the domain address spaces. The domain state is defined by the images of the 8-byte PSW, the 16 general registers, the four floating-point registers, and the 16 control registers. Domains contain an integral number of pages. The destruction of a domain requires that the domain address spaces be invalidated. There is no single instruction that creates or destroys a problem-state domain.

The supervisor-state domain is created by the IPL sequence. Subdomains for supervisor-state processes can be handled in the same manner as are problem-state domains, although these subdomains are less likely to be dynamically created and destroyed.

### Switching Domains

Domain switches can occur as a result of the following: program, I/O, restart, external, and machine-check interrupts, and supervisor-call, program-call, and program-transfer instructions. The interrupt service hardware saves the necessary state information in, and loads the new PSW from, predetermined low real addresses. The operating system must maintain the hardware and software contexts of processes in a multi-tasking environment. The address space translation of the program-call and program-transfer instructions is hardware controlled.

### Moving Information Between Domains

Information may be passed between problem-state domains only through the intervention of a supervisor-state message program (either at domain creation or at run-time). The supervisor-call instruction generates an interrupt, allowing the message program control of the processor. The control transfer is handled by the hardware interrupt procedure. The message program could copy the message between disjointed domains or set up an overlap in the two domains.

Information may be passed directly between supervisor-state processes. The recipient needs to know the location of the message and have read access to the appropriate pages.

### Security Checking on Operations Within a Domain

Address translation includes a storage key/access key comparison, page-protection, and low address protection. The translation look-aside buffer maintains the results of recent address translations, speeding up subsequent references to the same page or segment. There are also eight unused bits in the page table entry that could be used by software to encode information relating to a domain's access permission to that page.

### Moving Information Between Levels of the Storage Hierarchy

The channel subsystem handles all I/O processing. Access to memory by the channel subsystem is controlled by key-controlled protection. The subchannel key is set when the subchannel is programmed for an operation. The processor and the subsystem communicate using privileged I/O instructions and interrupts. Problem-state processes are therefore unable to move information directly between memory and a secondary device. Once an I/O process has been initiated, the processor is freed by the subsystem to attend to other tasks.

## REQUIREMENTS ON PHYSICAL STRUCTURE

- *Prevent Unauthorized Physical Access to the Computer*

  The control panel and console terminal may have locks that disable the functions of those devices. Certain models can be equipped with extra controls and barriers that limit access to various components.

- *Prevent Unauthorized Modification of Removable Media*

  There is no significant hardware support for this.

- *Assure Secure Communications With Remote System Components*

  A cryptographic subsystem is available to interface with some devices.

- *Prevent Unauthorized Viewing of System Output*

  There is no significant hardware support for this.

- *Assure Continuous Service*

  Design features include:

  (1) Redundant hardware may be configured in this system.

  (2) An I/O device may have as many as eight channel paths available to it. The channel subsystem may allocate a maximum of 256 paths. Path management is handled by the channel subsystem.

## REFERENCES

C1. IBM System/370 Extended Architecture—Principles of Operation. SA22-7085-0, IBM, 1983.

# Appendix D

# Intel 80286

## OVERVIEW

The Intel iAPX286 (80286) is a 16-bit microprocessor built into a 68-pin very large scale integration (VLSI) chip [D1,D2]. It can address up to 1G-bytes of virtual memory, with up to 16M-bytes in physical memory. There are two operating modes: real-address mode and protected virtual-address (protected) mode. In the real-address mode, the 80286 operates with a 1M-byte physical address space and is object code compatible with the 8086 and 8088 microprocessors. In the protected virtual-address mode, the microprocessor operates with a 1G-byte virtual-address space. The processor has the same base instruction set, register set, and addressing modes in both operating modes. The protected mode implements four hierarchical privilege levels (PL) that control access to memory and execution of privileged instructions. PL 0 is the most privileged; PL 3 the least.

There are fifteen 16-bit, program addressable registers: four general registers (that can also be addressed as eight 8-bit registers), two index registers, a base register, a flag register, a stack pointer, an instruction pointer, four segment registers (code, data, stack, and extra segment selectors), and the machine status word (MSW). The current operating mode is recorded in the MSW.

The 80286 is a VLSI microprocessor chip. The memory, clock generator, interrupt controller, and bus arbiter and controller are external chips. Systems can be configured for specific applications.

## REQUIREMENTS ON THE LOGICAL STRUCTURE

### Define and Separate Domains

In the real-address mode, privilege levels are not implemented and there are no access controls on physical memory: segments are visible as 64K-bytes of consecutive memory and can be read, written, and executed by any task. There are 168 instructions available in this mode: the 8086/8088 instruction set plus 22 additional instructions for the 80286. The 20-bit real address is generated by adding the 16-bit byte offset to the 20-bit segment base address. Thus the name space is 1M-bytes in the real-address mode. The value space is limited by the 20-bit address: it is also 1M-bytes.

In protected virtual-address (protected) mode, memory is maintained and protected by hardware data structures called descriptors that are stored in the global or a local descriptor table. Information recorded in the 8-byte memory descriptors includes a 24-bit segment base (real memory location), a 16-bit limit field (segment size, from 1 byte to 64K-bytes), and an access rights byte. The access rights byte contains the 2-bit descriptor privilege level (DPL), the present (P) (physical memory residency) bit, the descriptor type (code or data), usage permissions, and an accessed bit (A). The usage permissions and the segment type are encoded in the 3-bit type field.

| | |
|---|---|
| Intel Reserved (16 bits) | |
| Access Rights Byte | Base (high 8 bits) |
| Base (low 16 bits) | |
| Limit (16 bits) | |

*Memory Descriptor*

| | | | | |
|---|---|---|---|---|
| P | DPL | S | Type | A |

*Access Rights Byte*

A segment can only be accessed if a task has a descriptor for it and the task passes privilege and usage checks. The task's current privilege level (CPL) must be (numerically) less than or equal to the DPL for the task to have access. In addition, a segment's use can be restricted to specific operations by setting the usage permission bits in the descriptor's access rights byte: data segments can be read-only, or read and write; code segments can be execute-only, or read and execute.

The global descriptor table (GDT) contains descriptors available to all tasks. Descriptors listed in the GDT offer all tasks identical usage permission to a given segment, but the DPL can still be used to restrict access from tasks running at less privileged levels. Each task also has a local descriptor table (LDT) that contains descriptors to private (and shared) segments. The local descriptor table register selects the LDT, and is loaded by the microcode task switch routine from the task state segment (the hardware context of a task) or can be loaded explicitly. The task register points to the base of the current task state segment.

Other descriptor types are system control descriptors. These include gate descriptors, task state segment descriptors, and LDT descriptors. Gate descriptors include a destination selector, destination offset, argument count, valid bit, DPL, and type (task gate, call gate, interrupt gate, or trap gate) information. Task gates control task switches, call gates provide controlled inter- and intra-privilege level changes in the execution stream, and interrupt and trap gates point to interrupt service routines. Task and call gate descriptors are recorded in LDTs or the GDT; interrupt and trap gates are recorded in the processor's interrupt descriptor table (IDT). Task state segment and LDT descriptors are special system data descriptors that are recorded in the GDT and point to system data structures. They contain the same fields as do data segment descriptors.

| | |
|---|---|
| Intel Reserved | |
| Access Rights Byte | Argument Count |
| Destination Selector | |
| Destination Offset | |

*Gate Descriptor*

A virtual address is composed of a 16-bit segment selector and a 16-bit segment offset. The segment selector contains a descriptor table indicator (LDT or GDT), a 13-bit index into that table, and a selector privilege level. The physical memory address is obtained by the following steps. The selector index is loaded into a segment register. Microcode then copies the selector index, scales it by 8, and adds it to the descriptor table base. This value is the physical memory location of the descriptor for the desired segment. The descriptor is then loaded into a hidden part of the segment register (not directly accessible to any task: it is loaded by the microcode). The segment base is then read from the segment register, added to the offset from the virtual address, and placed on the address bus. The 24-bit address bus allows for 16M-bytes of physical memory in this mode. The name space available is 1G-bytes. Although any task can load the visible part of a segment register directly, if the value loaded does not correspond with a descriptor entry within the base and limit of the descriptor table, a trap will be generated.

The task CPL is determined by the DPL of the currently executing code segment. The DPL is recorded in the hidden part of the code segment (CS) register and not directly accessible to any task. The CPL can only be changed when the microcode loads a new code segment descriptor into the CS register. If the descriptor tables are protected (for example, protected from being modified by tasks not in PL 0) and descriptors are created properly, then a task cannot change its privilege level in an uncontrolled or insecure manner.

There are three instruction subsets in protected mode: unprivileged, trusted, and privileged. The 155 unprivileged instructions can be used by all tasks. The 11 trusted instructions can be used only when a task's CPL is (numerically) less than the I/O privilege level (IOPL) for that task. The IOPL is assigned by the operating system per task, and is recorded in the flag word. Trusted instructions deal with I/O and interrupt management. The eight privileged instructions can be executed only by tasks running with a CPL equal to zero. Privileged instructions are used to manage descriptor table registers and the machine status word and to halt the processor. The protected mode instruction set includes those instructions in the real address mode (although in the real address mode, no instructions are privileged), plus eight instructions for manipulating descriptor table registers and the task register, and for access rights checking.

A domain consists of memory described by an LDT and the GDT, the register set, and the instruction set (the instruction set depends on the CPL and the IOPL). A task is a stream of execution that operates within a domain. Gates can allow a task to switch domains. Domains can be separated by providing LDTs with no common segments. Since access rights to a segment are encoded in the descriptor and not the segment itself, several tasks can have different access permissions to the same segment. The descriptor tables themselves should be accessible only by specific authorized tasks, with descriptors for the tables labeled at DPL 0. Instructions that load descriptor table registers, the task register and the MSW are privileged, preventing modification of these system registers by tasks not operating in PL 0.

**Establish an Initial Domain**

The 80286 is initialized by triggering the RESET line. The MSW, instruction pointer, flag word, and the segment registers are set to predefined values, and the machine becomes active in the real address mode. The CS register is set so that there is an initial 64K-byte segment provided for initialization code. To initialize the protected mode, the task register and descriptor table registers must be loaded with descriptors for the protected mode initialization routine and then the MSW must be loaded to enable protected mode operation.

**Link Users with Domains**

Operators, users, and peripheral devices communicate to the 80286 through external chips. There is no hardware support to authenticate users or devices. A user must rely on the electric connectivity of the processor to memory chips and the I/O device to be assured that the processor is receiving the instructions that he is entering.

### Control Communication Between Domains

There are two structures in the 80286 that control communication between tasks: gates and descriptors. Gates provide point of entry control of all processor control transfers; descriptors can be constructed to allow message passing and memory sharing, or the complete isolation of segments.

Gates provide the capability to transfer control to another task. For a call or jump to succeed, the task must pass privilege checks (CPL vs DPL) using a valid descriptor for the destination.

Message passing and segment sharing can be accomplished by setting segment descriptors appropriately. Messages can be passed by calling a process that can copy data from one domain to another. The process must have a common segment with both the originating (at least readable) and target domains (at least writable). A segment can be given universal access by placing the segment descriptor in the GDT. All tasks will then have the same access rights to the segment and access will be limited only by the task's CPL and the DPL. Tasks can also be made to share LDTs. Thus the cooperating tasks can have access to a group of segments and universal access need not be granted. Again, the CPL and the DPL can limit accesses if necessary. A third method is to have multiple descriptors for the same segment, one for each sharing task. Access rights can then be assigned per task.

The selector privilege level provides a mechanism to prevent a more privileged called procedure from gaining access to data at a more privileged level than the calling task. A task's effective privilege level is the maximum (numerically) of the current privilege level and the selector privilege level. The selector privilege level is also known as the requested privilege level (RPL). The RPL can be generated by any task that generates its own virtual addresses, but service calls can use the adjust RPL instruction to force the RPL of selectors passed as parameters by less privileged callers to the CPL of the caller. This would prevent the service call from accessing segments on behalf of the caller that the caller normally would not have permission to access.

### Detect and Handle Faults

An error detection and correction unit (8206) can be used to interface the physical memory with the processor. It checks all memory transfers, correcting single bit errors and detecting double bit errors. Up to 256 interrupt vectors can be defined using the IDT. The IDT entries are pointers to the proper interrupt service routine. Interrupts numbered 0 - 31 are used for instruction exceptions or are 'Intel reserved.' The remaining 224 interrupts can be defined in any way, even for security error handling. When an interrupt occurs, the current task's instruction pointer and machine state flags are pushed onto the interrupt stack to permit a return to the interrupted task. Besides the interrupt request input (INTR), a nonmaskable interrupt request input (NMI) is included.

## REQUIREMENTS FOR EFFICIENT PROCESSING OF LOGICAL STRUCTURES

### Creating and Destroying Domains

A task state segment (TSS) is a 44-byte hardware-maintained context that, along with the LDT, characterizes a task. It includes an image of the register set, stack pointers and selectors for PL 0, 1, and 2, and the LDT selector. The creation of a domain requires the allocation and initialization of the TSS, LDT, and associated memory. The minimum amount of memory that can be allocated is a 1-byte segment. The destruction of a domain requires the deletion of the TSS from memory and the TSS descriptor from the GDT. There is no one instruction that will create or destroy a domain.

### Switching Domains

As described above, task gates define the tasks to which the current task can branch. All requests for task transition, therefore, require a descriptor table look up. The task switch itself is handled by a single instruction (load task register) that saves the current execution state and loads a new execution state. The back link selector in the new TSS points to the previous TSS, allowing a possible return to a calling task. Task switches are handled in about 22 $\mu$s.

### Moving Information Between Domains

Communication between domains involves manipulating the descriptor tables (either at domain creation or at run-time) or copying data between segments. To share memory, a call must be made to a task that has write access to the descriptor tables, the descriptor tables must be updated, and then control must be returned to the calling task by means of the back link selector. Once tables are set up, communication only involves changing data in the shared segments. Copying data requires that a service task has read access to a segment in the originating domain and write access to a segment in the target domain.

### Security Checking on Operations Within a Domain

In the protected mode, all memory references and gate transitions require a descriptor table look-up. Privilege level, access type, and limit checks are made by the microcode; failure of a check generates a trap.

Segment, task gate, and call gate descriptors have two 'Intel reserved' bytes that could be used by software as security labels. Tasks could have associated clearances stored in the software extension of the TSS. Segments could be assigned classifications to prevent unauthorized access; task gates could be locked unless the task clearance dominates the gate classification. Call gates, used for intradomain branching (eg., to another privilege level) and interdomain branching, could be similarly locked.

### Moving Information Between Levels of the Storage Hierarchy

I/O flows over the 16-bit data bus to peripheral chips. The 64K-byte I/O space is in addition to the physical address space; it is not in the virtual address space of any task. The I/O space is protected and accessed with segment descriptors like other memory. The space is divided into 8-bit or 16-bit ports. A descriptor must be available in the LDT or GDT for any port that a task is to access; devices can be protected on a per task basis. The I/O instructions are trusted instructions; a task may only execute an I/O (or any of the other trusted instructions) if the CPL is (numerically) less than the IOPL. The IOPL is protected from unauthorized modification; the flag word can only be loaded by the microcode.

## REQUIREMENTS ON PHYSICAL STRUCTURE

- *Prevent Unauthorized Physical Access to the Computer*

  There is no hardware support for this.

- *Prevent Unauthorized Modification of Removable Media*

  There is no hardware support for this.

- *Assure Secure Communications with Remote System Components*

  There is no hardware support for this.

- *Prevent Unauthorized Viewing of System Output*

  There is no hardware support for this.

- *Assure Continuous Service*

  Error detection and correction must be implemented by other devices. Software routines must be provided to service any interrupts.

## REFERENCES

D1. iAPX 286/10 High Performance Microprocessor with Memory Management and Protection — Advance Information. Intel Corporation, 1982.

D2. iAPX 286 Preliminary Users Manual. Intel Corporation, 1981.

## Appendix E

## Honeywell SCOMP

### OVERVIEW

The Secure Communications Processor (SCOMP) is a Honeywell DPS 6 minicomputer that has been converted to a certifiably secure communications processor by the addition of the security protection module and changes in the standard processor firmware [E1]. The SCOMP system (hardware and software) was awarded an A1 accreditation by the DoD Computer Security Center in December 1984. This analysis, though, deals only with the hardware elements of the SCOMP; no consideration is given to existing SCOMP software.

The Honeywell 6/75 is a 16-bit minicomputer with a maximum address space of 2M-bytes [E2]. There are four hierarchical privilege levels that are used to control access to memory and execution of privileged instructions. The privilege levels are arranged in a ring structure, with ring 0 (the innermost) the most privileged and ring 3 (the outermost) the least privileged.

The SCOMP instruction set is a modified 6/75 instruction set. Changes in the firmware include the removal of 6/75 instructions to validate access rights (VLD) and activate segment descriptors (ASD), and the addition of SCOMP-unique instructions to control cross-ring movement and argument validation.

The security protection module (SPM) mediates all memory accesses and I/O operations. It takes a virtual address from the processor and translates it using indirect or direct descriptor table look-ups. The SPM consists of three major components: a processor (register Arithmetic/Logical Unit (ALU) and virtual memory interface unit (VMIU)), bus interface (BSI), and descriptor store (DS). The VMIU lies between the processor memory address register and the address bus, and mediates all processor memory requests. The BSI mediates all I/O transactions, and the DS manages the SPM images of the descriptors used in the address translation.

The processor has 26 registers available to programs: seven 16-bit general, seven 20-bit address, a 20-bit program counter, system status, indicator, stack address, mode control, two scientific mode control, and five unused (reserved for future use). The processor execution state is encoded in the status register and contains the current right number, processor identification number (for multi-processor systems), and interrupt priority level. The smallest addressable memory unit is the 8-bit byte; most instructions involve the 2-byte word.

### REQUIREMENTS ON THE LOGICAL STRUCTURE

#### Define and Separate Domains

The SPM translates a 21-bit virtual address received from the processor into the physical memory address, allowing for 2M-bytes (1M-words) of memory (the byte-bit is passed unchanged). The virtual address is translated through a series of descriptor table look-ups. Each process has an associated descriptor base root (DBR) that establishes the set of descriptors the process will use. The

DBR is a four-word data structure: two words select the base (16-bit) and limit (11-bit) for the memory descriptors, two words select the base and limit for the I/O descriptors.

Memory descriptors are also four-word structures. They include access control, usage, type, and base and limit information. The 16-bit base address and the 11-bit limit field describe the location and size of the resource pointed to by the descriptor. The type field specifies whether the descriptor is direct or indirect. A direct descriptor points to data; an indirect descriptor points to another descriptor. Depending on whether the DBR is direct or indirect, a memory reference may use from one (DBR direct) to three (DBR indirect) descriptors to locate the data.

Using descriptor structures, the SPM can view memory as up to 512 discrete segments from one word to 2K-words in length. The descriptors also allow for pages of length from one word to 128 words. The type (direct or indirect) of the DBR and subsequent descriptors determines the interpretation and number of descriptors used.

Access control information is encoded in all descriptors, but only one descriptor's protections apply to a given memory reference. The A bit in a descriptor determines if the access information of that descriptor applies. If more than one descriptor encountered in the translation has its A bit ON, then the access information from the first descriptor with active protection is used, controlling access to the largest resource. All subsequent access information is ignored. If no descriptors are found with the A bit set, the SPM generates a trap.

Access control is implemented through the read (R), write (W), and execute (E) bits and ring brackets in the descriptors. There are three 2-bit ring brackets (R1, R2, and R3) that are used to restrict the access granted by the read, write, and execute bits to processes operating in certain ring numbers (with certain privilege). The values in these brackets correspond to a ring number 0 though 3. The process's effective ring number $R_{eff}$ (usually the process's current ring number $R_{cur}$ as recorded in the processor status register) is compared to the values in the brackets to determine if the access desired is allowed. Write permission is granted if and only if the W bit is ON and $R_{eff} \leq R1$; read permission if and only if the R bit is ON and $R_{eff} \leq R2$; and execute (instruction fetch) permission if and only if the E bit is ON and $R1 \leq R_{eff} \leq R2$. Crossring movement (call) permission is granted if and only if the E bit is ON and $R1 \leq R_{eff} \leq R3$.

| A | R1 | R2 | R3 | R | W | E | Y | DT | Type |
|---|---|---|---|---|---|---|---|---|---|
| Base Address (16 bits) | | | | | | | | | |
| 0 0 0 0 0 0 | | Limit (11 bits) | | | | | | | |
| Usage bits | | | 0 0 0 0 0 0 0 0 0 0 | | | | I/O Status | | |

*Memory Descriptor*

I/O descriptors are similar to memory descriptors. They are four-word structures containing address and access information used by the SPM for I/O mediation. The access control information is the same as that for memory descriptors. There is a 10-bit channel number (device address) and a 12-bit function table base address. A process has control permission of a device if the E bit is ON and $R_{eff} \leq R3$.

Encoded in the status register is the current ring number $R_{cur}$, specifying the privilege level (ring number) in which the current process is executing. Processes running in rings 0 and 1 are termed privileged processes, while those in rings 2 and 3 are termed user processes. There are six privileged instructions: HLT, LEV, RCTN, RTCF, WDTN, and WTCF. (The three privileged DPS 6 I/O commands are not privileged on the SCOMP). Only privileged processes can execute privileged instructions; a trap will be generated if a user process attempts to execute one of these six instructions.

A domain consists of memory described by a process's DBR, the register set, and the instruction set (depending on the privilege level). Domains can be separated using descriptors. The operating system can segregate domains completely or can permit shared memory by assigning ring numbers and DBRs properly. The descriptors and DBR must be protected from modification by unauthorized programs. This can be accomplished by giving only the authorized programs descriptors that point to the DBR table and descriptor tables.

When a process becomes active, its DBR and associated descriptors are copied from the processor to the descriptor store. The SPM has a back-up descriptor cache (BUDC) to contain images of the DBR, and memory descriptors and I/O memory descriptors (memory descriptors used in I/O transactions) used repeatedly.

The status, program counter, stack address, and indicator registers are not directly accessible to processes. They are modified by processor firmware, using values stored in process context areas and descriptor tables. There are also three registers only accessible from the control panel: the memory address, memory data, and instruction registers.

## Establish an Initial Domain

The SCOMP initialization procedure has two components: processor bootstrap and SPM initialization. By pressing a sequence of buttons on the control panel, the halted machine can be put into continuous mediation operation. Depressing master clear, load, and execute buttons in series causes the following to occur: the program counter, mode control, and instruction registers are zeroed; pending interrupts are cleared; the real-time clock and watchdog timer are reset; the interrupt priority level is set to zero; the ring number is set to zero; and the processor enters quality logic tests (QLT).

When the processor starts its QLT, the SPM firmware begins initialization. SPM initialization involves stalling the processor, clearing the descriptor cache, running the SPM QLT, redirecting memory requests to BSI Programmable Read Only Memory (PROM), and loading an initial DBR and descriptor set from the BSI PROM. The processor is then unstalled.

The processor QLT exercises logic paths in the processor, memory, and controllers. Failures are indicated by blinking lights on the panel and on failed boards. The SPM enters no-mediation mode as the processor loads the bootstrap code from processor PROM and executes it. When the processor changes the execution point from the processor PROM to memory, the SPM enters continuous mediation mode.

An initial domain is set up encompassing the memory allocated by the initial PROM DBR. The process ring number is zero.

## Link Users with Domains

Operators communicate with the processor using the control panel. A full panel includes a register display, system control switches and buttons, and condition indicators. There is a panel security lock that disables all panel controls except the power switch. There are no hardware features to authenticate operators.

Users and peripherals communicate with the processor through multiple device controllers over the Megabus. There is no significant hardware support to authenticate users or peripheral devices.

## Control Communication Between Domains

Communication between domains occurs in two forms: memory sharing and controlled transfer of the processor execution point. For memory to be shared, a process must set up descriptors and set access controls, either at domain initialization or run-time, to facilitate the communication. To do this, the process must have write permission to the descriptor tables.

Cross-ring movements are controlled by SCOMP-unique instructions and descriptor access information. The call (LNJR) instruction is used to transfer the processor execution point to an equally or more privileged process; return (RETN) is used by the called process to return control to the calling process; and the argument addressing mode (AAM) instruction permits argument validation by the called process at the privilege level of the calling process.

A process's memory descriptors determine what other processes it may call. If a descriptor for a certain process is present in the domain's descriptor set, then call permission is granted if and only if the E bit in the memory descriptor is ON and $R1 \leq R_{eff} \leq R3$. The only valid entry point into a process is at the zero offset location in the segment. The SPM checks these conditions before allowing the call to occur.

When the call instruction is executed, the SPM validates the caller's access to the called process via descriptor look-ups, verifies that the offset into that resource is zero, and computes the new $R_{cur}$. The return address is stored in an address register (B5) and the ring number of the calling process is stored in a general register (R5). Then the program counter is loaded with the entry point to the called process and execution begins. The called process must preserve the return address in another location if the B5 register is used (failure to maintain this value could result in a return to a different location). The return instruction restores $R_{cur}$ and the execution point to those of the calling process. The SPM allows calls only to rings of equal or greater privilege (equal or lesser ring number), and returns only to rings of equal or lesser privilege (equal or greater ring number).

The AAM instruction allows for argument validation by a called process. The SPM uses the maximum value of the ring number of the calling process and that of the called process to determine $R_{eff}$ in the mediation of the next assembly language instruction executed, validating the caller's access rights to the arguments. The AAM instruction inhibits interrupts until the next instruction has finished executing. If the next instruction is another AAM, a trap is generated.

## Detect and Handle Faults

On power-up and bootstrap, the processor and SPM run logic tests, exercising processor and SPM data paths, registers, memory boards, and device controllers. Failures in the QLTs are indicated by a warning light on the failed board and on the control panel. Memory is equipped with error detection and correction capabilities. Single bit errors are corrected; double bit errors are detected. Parity is checked on all bus transfers.

Traps and interrupts are serviced by processor firmware. When a trap or interrupt occurs, the handler stores a partial processor state in a save area (one per interrupt and trap type). The processor execution point is then switched to the service routine. Service routines operate at $R_{eff} = 0$ (forced by the SPM). Returning from a trap or interrupt restores the processor state from the save area. If the trap is SPM initiated, the SPM fault register (4-words) is also stored in the save area. There are 24 defined traps and 64 interrupt levels.

Memory can be an interleaved combination of magnetic and MOS modules. In case of power failure, semiconductor memory can be maintained with battery units: up to 64K-words for 2 hours per unit.

## REQUIREMENTS FOR EFFICIENT PROCESSING OF LOGICAL STRUCTURES

### Creating and Destroying Domains

The minimum hardware context of a domain (privileged or user) consists of a DBR and images of the 26 registers. To create a domain, the context must be allocated in a context save area. To delete a domain, the context must be removed from the save area. There is no single instruction that creates or deletes a domain.

### Switching Domains

Domain switches can be initiated two ways: by a dispatch from the processor or by using call and return instructions. By using the dispatch function, the processor notifies the SPM that a new process will gain control of execution. The SPM blocks the processor, reads the new DBR off the bus, and invalidates all memory descriptors (I/O memory descriptors are not invalidated) in the BUSC. In the processor, the current context must be saved and the new context loaded. There is no single instruction that performs this type of domain switch.

Cross-ring movements are initiated by the call and return instructions. The SPM performs descriptor look-ups to determine if the current process has access to called process. If access is granted, the calling process' ring number and return address are saved in registers, the program counter is loaded with the new execution point, and execution continues.

### Moving Information Between Domains

Information can be passed between domains if descriptors are constructed (either at domain creation or run-time) to allow the domains to share a memory space. At run-time, this requires issuing a call or domain switch to a message procedure that modifies the descriptor tables in memory, invalidates the SPM images of modified descriptors, and returns to the calling process. The SPM then loads the new descriptors into the descriptors into the descriptor store. A combination of access protection could be used, inhibiting the sender or receiver read, write, and execute permissions as desired, or even removing the sender's descriptor to the message. There is no single instruction that performs a message passing function.

### Security Checking on Operations Within a Domain

All memory references are mediated by the SPM. Access checks are performed during address translation by SPM firmware.

### Moving Information Between Levels of the Storage Hierarchy

The SCOMP supports two forms of I/O: premapped and mapped. Premapped I/O allows the SPM to program an I/O device for direct memory access. After completing address translation and protection checks as it would for normal memory accesses (using the effective ring number and a descriptor of the initiating process), the SPM supplies the I/O device with an absolute address and range of memory. The memory range is limited to memory described by a single descriptor. The

I/O device then accesses the memory with no further mediation by the SPM. The device must not modify the absolute address or range passed to it from the SPM; the SPM must not modify the descriptor until the I/O is complete.

Mapped I/O requires SPM mediation for each memory reference of the I/O device. The SPM supplies the device with a virtual address, and uses the effective ring number and descriptors of the initiating process (stored in the BUSC) in the address translation and protection checks.

Premapped I/O uses less overhead than mapped I/O, but is limited to memory described by a single direct descriptor. Mapped I/O allows for segment and page crossing during the I/O, but each reference requires SPM cache look-ups.

## REQUIREMENT ON PHYSICAL STRUCTURE

- *Prevent Unauthorized Physical Access to the Computer*

  The control panel has a panel security lock. When locked, the panel switches, push buttons (except for the power switch), and register display are disabled. There is no other hardware mechanism to authenticate the operator. There are no usage meters.

- *Prevent Unauthorized Modification of Removable Media*

  There is no hardware support for this.

- *Assure Secure Communications with Remote System Components*

  There is no hardware support for this.

- *Prevent Unauthorized Viewing of System Output*

  The SCOMP has been targeted to meet Tempest criteria. The display on the control panel is visible only from certain angles.

- *Assure Continuous Service*

  The SCOMP has fault detection and handling mechanisms as described above. Specific features include:

- The SPM, processor, memory, and controller boards contain firmware to conduct QLTs on boot strapping and power recovery.

- There is no redundant hardware. If the SPM or the processor should fail, the system will shut down to prevent a security compromise. The SPM is designed for a mean time between failure of greater than 20,000 hours and a probability of less than .000001 per hour that a hardware failure will result in the undetected loss of secure data protection functions. The SPM is designed for an effective life of 10 years.

- Memory boards are available with either standard parity (16-bit words with 2 parity bits) or error correcting parity (16-bit words with 6 parity bits). A battery backup can maintain 64K-words per unit for 2 hours.

- Power losses longer than three cycles and power out of tolerance conditions are treated as long-term power failures. The power-down cycle time is 1.5 $\mu$s.
- The machine is air cooled. The operating temperature ranges from 0 to 50 degrees centigrade in a range of 5% to 95% relative humidity. The SPM is designed to operate in up to 100% humidity.

## REFERENCES

E1. Detail Specification, Parts I and II, The Security Protection Module. DS34025843, Honeywell Incorporated, 1978.

E2. Series 60 (Level 6) Minicomputer Handbook, Addendum A. AS22A, Rev. 2, Honeywell Incorporated, 1978.